TERASOLUNA Server Framework for Java (5.x) Development Guideline Documentation

Release 5.3.0.RELEASE

NTT DATA Corporation

May 24, 2017

CONTENTS

1 In the Beginning 3

1.1 Terms of Use . . . 3
1.2 Reference . . . 4
1.2.1 Macchinetta Terms of Use . . . 4
1.3 This document covers the following . . . 6
1.4 Target readers of this document . . . 6
1.5 Structure of this document . . . 6
1.6 Reading this document . . . 7
1.7 Tested environments of this document . . . 8
1.8 Criteria based mapping of guideline . . . 9
1.8.1 Mapping based on security measures . . . 9
A point of view by OWASP(Open Web Application Security Project) . . . 9
A point of view by CVE(Common Vulnerabilities and Exposures) . . . 10
1.9 Change Log . . . 13

2 Architecture Overview 45

2.1 Stack of TERASOLUNA Server Framework for Java (5.x) . . . 45
2.1.1 Summary of Software Framework of TERASOLUNA Server Framework for Java (5.x) . . . 45
2.1.2 Main Structural Elements of Software Framework . . . 45
DI Container . . . 45
MVC Framework . . . 46
O/R Mapper . . . 46
View . . . 47
Security . . . 47
Validation . . . 47
Logging . . . 47
Common Library . . . 48
2.1.3 OSS Versions . . . 48
2.1.4 Building blocks of Common Library . . . 51
terasoluna-gfw-common . . . 55
terasoluna-gfw-string . . . 55
terasoluna-gfw-codepoints . . . 55
terasoluna-gfw-validator . . . 56

terasoluna-gfw-jodatime . . . 56
terasoluna-gfw-web . . . 56
terasoluna-gfw-web-jsp . . . 57
terasoluna-gfw-security-web . . . 58
2.2 Overview of Spring MVC Architecture . . . 59
2.2.1 Overview of Spring MVC Processing Sequence . . . 59
2.2.2 Implementation of each component . . . 60
Implementation of HandlerMapping . . . 60
Implementation of HandlerAdapter . . . 61
Implementation of ViewResolver . . . 61
Implementation of View . . . 62
2.3 First application based on Spring MVC . . . 64
2.3.1 Prerequisites . . . 64
2.3.2 Create a New Project . . . 64
2.3.3 Run on Server . . . 72
2.3.4 Create an Echo Application . . . 74
Creating a form object . . . 74
Create a Controller . . . 75
Create JSP Files . . . 78
Implement Input Validation . . . 80
Summary . . . 84
2.4 Application Layering . . . 85
2.4.1 Layer definition . . . 85
Application layer . . . 86
Controller . . . 86
View . . . 86
Form . . . 87
Helper . . . 87
Domain layer . . . 88
DomainObject . . . 88
Repository . . . 89
Service . . . 89
Infrastructure layer . . . 89
RepositoryImpl . . . 90
O/R Mapper . . . 90
Integration System Connector . . . 90
2.4.2 Dependency between layers . . . 91
Processing and data flow with Repository . . . 92
Processing and data flow without Repository . . . 94
2.4.3 Project structure . . . 95
[projectName]-domain . . . 96

[projectName]-web . . . 97
[projectName]-env . . . 99

3 Application Development 103
3.1 Create Web application development project . . . 103
3.1.1 Types of Blank Project . . . 103
3.1.2 Create development project . . . 104
3.1.3 Customization of development project . . . 107
POM file project information . . . 108
x.xx.fw.9999 format message ID . . . 110
Message wording . . . 111
Error screen . . . 112
Screen footer copyright . . . 113
In-memory database (H2 Database) . . . 114
DataSource configuration . . . 115
3.1.4 Structure of the development project . . . 118
Multi-project structure . . . 120
Structure of Web module . . . 124
Structure of Domain module . . . 130
Structure of Env module . . . 133
Structure of Initdb module . . . 135
Structure of Selenium module . . . 136
3.1.5 Appendix . . . 137
Hierarchical structure of the project . . . 137
Relationship of bean definition file and application context structure . . . 140
Description of the configuration file . . . 142
Application development in offline environment . . . 142
3.2 Domain Layer Implementation . . . 145
3.2.1 Roles of domain layer . . . 145
3.2.2 Flow of development of domain layer . . . 145
3.2.3 Implementation of Entity . . . 148
Policy of creating Entity class . . . 148
Example of creating Entity class . . . 149
Table structure . . . 149
Entity structure . . . 152
3.2.4 Implementation of Repository . . . 156
Roles of Repository . . . 156
Structure of Repository . . . 156
Creation of Repository . . . 160
Example of creating Repository . . . 160
Structure of Repository . . . 161
Definition of Repository interface . . . 161

Creation of Repository interface . . . 161
Method definition of Repository interface . . . 164
Creation of RepositoryImpl . . . 168
3.2.5 Implementation of Service . . . 168
Roles of Service . . . 168
Structure of Service class . . . 170
Reason for separating Service and SharedService . . . 172
Reason for prohibiting the calling of other Service classes from Service class . . . 172
Regarding interface and base classes to limit signature of method . . . 173
Patterns of creating service class . . . 177
Image of Application development - Creating Service for each Entity . 180
Image of Application development - Creating Service for each use case 181
Image of Application development - Creating Service for each use event 184
Creation of Service class . . . 188
Methods of creating Service class . . . 188
Creation of methods of Service class . . . 190
Regarding arguments and return values of methods of Service class . . 192
Implementation of SharedService class . . . 193
Creation of SharedService class . . . 193
Creation of SharedService class method . . . 194
Regarding arguments and return values of SharedService class method 194
Implementation of logic . . . 194
Operate on business data . . . 194
Returning messages . . . 194
Returning warning message . . . 195
Notifying business error . . . 197
Notifying system error . . . 198
3.2.6 Regarding transaction management . . . 200
Method of transaction management . . . 200
Declarative transaction management . . . 200
Information required for “Declarative transaction management” . . . 200
Propagation of transaction . . . 206
Way of calling the method which is under transaction control . . . 209
Settings for using transaction management . . . 210
PlatformTransactionManager settings . . . 210
Settings for enabling @Transactional . . . 211
Regarding attributes of <tx:annotation-driven> element . . . 212
3.2.7 Tips . . . 213
Method of dealing with violation of business rules as field error . . . 213
3.3 Implementation of Infrastructure Layer . . . 214

3.3.1 Implementing RepositoryImpl . . . 214
Implementing Repository using MyBatis3 . . . 214
Implementing Repository using JPA . . . 216
Implementing Repository to link with external system using RestTemplate . . . 217
3.4 Implementation of Application Layer . . . 218
3.4.1 Implementing Controller . . . 218
Creating Controller class . . . 220
Mapping request and handler method . . . 220
Mapping with request path . . . 223
Mapping by HTTP method . . . 223
Mapping by request parameter . . . 224
Mapping using request header . . . 225
Mapping using Content-Type header . . . 225
Mapping using Accept header . . . 225
Mapping request and handler method . . . 225
Overview of sample application . . . 226
Request URL . . . 228
Mapping request and handler method . . . 228
Implementing form display . . . 232
Implementing the display of user input confirmation screen . . . 234
Implementing ‘redisplay of form’ . . . 237
Implementing ‘create new user’ business logic . . . 240
Implementing notification of create new user process completion . . . 241
Placing multiple buttons on HTML form . . . 243
Source code of controller of sample application . . . 244
Regarding arguments of handler method . . . 245
Passing data to screen (View) . . . 247
Retrieving values from URL path . . . 249
Retrieving request parameters individually . . . 251
Retrieving request parameters collectively . . . 253
Performing input validation . . . 255
Passing data while redirecting request . . . 256
Passing request parameters to redirect destination . . . 259
Inserting values in redirect destination URL path . . . 260
Acquiring values from Cookie . . . 261
Writing values in Cookie . . . 262
Retrieving pagination information . . . 263
Retrieving uploaded file . . . 263
Displaying result message on the screen . . . 263
Regarding return value of handler method . . . 264
HTML response . . . 264

Responding to downloaded data . . . 266
Implementing the process . . . 269
Correlation check of input value . . . 269
Calling business logic . . . 270
Reflecting values to domain object . . . 271
Reflecting values to form object . . . 273
3.4.2 Implementing form object . . . 275
Creating form object . . . 276
Number format conversion of fields . . . 277
Date and time format conversion of fields . . . 278
DataType conversion in controller . . . 279
Specifying annotation for input validation . . . 279
Initializing form object . . . 280
Binding to HTML form . . . 282
Binding request parameters . . . 282
Determining binding result . . . 284
3.4.3 Implementing View . . . 284
Implementing JSP . . . 285
Creating common JSP for include . . . 286
Displaying value stored in model . . . 288
Displaying numbers stored in model . . . 289
Displaying date and time stored in model . . . 290
Generate request URL . . . 290
Binding form object to HTML form . . . 294
Displaying input validation errors . . . 294
Displaying message of processing result . . . 295
Displaying codelist . . . 295
Displaying fixed text content . . . 296
Switching display according to conditions . . . 297
Repeated display of collection elements . . . 298
Displaying link for pagination . . . 299
Switching display according to authority . . . 299
Implementing JavaScript . . . 300
Implementing style sheet . . . 300
3.4.4 Implementing common logic . . . 301
Implementing common logic to be executed before and after calling controller . 301
Implementing Servlet Filter . . . 301
Implementing HandlerInterceptor . . . 303
Implementing common processes of controller . . . 305
Implementing HandlerMethodArgumentResolver . . . 305
Implementing “@ControllerAdvice” . . . 307

3.4.5 Prevention of double submission . . . 315
3.4.6 Usage of session . . . 315
3.5 Build development project . . . 317
3.5.1 Build development project . . . 317
Build method wherein jar file of env module is not included in war file . . . 319
Create war file . . . 319
Create jar file of env module . . . 320
Build method wherein jar file of env module is included in war file . . . 321
Create war file . . . 321
Deploy . . . 322
Deploy on Tomcat . . . 322
Deployment to other application server . . . 323
Continuous deployment . . . 325

4 Web Application Development Features 331
4.1 Input Validation . . . 331
4.1.1 Overview . . . 331
Classification of input validation . . . 332
4.1.2 How to use . . . 332
Adding dependent libraries . . . 332
Single item check . . . 333
Basic single item check . . . 333
Date and time format check . . . 345
Single item check of nested Bean . . . 346
Grouped validation . . . 357
Correlation item check . . . 370
Correlation item check implementation using Spring Validator . . . 370
Implementation of input check of correlated items using Bean Validation 378
Definition of error messages . . . 378
Messages to be defined in ValidationMessages.properties . . . 380
Messages to be defined in application-messages.properties . . . 384
4.1.3 How to extend . . . 385
Creation of Bean Validation annotation by combining existing rules . . . 386
How to check the values in the collection using Bean Validation's annotation . . . 391
Implementation with Java SE 8 and Hibernate Validator 5.2+ . . . 391
Implementation of wrapper class of String using Java bean . . . 395
Creation of Bean Validation annotation by implementing new rules . . . 399
Rules that cannot be implemented by combining the existing rules . . . 400
Check rules for correlated items . . . 402
Business logic check . . . 408
Method Validation . . . 410
Application settings . . . 410

How to define for the method for Method Validation target 413
Exception handling at the time of violation of constraint 417
4.1.4 Appendix 419
Input validation rules provided by Hibernate Validator 419
Bean Validation check rules 419
Hibernate Validator check rules 423
Default messages provided by Hibernate Validator 425
Input check rules provided by a common library 426
terasoluna-gfw-common check rules 426
terasoluna-gfw-codepoints check rules 427
terasoluna-gfw-validator check rules 427
How to apply check rules of common library 433
How to extend check rules of common library 434
Type mismatch 436
Binding null to blank string field 438
Reading messages which are not converted from Native to Ascii 439
OS Command Injection Countermeasures 441
What is OS command injection - 441
Countermeasures 441
4.2 Exception Handling 443
4.2.1 Overview 443
Classification of exceptions 443
Exception handling methods 444
4.2.2 Detail 449
Types of exceptions 449
Business exception 451
Library exceptions that occurs during normal operation 451
System Exception 452
Unexpected System Exception 453
Fatal Errors 453
Framework exception in case of invalid requests 454
Exception Handling Patterns 454
When notifying partial redo of a use case (from middle) 457
When notifying redo of a use case (from beginning) 457
When notifying that the system or application is not in a normal state . 458
When notifying that the request contents are invalid 459
When a fatal error has been detected 460
When notifying that an exception has occurred in the presentation layer (JSP etc.) 461
Basic Flow of Exception Handling 462

Basic flow when the Controller class handles the exception at request level . . . 462
Basic flow when the Controller class handles the exception at use case level . . . 463
Basic flow when the framework handles the exception at servlet level . 464
Basic flow when the servlet container handles the exception at web application level . . . 465
4.2.3 How to use . . . 466
Application Settings . . . 467
Common Settings . . . 467
Domain Layer Settings . . . 473
Application Layer Settings . . . 474
Servlet Container Settings . . . 481
Coding Points (Service) . . . 483
Generating Business Exception . . . 483
Generating System Exception . . . 487
Catch the exception to continue the execution . . . 489
Coding Points (Controller) . . . 491
Method to handle exceptions at request level . . . 491
Method to handle exception at use case level . . . 492
Coding points (JSP) . . . 494
Method to display messages on screen using MessagesPanelTag . . . 494
Method to display system exception code on screen . . . 495
4.2.4 How to use (Ajax) . . . 496
4.2.5 Appendix . . . 496
Exception handling classes provided by the common library . . . 497
About SystemExceptionResolver settings . . . 503
Attribute name of result message . . . 505
Attribute name of exception code (message ID) . . . 506
Header name of exception code (message ID) . . . 507
Attribute name of exception object . . . 507
Cache control flag of HTTP response . . . 508
About HandlerExceptionResolverLoggingInterceptor settings . . . 509
List of exception classes to be excluded from scope of logging . . . 509
HTTP response code set by DefaultHandlerExceptionResolver . . . 510
4.3 Session Management . . . 513
4.3.1 Overview . . . 513
Session lifecycle . . . 516
Generating a session . . . 516
Storing attributes in a session . . . 518
Deleting attributes from a session . . . 520

Discarding a session . . . 521
Detecting a request after session timeout . . . 523
About using a session . . . 525
Advantages and disadvantages of using session . . . 526
Advantages and disadvantages of not using session . . . 527
About the data to be stored in session . . . 528
Serializable objects . . . 528
Amount of data to be stored in session . . . 528
Points to be considered in case of application server clustering . . . 529
About storage location of session . . . 530
4.3.2 How to use . . . 530
Using `@SessionAttributes` annotation . . . 530
Specifying the object to be stored in session . . . 531
Adding object to session . . . 533
Fetching the object stored in session . . . 535
Deleting the object stored in session . . . 539
Process implementation using @SessionAttributes . . . 543
Using session-scoped bean of Spring Framework . . . 543
Bean definition of session scope . . . 543
Using session-scoped bean . . . 546
Deleting objects stored in session . . . 548
Process implementation using session-scoped bean . . . 549
Debug log output of session operations . . . 549
Using the JSP implicit object `sessionScope` . . . 549
4.3.3 How to extend . . . 550
Synchronizing requests in same session . . . 550
4.3.4 Appendix . . . 551
Example of screen transition implementation using `@SessionAttributes` in wizard format . . . 551
Example of screen transition across multiple Controllers using session-scoped bean. . . . 567
4.4 Pagination . . . 577
4.4.1 Overview . . . 577
Display of list screen at the time of dividing into pages . . . 577
Page search . . . 579
Page search functionality of Spring Data . . . 579
Display of pagination link . . . 583
Structure of pagination link . . . 583
HTML of pagination link . . . 585
Parameters of JSP tag library . . . 591
Process flow when pagination is used . . . 600

4.4.2 How to use . . . 603
Application settings . . . 603
Settings for enabling pagination functionality of Spring Data . . . 603
Page search . . . 604
Implementation of application layer . . . 604
Implementation of domain layer (MyBatis3) . . . 610
Implementation of domain layer (JPA) . . . 610
Implementation of JSP (Base version) . . . 610
Display of fetched data . . . 610
Display of Pagination link . . . 613
Display of pagination information . . . 618
Carrying forward search conditions using page link . . . 620
Carrying forward the sort condition using page link . . . 623
Implementation of JSP (layout change) . . . 623
Removal of link to navigate to the first page and the last page . . . 623
Removal of link to navigate to previous page and next page . . . 624
Removal of disabled link . . . 625
Change in maximum number of display links to navigate to the specified page . . . 626
Removal of link to navigate to the specified page . . . 627
Implementation of JSP (Operation) . . . 628
Specifying sort condition . . . 628
To disable page link using JavaScript . . . 629
4.4.3 Appendix . . . 630
About property values of `PageableHandlerMethodArgumentResolver` 630
Property value of `SortHandlerMethodArgumentResolver` . . . 635
4.5 Double Submit Protection . . . 636
4.5.1 Overview . . . 636
Problems . . . 636
Double clicking of 'Update' button . . . 636
Reloading of screen after completion of update process . . . 638
Invalid screen transition using 'Back' button of the browser . . . 639
Solutions . . . 643
Preventing double clicking of a button using JavaScript . . . 645
About PRG (Post-Redirect-Get) pattern . . . 645
Transaction Token Check . . . 648
About NameSpace of transaction token . . . 656
Problems that occur when there is no NameSpace . . . 657
Behavior when NameSpace is specified . . . 658
4.5.2 How to use . . . 659
Preventing double clicking of button using JavaScript . . . 659

Using PRG (Post-Redirect-Get) pattern . . . 660
Using transaction token check . . . 664
Transaction token check provided by common library . . . 664
Attributes of `@TransactionTokenCheck` annotation . . . 664
Format of transaction token . . . 667
Lifecycle of transaction token . . . 671
Settings for using a transaction token check . . . 676
Settings for handling transaction token errors . . . 677
How to use transaction token check in Controller . . . 678
How to use transaction token check in View (JSP) . . . 679
When multiple usecases are to be implemented in one Controller . . . 682
When a process which does not update screen for file downloading process is included in the use case . . . 685
Typical example of using transaction token check . . . 690
Exclusion control of parallel processing while using a session . . . 693
4.5.3 How to extend . . . 693
How to change the maximum limit of transaction tokens . . . 693
4.5.4 Appendix . . . 694
Transaction Token Check in case that the cache of browser is disabled . . . 694
Global Tokens . . . 695
Changing the maximum limit of transaction tokens that can be stored for each NameSpace . . . 695
Implementation of Controller . . . 696
Quick Reference . . . 700
4.6 Message Management . . . 707
4.6.1 Overview . . . 707
Types of messages . . . 707
Types of messages depending on patterns . . . 708
Message ID . . . 710
Title . . . 710
Labels . . . 711
Result messages . . . 713
Input validation error message . . . 717
4.6.2 How to use . . . 717
Display of messages set in properties file . . . 717
Settings at the time of using properties . . . 717
Display of messages set in properties . . . 718
Display of result messages . . . 720
Using basic result messages . . . 720
Specifying attribute name of result messages . . . 728
Displaying business exception messages . . . 730

4.6.3 How to extend . . . 732
Creating independent message types . . . 732
4.6.4 Appendix . . . 733
Changing attribute of <t:messagesPanel> tag . . . 733
Display of result message wherein ResultMessages is not used . . . 736
Auto-generation tool of message key constant class . . . 739
4.7 Internationalization . . . 742
4.7.1 Overview . . . 742
4.7.2 How to use . . . 743
Configuration to define messages . . . 743
Changing locale as per browser settings . . . 745
Settings of AcceptHeaderLocaleResolver . . . 745
Definition of messages . . . 746
Implementation of JSP . . . 746
Changing locale depending on screen operations dynamically . . . 747
How to define LocaleChangeInterceptor . . . 748
How to define SessionLocaleResolver . . . 750
How to define CookieLocaleResolver . . . 750
Messages settings . . . 751
Implementation of JSP . . . 752
Measures when Internationalization is not applied . . . 753
4.8 Codelist . . . 756
4.8.1 Overview . . . 756
4.8.2 How to use . . . 757
Using SimpleMapCodeList . . . 758
Example of codelist settings . . . 758
Using codelist in JSP . . . 760
Using codelist in Java class . . . 762
Using NumberRangeCodeList . . . 763
Example of codelist settings . . . 764
Using codelist in JSP . . . 765
Using codelist in Java class . . . 766
Using JdbcCodeList . . . 766
Example of codelist settings . . . 767
Using codelist in JSP . . . 771
Using codelist in Java class . . . 772
How to use EnumCodeList . . . 772
Example of codelist settings . . . 773
Using codelist in JSP . . . 776
Using codelist in Java class . . . 776
How to use SimpleI18nCodeList . . . 776

Example of codelist settings . . . 777
Using codelist in JSP . . . 782
Using codelist in Java class . . . 784
Display code name corresponding to code value . . . 784
Input validation of code value using codelist . . . 785
Example of @ExistInCodeList settings . . . 786
4.8.3 How to extend . . . 787
When reloading the codelist . . . 787
Using Task Scheduler . . . 788
Calling refresh method in Controller (Service) class . . . 789
Customizing the codelist independently . . . 791
4.8.4 Appendix . . . 794
Setting SimpleI18nCodeList . . . 794
Set `java.util.Map` (key = code value, value = label) for each locale by rows . . . 794
Set `java.util.Map`(key = locale, value = label) for each code value by columns . . . 795
Variations of NumberRangeCodeList . . . 796
Create the Descending NumberRangeCodeList . . . 796
Change interval of NumberRangeCodeList . . . 798
4.9 File Upload . . . 800
4.9.1 Overview . . . 800
Basic flow of upload process . . . 801
About classes provided by Spring Web . . . 804
4.9.2 How to use . . . 806
Application settings . . . 806
Settings to enable Servlet 3.0 upload functionality . . . 806
Servlet Filter settings . . . 810
Settings for exception handling . . . 812
Uploading a single file . . . 815
Implementing form . . . 816
Implementing JSP . . . 816
Implementing Controller . . . 817
Bean Validation of file upload . . . 821
Implementing validation to verify that the file is selected . . . 821
Implementing validation to verify that the file is not empty . . . 822
Implementing validation to verify that file size is within allowable range 823
Implementing form . . . 825
Implementing Controller . . . 825
Uploading multiple files . . . 826
Implementing form . . . 826

Implementing JSP . . . 827
Implementing Controller . . . 828
Uploading multiple files using the “multiple” attribute of HTML5 . . . 829
Implementing form . . . 830
Implementing Validator . . . 830
Implementing JSP . . . 832
Implementing Controller . . . 833
Temporary upload . . . 833
Implementing Controller . . . 836
4.9.3 How to extend . . . 838
Housekeeping of unnecessary files at the time of temporary upload . . . 838
Implementing component class to delete unnecessary files . . . 839
Scheduling settings of the process for deleting unnecessary files . . . 840
4.9.4 Appendix . . . 842
Security issues related to file upload . . . 842
Dos attack with respect to upload functionality . . . 842
Attack by executing uploaded files on Web Server . . . 843
Directory traversal attack . . . 843
File upload using Commons FileUpload . . . 844
4.10 File Download . . . 848
4.10.1 Overview . . . 848
4.10.2 How to use . . . 849
Downloading PDF files . . . 849
Implementation of Custom View . . . 849
Definition of ViewResolver . . . 851
Specifying View in controller . . . 853
Downloading Excel files . . . 853
Implementation of Custom View . . . 854
Definition of ViewResolver . . . 856
Specifying View in controller . . . 856
Downloading arbitrary files . . . 857
Implementation of Custom View . . . 857
Definition of ViewResolver . . . 858
Specifying View in controller . . . 858
4.11 Screen Layout using Tiles . . . 860
4.11.1 Overview . . . 860
4.11.2 How to use . . . 862
pom.xml setting . . . 862
Integration of Spring MVC and Tiles . . . 862
4.11.3 How to extend . . . 872
Setting multiple layouts . . . 872

4.12 JSP Tag Libraries and EL Functions offered by common library . . . 881
4.12.1 Overview . . . 881
JSP Tag Library . . . 881
EL Functions . . . 882
4.12.2 How to use . . . 883
<t:pagination> . . . 883
<t:messagesPanel> . . . 883
<t:transaction> . . . 884
f:h() . . . 884
f:h() function specification . . . 884
f:h() how to use . . . 885
f:js() . . . 885
f:js() function specification . . . 885
f:js() how to use . . . 886
f:hjs() . . . 886
f:hjs() function specification . . . 886
f:hjs() how to use . . . 886
f:query() . . . 887
f:query() function specification . . . 887
f:query() how to use . . . 890
f:u() . . . 890
f:u() function specification . . . 890
f:u() how to use . . . 890
f:link() . . . 891
f:link() function specification . . . 891
f:link() how to use . . . 891
f:br() . . . 892
f:br() function specification . . . 893
f:br() how to use . . . 893
f:cut() . . . 894
f:cut() function specification . . . 894
f:cut() how to use . . . 894
4.13 Ajax . . . 895
4.13.1 Overview . . . 895
4.13.2 How to use . . . 895
Application settings . . . 895
Settings to enable the Ajax functionality in Spring MVC . . . 896
Implementing Controller . . . 900
Fetching data . . . 901
Posting form data . . . 908
Posting form data in JSON format . . . 916

Input error handling . . . 919
Handling BindException . . . 919
Handling MethodArgumentNotValidException . . . 923
Handling HttpMessageNotReadableException . . . 924
Handling by using BindingResult . . . 925
Business error handling . . . 927
Handling business exception by exception handling method . . . 927
Handling business exception in handler method . . . 928
4.14 Health Check . . . 930
4.14.1 Overview . . . 930
Load sharing and fallback of load balancer . . . 930
Types of health checks . . . 930
Structure of health check shown in the guideline . . . 933
4.14.2 How to use . . . 935
Repository interface . . . 935
Service class . . . 936
Controller class . . . 937
JSP file . . . 938
Configuring access rights . . . 938
4.14.3 Appendix . . . 939
Configuration for minimising the data volume for response . . . 939
Should not receive settings of Apache Tiles . . . 939
Prevent reading of header and footer files . . . 939
Remove additional line breaks from response . . . 940
5 Web Services 943
5.1 RESTful Web Service . . . 943
5.1.1 Overview . . . 943
What is RESTful Web Service . . . 943
RESTful Web Service development . . . 945
Configuration for RESTful Web Service module . . . 949
REST API implementation sample . . . 951
5.1.2 Architecture . . . 956
Publishing as a resource on Web . . . 958
Identifying the resource using URI . . . 958
Resource operations using HTTP methods . . . 959
Using an appropriate format . . . 963
Using the appropriate HTTP status code . . . 964
Stateless communication between client and server . . . 965
Link to related resource . . . 966
5.1.3 How to design . . . 973
Resource extraction . . . 973

Assigning URI . . . 974
Assigning a URI that indicates the API as REST API . . . 974
Assigning a URI for identifying the API version . . . 974
Assigning a path for identifying resource . . . 975
Assigning HTTP methods . . . 976
Assigning HTTP methods for resource collection URI . . . 977
Assigning HTTP methods for URI of specific resources . . . 978
Resource format . . . 978
JSON Field name . . . 978
NULL and blank characters . . . 979
Date format . . . 979
Hypermedia link format . . . 980
Format at the time of error response . . . 981
HTTP Status Code . . . 981
HTTP status codes when the request is successful . . . 982
HTTP status code when the cause of request failure lies at client side . 983
HTTP status code when the cause of request failure lies at server side . 985
Authentication and Authorization . . . 986
Conditional update control of resource . . . 986
Conditional acquisition control of resource . . . 986
Cache control of resource . . . 987
Versioning . . . 987
5.1.4 How to use . . . 988
Web application configuration . . . 988
pom.xml configuration . . . 990
Application settings . . . 991
Settings for activating the Spring MVC components necessary for RESTful Web Service . . . 991
Servlet settings for RESTful Web Service . . . 996
REST API implementation . . . 998
Creating REST API packages . . . 1003
Creating Resource class . . . 1004
Creating Controller class . . . 1010
Implementing REST API that fetches collection of resources . . . 1011
Implementing REST API that adds a resource to collection . . . 1019
Implementing REST API that fetches specified resource . . . 1022
Implementing REST API that updates specified resource . . . 1024
Implementing REST API that deletes specified resource . . . 1027
Implementing exception handling . . . 1029
Implementation to output error information in response Body . . . 1031
Implementing input error exception handling . . . 1037

Implementing exception handling for “Resource not found” error . . . 1044
Implementing exception handling for business errors . . . 1048
Implementing exception handling for exclusive errors . . . 1049
Implementing exception handling for system errors . . . 1050
Resolving error codes and messages using ExceptionCodeResolver . . . 1052
Implementing the error handling notified to Servlet Container . . . 1056
Implementing the Controller that sends error response . . . 1058
Creating a static JSON file to be sent as response when a fatal error occurs . . . 1061
Settings for handling an error that is notified to Servlet Container . . . 1061
Security measures . . . 1063
Authentication and Authorization . . . 1063
CSRF measures . . . 1063
Conditional operations for resource . . . 1064
Cache control for resource . . . 1064
5.1.5 How to extend . . . 1064
Output control for response using @JsonView . . . 1064
5.1.6 Appendix . . . 1073
Configuration while using JSR-310 Date and Time API / Joda Time . . . 1073
Settings when RESTful Web Service and client application are operated as the same Web application . . . 1074
How to set `DispatcherServlet` for RESTful Web Service . . . 1074
Implementing hypermedia link . . . 1077
Implementing common parts . . . 1077
Implementation for each resource . . . 1079
Creating RESTful Web Service conforming to HTTP specifications . . . 1082
Settings for location header at the time of POST . . . 1082
Implementation for each resource . . . 1083
Disabling CSRF measures . . . 1086
Enabling XXE Injection measures . . . 1087
How to copy Joda-Time classes using Dozer . . . 1090
Source code for application layer . . . 1093
MemberRestController.java . . . 1094
ApiErrorCreator.java . . . 1097
ApiGlobalExceptionHandler.java . . . 1098
Source code of the domain layer class created at the time of REST API implementation . . . 1101
Member.java . . . 1103
MemberCredentia.java . . . 1106
Gender.java . . . 1108
MemberRepository.java . . . 1109

MemberService.java . . . 1110
MemberServiceImpl.java . . . 1110
DomainMessageCodes.java . . . 1113
GenderTypeHandler.java . . . 1114
member-mapping.xml . . . 1115
mybatis-config.xml . . . 1116
MemberRepository.xml . . . 1117
5.2 REST Client (HTTP Client) . . . 1122
5.2.1 Overview . . . 1122
What is `RestTemplate` . . . 1122
`HttpMessageConverter` . . . 1124
`ClientHttpRequestFactory` . . . 1130
`ResponseErrorHandler` . . . 1132
`ClientHttpRequestInterceptor` . . . 1132
5.2.2 How to use . . . 1133
`RestTemplate` Setup . . . 1133
Dependent library setup . . . 1133
Bean definition of `RestTemplate` . . . 1134
Using `RestTemplate` . . . 1135
Sending GET request . . . 1135
Implementation by using `getForObject` method . . . 1136
Implementation by using `getForEntity` method . . . 1136
Implementation by using `exchange` method . . . 1137
Sending POST request . . . 1138
Implementation by using `postForObject` method . . . 1139
Implementation using `postForEntity` method . . . 1139
Implementation using `exchange` method . . . 1140
Fetch data in collection format . . . 1141
Configuration of request header . . . 1141
Configuration of Content-Type header . . . 1142
Configuration of Accept header . . . 1142
Configuration of an arbitrary request header . . . 1143
Error Handling . . . 1143
Exception Handling (Default Behaviour) . . . 1143
Returning `ResponseEntity` (Error handler extension) . . . 1145
Setting communication timeout . . . 1147
Using SSL self-signed certificate . . . 1148
Basic authentication . . . 1151
File upload (multi-part request) . . . 1152
File download . . . 1153
How to handle RESTful URL (URI template) and implementation example . . . 1155

5.2.3 How to extend . . . 1156
How to register an arbitrary `HttpMessageConverter` . . . 1156
Application of common process (`ClientHttpRequestInterceptor`) . . . 1157
Logging process . . . 1157
Process to configure a request header for Basic authentication . . . 1159
Applying `ClientHttpRequestInterceptor` . . . 1160
Asynchronous request . . . 1161
Bean definition for `AsyncRestTemplate` . . . 1161
Implementation of asynchronous request . . . 1162
Implementation of common processing for asynchronous request . . . 1163
5.2.4 Appendix . . . 1166
How to configure HTTP Proxy server . . . 1166
How to specify a HTTP Proxy server . . . 1167
How to specify credentials information of HTTP Proxy server . . . 1169
Configuration while using JSR-310 Date and Time API in JSON . . . 1172
5.3 SOAP Web Service (Server/Client) . . . 1173
5.3.1 Overview . . . 1173
SOAP . . . 1173
JAX-WS . . . 1174
JAX-WS linkage function of Spring Framework . . . 1175
Development of Web service using JAX-WS . . . 1178
Configuration of Web service module which uses JAX-WS . . . 1178
URL to be published as Web service . . . 1180
5.3.2 How to use . . . 1182
How to create SOAP server . . . 1182
Project configuration . . . 1182
Application configuration . . . 1184
Web service implementation . . . 1186
Input check implementation . . . 1193
Security measures . . . 1195
Implementation of exception handling . . . 1198
How to handle large binary data using MTOM . . . 1208
Creation of client . . . 1211
Project configuration . . . 1211
Implementation of Web service client . . . 1212
Security measures . . . 1217
Implementing exception handling . . . 1218
Timeout configuration . . . 1219
5.3.3 Appendix . . . 1221
Project configuration is changed for SOAP server. . . . 1221
Changing existing project . . . 1221

Creating model project . . . 1222
Creating webservice project . . . 1223
Package configuration of SOAP server . . . 1225
[server projectName]-domain . . . 1225
[server projectName]-web . . . 1226
[server projectName]-env . . . 1229
[server projectName]-model . . . 1229
[server projectName]-webservice . . . 1230
Package configuration of client . . . 1230
[client projectName]-domain . . . 1231
[client projectName]-web . . . 1232
[client projectName]-env . . . 1232
wsimport . . . 1233
Using wsimport . . . 1234
How to use wsimport . . . 1234
Web service development on Tomcat . . . 1236
Configuration while using CXFServlet . . . 1236
Configuration required in POJO method . . . 1238
Configuration required for the method that inherits SpringBeanAutowiringSupport . . . 1240

6 Data Access 1243
6.1 Database Access (Common) . . . 1243
6.1.1 Overview . . . 1243
About JDBC DataSource . . . 1244
JDBC datasource provided by Application Server . . . 1244
JDBC datasource provided by OSS/Third-Party library . . . 1245
JDBC datasource provided by Spring Framework . . . 1246
About transaction management . . . 1246
About declaration of transaction boundary/attribute . . . 1246
About exclusion control of data . . . 1246
About exception handling . . . 1246
About multiple datasources . . . 1248
About common library classes . . . 1249
6.1.2 How to use . . . 1250
Datasource settings . . . 1250
Settings when using DataSource defined in Application Server . . . 1250
Settings when using DataSource for which Bean is defined . . . 1251
Settings to enable transaction management . . . 1253
JDBC debug log settings . . . 1254
Settings related to datasource provided by log4jdbc . . . 1254
log4jdbc logger settings . . . 1255

Settings of log4jdbc option . . . 1257
6.1.3 How to extend . . . 1257
Settings for using multiple datasources . . . 1257
Settings to switch the datasource dynamically . . . 1258
Implementation of AbstractRoutingDataSource . . . 1258
Datasource definition . . . 1259
6.1.4 how to solve the problem . . . 1260
How to resolve N+1 . . . 1260
Resolving N+1 using JOINs (Join Fetch) . . . 1261
Resolving N+1 by fetching related records in batch . . . 1263
6.1.5 Appendix . . . 1264
Escaping during LIKE search . . . 1264
Specifications of escaping of common library . . . 1265
About escaping methods provided by common library . . . 1267
How to use common library . . . 1268
About Sequencer . . . 1269
About classes provided by common library . . . 1270
How to use common library . . . 1270
Classes provided by Spring Framework for converting to data access exception . 1273
JDBC datasource classes provided by Spring Framework . . . 1274
6.2 Database Access (MyBatis3) . . . 1277
6.2.1 Overview . . . 1277
About MyBatis3 . . . 1277
Component structure of MyBatis3 . . . 1278
Regarding MyBatis3 and Spring integration . . . 1281
Component structure of MyBatis-Spring . . . 1282
6.2.2 How to use . . . 1285
pom.xml settings . . . 1286
Settings for integration of MyBatis3 and Spring . . . 1288
Datasource settings . . . 1288
Transaction control settings . . . 1289
MyBatis-Spring settings . . . 1291
MyBatis3 settings . . . 1292
`fetchSize` settings . . . 1293
SQL execution mode settings . . . 1294
TypeAlias settings . . . 1295
Mapping settings of NULL value and JDBC type . . . 1297
TypeHandler settings . . . 1299
Implementation of database access process . . . 1304
Creating Repository interface . . . 1304
Creating Mapping file . . . 1304

CRUD process implementation . . . 1305
How to map a JavaBean in Search results . . . 1307
Automatic mapping for search results . . . 1307
Manual mapping of search results . . . 1311
Search process for Entity . . . 1314
Fetching a single key Entity . . . 1314
Fetching Entity of composite key . . . 1317
Entity search . . . 1319
Fetching Entity records . . . 1323
Pagination search of Entity (MyBatis3 standard method) . . . 1324
Pagination search for Entity (SQL refinement method) . . . 1330
Entity registration process . . . 1333
Registering a single Entity record . . . 1333
Generating key . . . 1336
Batch registration of Entity . . . 1339
Update process of Entity . . . 1342
Updating a single Entity . . . 1343
Batch update of Entity . . . 1346
Delete process for Entity . . . 1348
Deleting a single Entity . . . 1348
Batch deletion of Entity . . . 1352
Implementing dynamic SQL . . . 1353
Implementation of if element . . . 1354
Implementation of choose element . . . 1355
Implementation of where element . . . 1357
Implementation example for set element . . . 1359
Implementation example of foreach element . . . 1360
Implementation example for bind element . . . 1363
Escape during LIKE search . . . 1364
SQL Injection countermeasures . . . 1365
How to embed using a bind variable . . . 1366
How to embed using a substitution variable . . . 1367
6.2.3 How to extend . . . 1369
Sharing SQL statement . . . 1369
Implementation of TypeHandler . . . 1371
Implementing the TypeHandler for CLOB . . . 1371
Implementing TypeHandler for Joda-Time . . . 1373
Implementation of ResultHandler . . . 1375
Using SQL execution mode . . . 1380
Using PreparedStatement reuse mode . . . 1380
Using batch mode . . . 1381

Precautions when using batch mode Repository 1388
Implementation of a stored procedure 1393
6.2.4 Appendix 1395
Mapper interface mechanism 1395
TypeAlias settings 1400
Configuring TypeAlias per class 1400
Overwriting alias name assigned as default 1401
SQL switching by the database 1402
How to fetch a related Entity by a single SQL 1406
Table layout and data 1407
Entity class diagram 1411
Implementation of Repository interface 1414
SQL implementation 1414
Implementation of mapping 1418
How to fetch a related Entity using a nested SQL 1432
How to fetch a related Entity using a nested SQL 1432
Settings to apply Lazy Load for related Entity 1433
6.3 Database Access (JPA) 1438
6.3.1 Overview 1438
About JPA 1439
O/R Mapping of JPA 1439
Basic JPA terminology 1439
Managing life cycle of entity 1443
About Spring Data JPA 1445
6.3.2 How to use 1449
pom.xml settings 1449
Application Settings 1449
Datasource settings 1449
EntityManager settings 1449
PlatformTransactionManager settings 1454
persistence.xml settings 1454
Settings for validating Spring Data JPA 1455
Settings for using JPA annotations 1458
Settings for converting JPA exception to DataAccessException 1458
OpenEntityManagerInViewInterceptor settings 1458
Creating Repository interface 1462
Inheriting the interface of Spring Data 1463
Inheriting a common project specific interface in which only the required methods are defined 1472
Not inheriting the interface 1474
Adding query method 1475

Defining query method . . . 1476
Specifying query to be executed . . . 1476
Fetching entity lock . . . 1478
Operating the entities of Persistence Layer directly . . . 1479
Setting QueryHints . . . 1481
Specifying a query while calling a query method . . . 1482
Specifying the query using @Query annotation . . . 1482
Specifying with the method name based on naming conventions . . . 1487
Specifying as Named query in Properties file . . . 1490
Implementing the process to search entities . . . 1491
Searching all entities matching the conditions . . . 1492
Searching page of entities matching the conditions . . . 1493
Implementing search process as per the dynamic conditions of entities . . . 1494
Searching all entities matching the dynamic conditions . . . 1495
Page search for the entities matching the dynamic conditions . . . 1503
Implementing the process to fetch entities . . . 1508
Fetching 1 record of entity by specifying ID . . . 1508
Fetching 1 record of entity by specifying conditions other than ID . . . 1510
Adding entities . . . 1512
How to add entities . . . 1513
Adding parent-entity and related-entity . . . 1516
Adding the related-entity . . . 1519
Adding the related-entity directly . . . 1520
Updating entities . . . 1522
How to update entities . . . 1522
Updating the related-entity . . . 1523
Updating the related-entity directly . . . 1523
Updating by using query method . . . 1524
Deleting entities . . . 1524
Deleting parent-entity and related-entity . . . 1524
Deleting the related-entity . . . 1525
Deleting the related-entity directly . . . 1528
Deleting using query method . . . 1529
Escaping at the time of LIKE search . . . 1530
Usage method when type of matching is to be specified in query . . . 1530
Usage method when specifying type of matching in logic . . . 1532
JOIN FETCH . . . 1533
6.3.3 How to extend . . . 1534
How to add custom method . . . 1534
Adding individual custom method to entity specific Repository interface 1535
Adding the custom methods to all Repository interfaces in batch . . . 1537

Storing query fetch results in objects other than entity . . . 1544
Setting Audit properties . . . 1546
Adding common conditions to JPQL to fetch entities from persistence layer . . . 1552
Adding common conditions in JPQL to fetch entities . . . 1552
Adding common conditions to JPQL to fetch the related-entities . . . 1555
How to use multiple PersistenceUnits . . . 1556
How to use Native query . . . 1557
6.4 Exclusive Control . . . 1558
6.4.1 Overview . . . 1558
Necessity of exclusive control . . . 1558
Problem 1 . . . 1558
Problem 2 . . . 1560
Problem 3 . . . 1560
Exclusive control according to the isolation level of transaction . . . 1562
Exclusive control using database locking . . . 1564
Exclusive control using row lock function of the database . . . 1566
Exclusive control using optimistic locking . . . 1571
Exclusive control using pessimistic locking . . . 1573
Prevention of deadlock . . . 1578
Deadlock in table . . . 1578
Deadlock between tables . . . 1580
6.4.2 How to use . . . 1581
How to implement while using MyBatis3 . . . 1581
Row lock function of RDBMS . . . 1581
Optimistic locking . . . 1584
Pessimistic locking . . . 1589
How to implement while using JPA (Spring Data JPA) . . . 1589
Row lock function of RDBMS . . . 1589
Optimistic locking . . . 1591
Pessimistic locking . . . 1595
How to handle an exclusive error . . . 1598
Error handling in case of optimistic locking failure . . . 1598
Error handling in case of pessimistic locking failure . . . 1600

7 General-purpose functions that do not depend on the application 1603
7.1 Logging . . . 1603
7.1.1 Overview . . . 1603
Types of Logs . . . 1604
Output contents of log . . . 1606
Log output points . . . 1607
7.1.2 How to use . . . 1610
Settings of Logback . . . 1610

Basic log output by calling API of SLF4J . . . 1616
Points to be noted for the description of log output . . . 1619
7.1.3 How to extend . . . 1620
Uniform management of log messages . . . 1620
Integration of output format of log messages . . . 1626
Integration with the format of the log output by framework after detecting an exception . . . 1627
Integration to a unique format . . . 1628
7.1.4 Appendix . . . 1630
Using MDC . . . 1630
Basic usage method . . . 1630
Setting value in MDC using Filter . . . 1632
Log output related functionalities provided by common library . . . 1635
HttpSessionEventLoggingListener . . . 1635
TraceLoggingInterceptor . . . 1636
ExceptionLogger . . . 1637
7.2 Properties Management . . . 1638
7.2.1 Overview . . . 1638
7.2.2 How to use . . . 1639
About properties file definition . . . 1639
Using properties in bean definition file . . . 1641
Using properties in Java class . . . 1643
7.2.3 How to extend . . . 1645
Decrypting encrypted values and using them . . . 1646
7.3 Date operations (JSR-310 Date and Time API) . . . 1652
7.3.1 Overview . . . 1652
7.3.2 How to use . . . 1652
Fetch date and time . . . 1654
Fetch by current date and time . . . 1654
Fetch by specifying year, month, day, hours, minutes and seconds . . . 1655
Fetch date and time when time zone is specified . . . 1655
Time period . . . 1657
Fetch time period . . . 1657
Type conversion . . . 1658
Interoperability of each class of Date and Time API . . . 1658
Interoperability with java.util.Date . . . 1660
Interoperability with java.sqlpackage . . . 1660
How to use org.terasoluna.gfw.common.date package . . . 1661
Format for the string . . . 1662
Path from the string . . . 1664
Date operation . . . 1665

Date and time calculation . . . 1665
Date and time comparison . . . 1665
Determination of date and time . . . 1666
Fetching year, month, day, hours, minutes, seconds . . . 1667
Japanese calendar （JapaneseDate） . . . 1668
Fetching Japanese calendar . . . 1668
Format for the string . . . 1668
Path from the string . . . 1669
Conversion of Western and Japanese calendar . . . 1669
7.4 Date Operations (Joda Time) . . . 1669
7.4.1 Overview . . . 1670
7.4.2 How to use . . . 1670
Fetching date . . . 1670
Fetching current time . . . 1670
Fetching current time by specifying the time zone . . . 1671
Fetching the date and time by specifying Year Month Day Hour Minute and Second . . . 1672
Fetching Year Month Day individually . . . 1672
Type conversion . . . 1674
Interoperability with java.util.Date . . . 1674
String format . . . 1674
Parsing from string . . . 1675
Date operations . . . 1675
Date calculations . . . 1675
Fetching first and last day of the month . . . 1676
Fetching the first and the last day of the week . . . 1677
Comparison of date time . . . 1678
Fetching the duration . . . 1678
Interval . . . 1679
Period . . . 1680
JSP Tag Library . . . 1681
How to set . . . 1681
joda:format tag . . . 1681
Example (display of calendar) . . . 1684
7.4.3 Appendix . . . 1688
Japanese calendar operation before Java8 . . . 1688
7.5 System Date . . . 1689
7.5.1 Overview . . . 1689
About the components that are offered by common library . . . 1689
terasoluna-gfw-common . . . 1689
terasoluna-gfw-jodatime . . . 1690

7.5.2 How to use . . . 1691
pom.xml setting . . . 1692
Returning system time of server . . . 1693
Returning the fixed time fetched from DB . . . 1694
Returning time obtained by adding the difference registered in DB to the server system time . . . 1697
Caching and reloading the difference . . . 1700
7.5.3 Testing . . . 1701
Unit Test . . . 1701
Example wherein process changes with date . . . 1704
Integration Test . . . 1707
System Test . . . 1708
Production . . . 1708
7.6 String Processing . . . 1710
7.6.1 Overview . . . 1710
7.6.2 How to use . . . 1710
Trim . . . 1710
Padding, Suppress . . . 1711
Processing of a string considered as a surrogate pair . . . 1711
Fetching string length . . . 1711
Fetch string in the specified range . . . 1712
String split . . . 1713
Full width, half width string conversion . . . 1714
How to apply common library . . . 1714
Conversion to full width string . . . 1715
Conversion to half width string . . . 1715
Creating FullHalfConverter class for which a unique full width and half width character pair definition is registered . . . 1717
Code point set check (character type check) . . . 1719
How to apply common library . . . 1719
Creating code point set . . . 1720
Set operation of the code point sets . . . 1722
String check by using code point set . . . 1724
String check linked with Bean Validation . . . 1725
Creating new code point set class . . . 1726
Code point set class provided by common library . . . 1727
7.7 Bean Mapping (Dozer) . . . 1731
7.7.1 Overview . . . 1731
7.7.2 How to use . . . 1733
Bean definition for using Dozer . . . 1733
Mapping when the field name and the type between Beans is same . . . 1734

Mapping when the field name is the same and type between Beans is different . . 1736
Mapping when the field name between Beans is different 1737
One-way/Two-way mapping 1739
Mapping of nested fields 1740
Collection mapping 1743
7.7.3 How to extend 1752
Creating custom convertor 1752
7.7.4 Appendix 1756
Field exclusion settings (field-exclude) 1756
Specifying mapping (map-id) 1758
Settings to exclude null/empty field of copy source (map-null, map-empty) . . . 1760
Mapping from string to date/time object 1761
Mapping error 1764

8 Messaging 1765
8.1 Sending E-mail (SMTP) 1765
8.1.1 Overview 1765
Regarding JavaMail 1765
Regarding component of Spring Framework for email coordination 1766
8.1.2 How to use 1769
Regarding dependent library 1769
How to configure JavaMailSender 1770
When a mail session offered by application server is used 1770
When a mail session offered by application server is not used (no authentication) 1772
When a mail session offered by application server is not used (authenticated) 1772
How to send an email using SimpleMailMessage 1774
How to send an email using MimeMessage 1777
Sending a text email 1777
Sending a HTML email 1779
Sending an email with attachment 1780
Sending an email with inline resource 1782
Regarding exceptions while sending an email 1785
8.1.3 How to extend 1786
How to create an email text using a template 1786
Creating the email text using FreeMarker 1786
8.1.4 Appendix 1789
Considerations for ISO-2022-JP encoding 1789
Precautions while using multibyte characters in JavaMail 1792
Email header injection countermeasures 1793
Processing method 1793

Send an email based on the email information stored in database or message queue . . . 1793
Test using GreenMail . . . 1794
8.2 JMS(Java Message Service) . . . 1796
8.2.1 Overview . . . 1796
What is JMS . . . 1796
Using JMS . . . 1799
Using JMS which use component of Spring Framework . . . 1800
When messages are sent synchronously . . . 1802
When messages are received asynchronously . . . 1803
When the messages are received synchronously . . . 1804
Regarding project configuration . . . 1805
8.2.2 How to use . . . 1806
Settings that are common for both sending and receiving messages . . . 1806
Configuration of dependent library . . . 1806
Configuration of `ConnectionFactory` . . . 1807
Configuration of `DestinationResolver` . . . 1808
A method to synchronously send messages . . . 1809
Basic synchronous sending . . . 1809
When message header is to be edited and sent synchronously . . . 1817
Transaction control . . . 1818
A method to receive messages asynchronously . . . 1825
Basic asynchronous receiving . . . 1825
Fetching header information of messages . . . 1831
Process results after asynchronous receiving are sent as messages . . . 1832
When messages which are asynchronously received are to be restricted 1834
Input check for the messages which are received asynchronously . . . 1835
Transaction control . . . 1837
Exception handling at the time of asynchronous receiving . . . 1849
A method wherein the messages are synchronously received . . . 1852
8.2.3 Appendix . . . 1857
JMS provider dependent configuration . . . 1857
While using Apache ActiveMQ . . . 1857
Mass-mailing of identical messages . . . 1860
Sending and receiving large size data . . . 1861
While using Apache ActiveMQ . . . 1862

9 Security countermeasures 1867
9.1 Spring Security Overview . . . 1867
9.1.1 Spring Security functions . . . 1867
Basic functions of security countermeasures . . . 1867
Function to strengthen security countermeasures . . . 1868

9.1.2 Spring Security architecture . . . 1868
Spring Security module . . . 1869
Set of framework stack modules . . . 1869
Set of modules used according to the requirements . . . 1869
Test module . . . 1870
Set of related modules used according to the requirements . . . 1870
Framework processing . . . 1871
FilterChainProxy . . . 1872
HttpFirewall . . . 1872
SecurityFilterChain . . . 1872
Security Filter . . . 1873
9.1.3 Spring Security setup . . . 1874
Applying dependent library . . . 1874
Creating a bean definition file . . . 1875
Settings for servlet filter . . . 1877
Settings not to apply security countermeasures . . . 1877
9.2 Authentication . . . 1879
9.2.1 Overview . . . 1879
Authentication process architecture . . . 1879
Authentication Filter . . . 1880
AuthenticationManager . . . 1881
AuthenticationProvider . . . 1882
9.2.2 How to use . . . 1882
Form authentication . . . 1882
Applying form authentication . . . 1884
Default operation . . . 1885
Creating login form . . . 1885
Response when authentication is successful . . . 1888
Default operation . . . 1888
Response when authentication fails . . . 1888
Default operation . . . 1889
DB authentication . . . 1889
Creating UserDetails . . . 1892
Creating UserDetailsService . . . 1897
Applying DB authentication . . . 1900
Password hashing . . . 1901
BCryptPasswordEncoder . . . 1903
Handling authentication event . . . 1906
Authentication successful event . . . 1908
Authentication failure event . . . 1909
Creating event listener . . . 1910

Logout . . . 1910
Applying logout process . . . 1912
Default operation . . . 1913
Response when logout is successful . . . 1914
Default operation . . . 1914
Accessing authentication information . . . 1914
Access from Java . . . 1915
Access from JSP . . . 1916
Coordination between authentication process and Spring MVC . . . 1917
Accessing authentication information . . . 1918
9.2.3 How to extend . . . 1919
Customizing form authentication . . . 1919
Changing authentication path . . . 1919
Change request parameter name which sends credentials . . . 1920
Customising response when authentication is successful . . . 1921
Changing default transition destination . . . 1921
Fixing transition destination . . . 1921
Applying AuthenticationSuccessHandler . . . 1922
Customising response at the time of authentication failure . . . 1923
Changing transition destination . . . 1923
Applying AuthenticationFailureHandler . . . 1923
Customising logout process . . . 1926
Changing logout path . . . 1926
Customising response when logout is successful . . . 1928
Changing transition destination . . . 1928
Applying LogoutSuccessHandler . . . 1928
Customising error message . . . 1929
Message during system error . . . 1929
Input check at the time of authentication . . . 1930
Input check using Bean Validation . . . 1930
Expansion of authentication process . . . 1932
Creating an implementation class of Authentication interface . . . 1933
Creating an implementation class of AuthenticationProvider interface . 1935
Creating Authentication Filter . . . 1937
Modifying login form . . . 1938
Applying expanded authentication process . . . 1939
Using PasswordEncoder of deprecated package . . . 1944
Using ShaPasswordEncoder . . . 1944
9.2.4 Appendix . . . 1947
Receive a request by Spring MVC and display login form . . . 1947
Using Remember Me authentication . . . 1948

9.3 Authorization . . . 1951
9.3.1 Overview . . . 1951
Authorization process architecture . . . 1951
ExceptionTranslationFilter . . . 1953
FilterSecurityInterceptor . . . 1953
AccessDecisionManager . . . 1953
AccessDecisionVoter . . . 1954
9.3.2 How to use . . . 1955
How to describe an access policy . . . 1956
Built-In Common Expressions . . . 1956
Built-In Web Expressions . . . 1958
Using operator . . . 1958
Authorization of Web resource . . . 1959
Applying authorization process . . . 1959
Defining access policy . . . 1960
Authorization for the method . . . 1966
Enabling AOP . . . 1966
Applying authorization process . . . 1967
Defining access policy . . . 1967
Authorization for JSP screen fields . . . 1969
Defining an access policy . . . 1969
Linking with the access policy specified in the Web resource . . . 1970
Storing determination results of authorization process in the variable . 1971
Response at the time of authorization error . . . 1972
AccessDeniedHandler . . . 1973
AuthenticationEntryPoint . . . 1974
Transition destination at the time of authorization error . . . 1976
9.3.3 How to extend . . . 1976
Response at the time of authorization error (Authenticated user) . . . 1977
Applying AccessDeniedHandler . . . 1977
Response at the time of authorization error (Unauthenticated user) . . . 1978
Applying AuthenticationEntryPoint for each request . . . 1978
Role hierarchy . . . 1980
Setting hierarchy relation . . . 1980
Applying authorization process of Web resource . . . 1981
Applying authorization process of method . . . 1982
9.4 Session Management . . . 1984
9.4.1 Overview . . . 1984
Security measures at the time of using a session . . . 1984
Countermeasures for session hijacking attacks . . . 1984
Countermeasures for Session fixation attack . . . 1986

Session management function provided by Spring Security . . . 1986
9.4.2 How to use . . . 1987
Countermeasures for Session hijacking attacks . . . 1987
Disabling URL Rewriting function by Spring Security . . . 1987
Disabling URL Rewriting function by Servlet Container . . . 1988
Applying Session management function . . . 1988
Countermeasures for Session fixation attack . . . 1989
Controlling session lifecycle . . . 1990
Generating a session . . . 1990
Discarding a session . . . 1991
Controlling session timeout . . . 1992
Specifying session timeout . . . 1992
Detection of request with invalid session . . . 1992
Specifying Exclusion path . . . 1993
Controlling multiple logins . . . 1994
Enabling session lifecycle detection . . . 1995
Preventing multiple logins (Pre-measure) . . . 1996
Preventing multiple logins (Post-measure) . . . 1996
9.5 CSRF Countermeasures . . . 1998
9.5.1 Overview . . . 1998
CSRF countermeasures of Spring Security . . . 1999
Target request of token check . . . 2000
9.5.2 How to use . . . 2001
Applying CSRF countermeasures . . . 2001
Linking CSRF token value . . . 2002
Coordination by using Spring MVC . . . 2002
Coordination while using HTML form . . . 2004
Coordination while using Ajax . . . 2005
Controlling transition destination in case of token check error . . . 2006
9.6 Coordinating with browser security countermeasure function . . . 2010
9.6.1 Overview . . . 2010
Security headers supported by default . . . 2011
Cache-Control . . . 2011
X-Frame-Options . . . 2012
X-Content-Type-Options . . . 2012
X-XSS-Protection . . . 2012
Strict-Transport-Security . . . 2013
Content-Security-Policy . . . 2013
Public-Key-Pins . . . 2014
9.6.2 How to use . . . 2015
Applying security header output function . . . 2015

Selecting security header . . . 2015
Specifying options of security header . . . 2018
Output of custom header . . . 2018
Displaying security header for each request pattern . . . 2019
9.7 XSS Countermeasures . . . 2021
9.7.1 Overview . . . 2021
Stored & Reflected XSS Attacks . . . 2021
9.7.2 How to use . . . 2021
Output Escaping . . . 2022
Example of vulnerability when output values are not escaped . . . 2022
Example of escaping output value using f:h() function . . . 2023
JavaScript Escaping . . . 2025
Example of vulnerability when output values are not escaped . . . 2026
Example of escaping output value using f:js() function . . . 2027
Event handler Escaping . . . 2028
Example of vulnerability when output values are not escaped . . . 2028
Example of escaping output value using f:hjs() function . . . 2029
9.8 Encryption . . . 2030
9.8.1 Overview . . . 2030
Encryption method . . . 2030
Common key encryption method . . . 2030
Public key encryption method . . . 2031
Hybrid encryption method . . . 2031
Encryption algorithm . . . 2032
DES / 3DES . . . 2032
AES . . . 2032
RSA . . . 2033
DSA / ECDSA . . . 2033
Pseudo-random number (Generator) . . . 2033
javax.crypto.Cipher class . . . 2033
Encryption function in Spring Security . . . 2034
Components for encryption and decryption . . . 2034
Components to generate random numbers . . . 2034
9.8.2 How to use . . . 2035
Common key encryption method . . . 2035
Encryption of character string . . . 2036
Decryption of string . . . 2038
Encryption of byte array . . . 2039
Decryption of byte array . . . 2040
Public key encryption method . . . 2041
Preliminary preparation (Generation of key pairs using JCA) . . . 2041

Encryption . . . 2043
Decryption . . . 2044
OpenSSL . . . 2044
Hybrid encryption method . . . 2048
Encryption . . . 2048
Decryption . . . 2051
Random number generation . . . 2053
Generation of string type pseudo-random number . . . 2053
Generation of byte array type pseudo-randon number . . . 2053
9.9 Implementation Example of Typical Security Requirements . . . 2055
9.9.1 Introduction . . . 2055
Topics described in this chapter . . . 2055
Target Readers . . . 2055
9.9.2 Description of application . . . 2055
Security requirements . . . 2055
Functions . . . 2060
Specifications for authentication/authorization . . . 2060
Authentication . . . 2060
Authorization . . . 2060
Authentication at the time of reissuing password . . . 2060
Design information . . . 2061
Page transition . . . 2061
URL List . . . 2064
ER diagram . . . 2066
9.9.3 Implementation method and code description . . . 2069
Force/Prompt password change . . . 2069
List of requirements to be implemented . . . 2069
Working image . . . 2069
Implementation method . . . 2069
Code description . . . 2071
Check password strength . . . 2084
List of requirements to be implemented . . . 2084
Working image . . . 2084
Implementation method . . . 2084
Code description . . . 2086
Account lock . . . 2101
List of requirements to be implemented . . . 2101
Working image . . . 2101
Implementation method . . . 2103
Code description . . . 2105
Display the date and time of last login . . . 2120

List of requirements to be implemented . . . 2120
Working image . . . 2120
Implementation method . . . 2120
Code description . . . 2120
Creating authentication information for password reissue . . . 2129
List of requirements to be implemented . . . 2129
Working image . . . 2129
Implementation method . . . 2130
Code description . . . 2130
Distribution of authentication information for password reissue . . . 2140
List of requirements to be implemented . . . 2140
Working image . . . 2140
Implementation method . . . 2140
Code description . . . 2141
Validation at the time of executing password reissue . . . 2145
List of requirements to be implemented . . . 2145
Working image . . . 2146
Implementation method . . . 2146
Code description . . . 2147
Setting failure limit for the password reissue . . . 2157
List of requirements to be implemented . . . 2157
Working image . . . 2157
Implementation method . . . 2158
Code description . . . 2158
Input check validation for security . . . 2165
List of requirements to be implemented . . . 2165
Working image . . . 2165
Implementation method . . . 2165
Code description . . . 2167
Audit log output . . . 2187
List of requirements to be implemented . . . 2187
Working image . . . 2187
How to implement . . . 2188
Code description . . . 2189
9.9.4 Conclusion . . . 2194
9.9.5 Appendix . . . 2195
Passay . . . 2195
Password validation . . . 2195
Password generation . . . 2201
9.10 OAuth . . . 2205
9.10.1 Overview . . . 2205

About OAuth 2.0 . . . 2205
Architecture of OAuth 2.0 . . . 2207
Roles . . . 2208
Scope . . . 2209
Protocol flow . . . 2210
Authorization grant . . . 2212
Life cycle of access token . . . 2219
Architecture of Spring Security OAuth . . . 2222
Authorization server . . . 2224
Resource server . . . 2228
Client . . . 2231
9.10.2 How to use . . . 2233
Set-up of Spring Security OAuth . . . 2233
Application settings . . . 2233
Implementation of authorization server . . . 2234
Creation of setting file (Authorization server) . . . 2234
Defining authorization server . . . 2235
Client authentication . . . 2237
Resource owner authentication . . . 2245
Authorization for each scope . . . 2246
Customising scope authorization screen . . . 2249
Error handling at the time of authorization request . . . 2251
How to share access token with resource server . . . 2253
Canceling a token (authorization server) . . . 2258
Transaction control . . . 2261
Implementation of resource server . . . 2263
Creating a setup file (resource server) . . . 2263
Setting of accessible scope for the resource . . . 2266
Configuration of access token (Resource Server) . . . 2269
Fetching user information . . . 2269
Implementation of client . . . 2271
Accessing a resource using OAuth2RestTemplate . . . 2272
Accessing a resource using JavaScript . . . 2283
Cancellation of token (Client server) . . . 2292
9.10.3 How to extend . . . 2296
Linking Authorization Server and Resource Server through endpoints . . . 2296
Linkage through HTTP access . . . 2296
Extension of DefaultAccessTokenConverter . . . 2299

10 Tutorials 2307
10.1 Tutorial (Todo Application) . . . 2307
10.1.1 Introduction . . . 2307

Points to study in this tutorial . . . 2307
Target readers . . . 2307
Verification environment . . . 2307
10.1.2 Description of application to be created . . . 2308
Overview of the application . . . 2308
Business requirements of application . . . 2308
Screen transition of application . . . 2308
Show all TODO . . . 2309
Create TODO . . . 2309
Finish TODO . . . 2310
Delete TODO . . . 2310
Error message list . . . 2310
10.1.3 Environment creation . . . 2310
Project creation . . . 2311
Creating O/R Mapper independent blank project . . . 2311
Creating blank project for MyBatis3 . . . 2312
Creating blank project for JPA . . . 2312
Project import . . . 2312
Project configuration . . . 2315
Confirmation of configuration file . . . 2318
Operation verification of the project . . . 2318
10.1.4 Creation of Todo application . . . 2324
Creation of Domain layer . . . 2325
Creation of Domain Object . . . 2325
Repository creation . . . 2328
Creation of RepositoryImpl (Infrastructure layer) . . . 2330
Service creation . . . 2332
Creation of JUnit for Service . . . 2338
Creation of application layer . . . 2338
Creation of Controller . . . 2338
Show all TODO implementation . . . 2339
Create TODO . . . 2346
Finish TODO . . . 2353
Delete TODO . . . 2361
Use of CSS file . . . 2367
10.1.5 Creating infrastructure layer with a Database access . . . 2371
Creating a blank project dependent on O/R Mapper . . . 2371
Database set-up . . . 2371
Modification in todo-infra.properties . . . 2371
Creating infrastructure layer with MyBatis3 . . . 2372
Create TodoRepository . . . 2372

Create TodoRepositoryImpl . . . 2373
Create Mapper file . . . 2373
Creating infrastructure layer with Spring Data JPA . . . 2378
Modification in Entity . . . 2378
Creat TodoRepository . . . 2380
Create TodoRepositoryImpl . . . 2381
10.1.6 In the end... . . . 2383
10.1.7 Appendix . . . 2383
Description of the configuration file . . . 2383
web.xml . . . 2384
Common JSP . . . 2389
Bean definition file . . . 2390
logback.xml . . . 2410
10.2 Tutorial (Todo Application REST) . . . 2414
10.2.1 Introduction . . . 2414
Points to study in this tutorial . . . 2414
Target readers . . . 2414
Verification environment . . . 2414
10.2.2 Environment creation . . . 2414
Install DHC . . . 2414
Project creation . . . 2417
10.2.3 REST API creation . . . 2417
API specification . . . 2419
GET Todos . . . 2419
POST Todos . . . 2419
GET Todo . . . 2420
PUT Todo . . . 2420
DELETE Todo . . . 2421
Error Response . . . 2421
DispatcherServlet for REST API . . . 2422
Modification of web.xml . . . 2422
Creation of spring-mvc-rest.xml . . . 2426
Definition of Spring Security for REST API . . . 2430
Creation of REST API package . . . 2433
Creation of Resource class . . . 2433
Creation of Controller class . . . 2435
Implementation of GET Todos . . . 2436
Implementation of POST Todos . . . 2439
Implementation of GET Todo . . . 2443
Implementation of PUT Todo . . . 2449
Implementation of DELETE Todo . . . 2452

Implementation of exception handling . . . 2457
Change Domain layer implementation . . . 2457
Error message definition . . . 2460
Create a package that contains the error handling class . . . 2462
Creating REST API error handling class . . . 2462
Creating JavaBean for holding the REST API error information . . . 2463
Implementation of putting error information to the HTTP response BODY . . . 2465
Error handling of Input errors . . . 2466
Business exception error handling . . . 2469
Resource not found exception error handling . . . 2473
System exception error handling . . . 2476
10.2.4 In the end... . . . 2480
10.3 Session tutorial . . . 2481
10.3.1 Introduction . . . 2481
Flow of learning . . . 2481
Lessons to be learnt in the tutorial . . . 2481
Target Readers . . . 2481
Test environment . . . 2481
10.3.2 Application overview and requirements . . . 2482
Overview . . . 2482
Requirements . . . 2483
Functional requirements . . . 2483
Non-functional requirements . . . 2483
Configuration of platform . . . 2485
10.3.3 Application design . . . 2485
Screen definition . . . 2485
Extracting URL . . . 2486
Designing input/output data . . . 2490
Extracting data . . . 2490
Defining lifecycle . . . 2491
Determining the status of using a session . . . 2493
Method of implementation to use data during the session . . . 2497
Considerations while using session . . . 2498
Session synchronization . . . 2498
Session time out . . . 2499
Overall application design . . . 2499
10.3.4 Project configuration . . . 2500
Project creation . . . 2500
Project configuration . . . 2500
Operation verification . . . 2504

10.3.5 Creating simple EC site application . . . 2505
Create account information change function . . . 2505
Creating form object . . . 2507
Creating Controller . . . 2510
Creating JSP . . . 2514
Checking operation . . . 2520
Create cart item registration function . . . 2520
Defining the Bean for session scope . . . 2520
Creating the form object . . . 2522
Creating Controller . . . 2523
Creating JSP . . . 2525
Checking operation . . . 2529
Create a mechanism to retain the product search information . . . 2530
Create session scope Bean . . . 2530
Modifying Controller . . . 2532
Checking operation . . . 2535
Create cart item deletion function . . . 2535
Creating form object . . . 2535
Creating Controller . . . 2536
Creating JSP . . . 2537
Checking operation . . . 2539
Create the product order function . . . 2539
Creating Controller . . . 2540
Creating JSP . . . 2542
Checking operation . . . 2547
Settings for session synchronization and timeout . . . 2547
10.3.6 Conclusion . . . 2548
10.4 Spring Security Tutorial . . . 2549
10.4.1 Introduction . . . 2550
Topics covered in this tutorial . . . 2550
Target Audience . . . 2550
Verification environment . . . 2550
10.4.2 Overview of application to be created . . . 2551
10.4.3 Creating environment . . . 2552
Creating a project . . . 2552
10.4.4 Creating an application . . . 2552
Implementing domain layer . . . 2552
Creating a Domain Object . . . 2553
Creating AccountRepository . . . 2554
Creating AccountSharedService . . . 2555
Creating Authentication Service . . . 2557

Setting database initialization script . . . 2561
Package Explorer after creating Domain Layer . . . 2563
Implementing Application Layer . . . 2563
Spring Security settings . . . 2563
Creating login page . . . 2569
Accessing account information of login user from JSP . . . 2571
Adding logout button . . . 2572
Accessing account information of login user from Controller . . . 2574
Package explorer after creating application layer . . . 2576
10.4.5 Summary . . . 2576
10.4.6 Appendix . . . 2577
Description of configuration file . . . 2577
spring-security.xml . . . 2577
spring-mvc.xml . . . 2580

11 Appendix(Know How) 2585
11.1 Maven Repository Management using NEXUS . . . 2585
11.1.1 Why NEXUS? . . . 2585
11.1.2 Install and Start-up . . . 2586
11.1.3 settings.xml . . . 2587
11.1.4 mvn deploy how to . . . 2588
11.1.5 pom.xml . . . 2589
11.1.6 Upload 3rd party artifact (ex. ojdbc6.jar) . . . 2589
use artifact . . . 2591
11.2 Reducing Boilerplate Code (Lombok) . . . 2592
11.2.1 Lombok . . . 2592
11.2.2 Efficiency of Lombok . . . 2593
11.2.3 Lombok setup . . . 2595
Inclusion of dependent library . . . 2595
IDE Integration . . . 2596
Download Lombok . . . 2596
Lombok Installation . . . 2596
11.2.4 How to use Lombok . . . 2597
Lombok Annotations . . . 2598
Creation of JavaBean . . . 2599
How to exclude specific field from toString . . . 2600
How to exclude specific field from equals and hashCode . . . 2601
How to generate constructor for field initialization . . . 2603
Creating logger instance . . . 2604
11.3 Reference Books . . . 2607
11.4 Spring Framework Comprehension Check . . . 2608

Note: If there are any mistakes in the contents or any comments related to the contents, please raise them at Issues on Github.

1

In the Beginning

1.1 Terms of Use

In order to use this document, you are required to agree to abide by the following terms. If you do not agree with the terms, you must immediately delete or destroy this document and all its duplicate copies.

1. Copyrights and all other rights of this document shall belong to NTT DATA or third party possessing such rights.

2. This document may be reproduced, translated or adapted, in whole or in part for personal use. However, deletion of the terms given on this page and copyright notice of NTT DATA is prohibited.

3. This document may be changed, in whole or in part for personal use. Creation of secondary work using this document is allowed. However, “Reference document: TERASOLUNA Server Framework for Java (5.x) Development Guideline” or equivalent documents may be mentioned in created document and its duplicate copies.

4. Document and its duplicate copies created according to Clause 2 may be provided to third party only if these are free of cost.

5. Use of this document and its duplicate copies, and transfer of rights of this contract to a third party, in whole or in part, beyond the conditions specified in this contract, are prohibited without the written consent of NTT Data.

6. NTT DATA shall not bear any responsibility regarding correctness of contents of this document, warranty of fitness for usage purpose, assurance for accuracy and reliability of usage result, liability for defect warranty, and any damage incurred directly or indirectly.

7. NTT DATA does not guarantee the infringement of copyrights and any other rights of third party through this document. In addition to this, NTT DATA shall not bear any responsibility regarding any claim (Including the claims occurred due to dispute with third party) occurred directly or indirectly due to infringement of copyright and other rights.

This document is created with reference to Macchinetta “Reference: Macchinetta Server Framework Development Guideline”

Registered trademarks or trademarks of company name and service name, and product name of their respective companies used in this document are as follows.

- TERASOLUNA is a registered trademark of NTT DATA Corporation.
- Macchinetta is a registered trademark of NTT
- All other company names and product names are the registered trademarks or trademarks of their respective companies.

1.2 Reference

1.2.1 Macchinetta Terms of Use

In order to use this document, you are required to agree to abide by the following terms. If you do not agree with the terms, you must immediately delete or destroy this document and all its duplicate copies.

1. Copyrights and all other rights of this document shall belong to Nippon Telegraph and Telephone Corporation(hereinafter called “NTT”) or third party possessing such rights.
2. This document may be reproduced, translated or adapted, in whole or in part for personal use. However, deletion of the terms given on this page and copyright notice of NTT is prohibited.
3. This document may be changed, in whole or in part for personal use. Creation of secondary work using this document is allowed. However, “Reference document: Macchinetta Server Framework Development Guideline” or equivalent documents may be mentioned in created document and its duplicate copies.
4. Document and its duplicate copies created according to Clause 2 may be provided to third party only if these are free of cost.
5. Use of this document and its duplicate copies, and transfer of rights of this contract to a third party, in whole or in part, beyond the conditions specified in this contract, are prohibited without the written consent of NTT Data.
6. NTT shall not bear any responsibility regarding correctness of contents of this document, warranty of fitness for usage purpose, assurance for accuracy and reliability of usage result, liability for defect warranty, and any damage incurred directly or indirectly.
7. NTT does not guarantee the infringement of copyrights and any other rights of third party through this document. In addition to this, NTT DATA shall not bear any responsibility regarding any claim (Including the claims occurred due to dispute with third party) occurred directly or indirectly due to infringement of copyright and other rights.

Registered trademarks or trademarks of company name and service name, and product name of their respective companies used in this document are as follows.

- Macchinetta is a registered trademark of NTT
- TERASOLUNA is a registered trademark of NTT DATA Corporation.
- All other company names and product names are the registered trademarks or trademarks of their respective companies.

1.3 This document covers the following

This guideline provides best practices to develop highly maintainable Web applications using full stack framework focussing on Spring, Spring MVC and JPA, MyBatis.

This guideline helps to proceed with the software development (mainly coding) smoothly.

1.4 Target readers of this document

This guideline is written for the architects and programmers having software development experience and knowledge of the following.

- Basic knowledge of DI and AOP of Spring Framework
- Web development experience using Servlet/JSP
- Knowledge of SQL
- Experience of building Web Application using Maven

This guideline is not for beginners.

In order to check whether the readers have basic knowledge of Spring Framework, refer to *Spring Framework Comprehension Check*. It is recommended to study the following books if one is not able to answer 40% of the comprehension test.

- Spring 徹底入門 (翔泳社) [Japanese]
- Spring3 入門—Java フレームワーク・より良い設計とアーキテクチャ（技術評論社）[Japanese]
- Pro Spring 4th Edition (Apress)

1.5 Structure of this document

•*Architecture Overview* Overview of Spring MVC and basic concepts of TERASOLUNA Server Framework for Java (5.x) is explained.

•*Application Development* Knowledge and methods for application development using TERASOLUNA Server Framework for Java (5.x) are explained.

•../ArchitectureInDetail/index Method to implement the functions required for general application development using TERASOLUNA Server Framework for Java (5.x) or features of each function is explained.

•*Security countermeasures* Security measures are explained focusing on Spring Security.

•*Tutorials* Experience in application development using TERASOLUNA Server Framework for Java (5.x) through simple application development.

- *Appendix(Know How)* Describing the additional information when TERASOLUNA Server Framework for Java (5.x) is being used.

1.6 Reading this document

Firstly read "*Architecture Overview*".

Implement "*First application based on Spring MVC*" for beginners of Spring MVC.

Read "*Application Layering*" as the terminology and concepts used in this guideline are explained here.

Then continue with "*Tutorials*".

Get a feel of application development using TERASOLUNA Global

Framework by firstly trying it as per the proverb "Practice makes perfect".

Use this tutorial to study the details of application development in "*Application Development*".

Since the knowhow for development is explained using Spring MVC in "*Implementation of Application Layer*",

it is recommended to read it again and again several times.

One can get a better understanding by studying "*Tutorials*" once again after reading this chapter.

It is strongly recommended that all the developers who use TERASOLUNA Server Framework for Java (5.x) read it.

Refer to "../ArchitectureInDetail/index" and "*Security countermeasures*"

as per the requirement. However, read "*Input Validation*" since it is normally required for application development.

The technical leader understands all the contents and checks the type of policy to be decided in the project.

Note: If you do not have sufficient time, first go through the following.

1. *First application based on Spring MVC*
2. *Application Layering*
3. *Tutorial (Todo Application)*
4. *Application Development*
5. *Tutorial (Todo Application)*

6. *Input Validation*

1.7 Tested environments of this document

For tested environments of contents described in this guideline, refer to "Tested Environment".

1.8 Criteria based mapping of guideline

The chapter 4 of this guideline is structured functionality wise. This section shows a mapping from a point of view other than functionality. It indicates which part of guideline contains which type of content.

1.8.1 Mapping based on security measures

A point of view by OWASP(Open Web Application Security Project)

Using OWASP Top 10 for 2013 as an axis, links to explanation of functionalities related to security have been given

Sr. No.	Item Name	Corresponding Guideline
A1	Injection SQL Injection	• *Database Access (MyBatis3)* • *Database Access (JPA)* (Details about using bind variable at the time of placeholders for query parameters)
A1	Injection XXE(XML External Entity) Injection	• *Ajax*
A1	Injection OS Command Injection	• *Input Validation*
A1	Injection Email Header Injection	• *Email header injection countermeasures*
A1	Injection	• *Input check validation for security* (Shows how to validate input values)
A2	Broken Authentication and Session Management	• *Session Management* • *Authentication*
A3	Cross-Site Scripting (XSS)	• *XSS Countermeasures* • *Coordinating with browser security countermeasure function*

次のページに続く

Table. 1.1 – 前のページからの続き

Sr. No.	Item Name	Corresponding Guideline
A4	Insecure Direct Object References Directory Traversal	• *Directory traversal attack*
A5	Security Misconfiguration	• *Logging*(Mention about message contents of log) • *Method to display system exception code on screen*(Mention about message output at the time of system exception)
A6	Sensitive Data Exposure	• *Properties Management* • *Encryption* • *Password hashing*
A7	Missing Function Level Access Control	• *Authorization*
A8	Cross-Site Request Forgery (CSRF)	• *CSRF Countermeasures*
A9	Using Components with Known Vulnerabilities	No mention in particular
A10	Unvalidated Redirects and Forwards	No mention in particular

A point of view by CVE(Common Vulnerabilities and Exposures)

Explain the CVE mentioned in this guideline and show the link. About the CVE not mentioned in this guideline, refer toPivotal Product Vulnerability Reports

CVE	Outline	The mentioned point in this guideline
CVE-2014-0050 CVE-2016-3092	Apache Commons FileUpload allows remote attackers to cause a denial of service via a malicious request.	• *Overview* • *File upload using Commons FileUpload*
CVE-2014-1904	When `action`parameter of `<form:form>`tag is omitted, an attacker can use XSS attack.	• *Implementing JSP*
CVE-2015-3192	Using DTD allows DoS attack.	• *How to use* • *Application settings*
CVE-2016-5007	Differences in the strictness of the pattern matching mechanisms between Spring MVC and Spring Security cause security bypass vulnerability.	• *Specifying a Web resource for applying access policy*

1.9 Change Log

Modified on	Modified locations	Modification details
2017-03-17	-	5.3.0 RELEASE version published • For details of update, refer Issue list of 5.3.0.
	General	Correction of guideline mistakes (typing errors, simple mistakes, etc.) • For details of the correction,please refer 5.3.0's Issue list (clerical error). Description details improved • For details of the improvement、please refer 5.3.0's Issue list(improvement). Modification in Maven command's startup option for generating a blank project due to the change of Maven archetype (change to Maven Central)(guideline#2444) • *First application based on Spring MVC* • *Create Web application development project* • *Tutorial (Todo Application)* • *Spring Security Tutorial*
	Criteria based mapping of guideline	Description details added • A table listing a point of view by CVE is added in Mapping based on security measures (guideline#2439)
	Terms of Use	Description details modified • Terms of use modified(guideline#2625)
	Stack of TERASOLUNA Server Framework for Java (5.x)	Version of OSS to be used updated(guideline#2441) • Update version of Spring IO Platform to Athens-SR2 • Update version of MyBatis to 3.4.2 • Update version of MyBatis-Spring to 1.3.1 • Update mybatis-typehandlers-jsr310 to 1.0.2 OSS version to be used in accordance with version update of Spring IO Platform is updated
	Domain Layer Implementation	Description details modified • Modified signature-limiting interface and base class implementation sample(guideline#2219)
	Implementation of Application Layer	Description details added • Added "`<mvc:view-controller>` is used when a simple view controller is to be created" (guideline#2371)
		• Precautions to indicate the existence of unusable characters in Cookie name or value added (guideline#2518) Modifications related to Spring Framework 4.3 • Precautions while using `@DateTimeFormat` for JSR-310 Date and Time API class deleted (guideline#2505)

* -

  - *Coordinating with browser security countermeasure function*
  - Modifications associated with Spring Security 4.1.4 support
    - Added description on Content Security Policy (CSP)"(guideline#2400)
    - Description added for HTTP Public Key Pinning (HPKP) (guideline#2401)
  - –
    - *OAuth*
    - New addition
      * Added OAuth(guideline#2145)
  - –
    - *Tutorial (Todo Application)*
    - Correction of description
      * Code example of entity when using JPA modified (guideline#2476)
  - –
    - *Maven Repository Management using NEXUS*
    - Modification to migration into Maven Central
      * Delete the description about TERASOLUNA Server Framework for Java (5.x) repository (guideline#2496)
  - – 2016-08-31
    - -
    - 5.2.0 RELEASE version published
      * For details of update, refer Issue list of 5.2.0.
  - –
    - General
    - Correction of errors in the guideline (typos or simple description errors)
      * For details of modifications, refer Issue list of 5.2.0 (clerical error).

      Description details modified
      * For details of modification, refer Issue list of 5.2.0 (improvement).

Review of all the chapters

* For details of update, refer Optimize the order of chapters and sections #1683.

Update common library version to 5.2.0.

* For details of update, refer Check Version #2076.

Description details modified

* Added regarding pom dependency of common library (guideline#1982)

- –
  - *Stack of TERASOLUNA Server Framework for Java (5.x)*
  - Description details added
    * Embedding status of common library standards of blank project added (guideline#1700)
    * mybatis-typehandlers-jsr310, jackson-datatype-jsr310 added to OSS stack (guideline#1966)
    * spring-jms and its dependent libraries added to OSS stack (guideline#1992)

    Version of OSS used (Spring IO Platform version) updated)
    * Spring IO Platform version updated to 2.0.6.RELEASE
    * Spring Framework version updated to 4.2.7.
    * Spring Security version updated to 4.0.4.RELEASE

    OSS version used in accordance with Spring IO Platform version update is updated
- –
  - *Domain Layer Implementation*
  - Description details added
    * For MyBatis 3.3 + MyBatis-Spring 1.2, "value specified in timeout attribute of @Transactinal is not used" is added (guideline#1777)
- –
  - *Implementation of Application Layer*
  - Description details added
    * HttpSession should not be used as an argument for handler method (guideline#1313)
    * Precautions for using JSR-310 Date and Time API are described (guideline#1991)
- –

  - *Input Validation*
  - Addition of description contents
    * Add input check method to values in collection(guideline#407)

    Description details modified

    * A method to directly handle a message property file without conversion from Native to Ascii is added (guideline#994)
    * Description for cross-field validation added (guideline#1561)
    * @DateTimeFormat description added (guideline#1873)
    * Description for ValidationMessages.properties modified (guideline#1948)
    * Precautions for input check which use Method Validation added (guideline#1998)

    Description details added

    * Description for OS command injection added (guideline#1957)

- –
  - *Exception Handling*
  - Modification associated with Spring Framework 4.2.7
    * Description details for HTTP response header output modified (guideline#1965)

- –
  - *Double Submit Protection*
  - Description details added
    * Description for specifications and implementation methods of `TransactionTokenType.CHECK` which was newly added in type attribute of `@TransactionTokenCheck` annotation (guideline#2071)

    "How to manage transaction token life cycle in How To Extend programmatic" deleted.

    * When API for application offered by `TransactionTokenContext`is used, it impacts the behaviour of internal framework like inability to maintain `TransactionToken` in the appropriate state Current API is deprecated. Description for how to use function in accordance with deprecation, deleted.

- –
  - *Internationalization*
  - Description details modified

    - Position of request parameter (default parameter name) description modified (guideline#1354)

- –
  - *File Upload*
  - Description details added
    - CVE-2016-3092Precautions for (File Upload vulnerability) added (guideline#1973)
    - Description for directory traversal attack added (guideline#2010)
- –
  - *Health Check*
  - Added new
    - Health check added (guideline#1698)
- –
  - *RESTful Web Service*
  - Description details changed / added
    - Description for the configuration while using JSR-310 Date and Time API / Joda Time changed (guideline#1966)
    - Precautions while using Jackson in Java SE 7 environment described (guideline#1966)
    - Configuration while using JSR-310 Date and Time API in JSON described (guideline#1966)
- –
  - *REST Client (HTTP Client)*
  - Description details modified
    - HTTP Proxy server configuration for RestClient added (guideline#1856)
- –
  - *SOAP Web Service (Server/Client)*
  - Description details added
    - Added an option “Do not connect to SOAP server at the time of SOAP client start (guideline#1871)
    - Description for env project of SOAP client modified (guideline#1901)

        - How to fetch status code at the time of SOAP Web service exception occurrence added (guideline#2007)

- –
  - *Database Access (MyBatis3)*
  - Description details added
    - "How to avoid tentative WARN log output" deleted (guideline#1292)
    - "How to configure for using JSR-310 Date and Time API in Mybatis3.3" described (guideline#1966)
    - Precautions while using MyBatis in Java SE 7 environment described (guideline#1966)
- –
  - *Exclusive Control*
  - Description details added
    - warning message added to ExclusionControl (guideline#1694)
- –
  - *Logging*
  - Description details added
    - "How to extend in order to output log message with ID" described (guideline#1928)
- –
  - *String Processing*
  - Description details added
    - An example to add terasoluna-gfw-string to dependency is added (guideline#1699)
    - Precautions for surrogate pair added to description of @Size annotation (guideline#1874)
    - Description for JIS characters `U+2014`(EM DASH) UCS(Unicode) characters added (guideline#1914)
- –
  - *Bean Mapping (Dozer)*
  - Description details added
    - Precautions while using JSR-310 Date and Time API described (guideline#1966)
- –

  - *JMS(Java Message Service)*
  - Added new
    * JMS added (guideline#1407)
- –
  - *Authentication*
  - Modifications for Spring Security 4.0.4
    * Code example modified to include modification of specifications of authentication-failure-url in Spring Security 4.0.4 and Note deleted (guideline#1963)
- –
  - *Authorization*
  - Description details added
    * How to handle CVE-2016-5007 Spring Security / MVC Path Matching Inconsistencyadded (guideline#1976)
- –
  - *Implementation Example of Typical Security Requirements*
  - Description details added
    * “Input value check for security” added
    * “Audit log output” added
- –
  - *Reference Books*
  - Description details added
    * Spring thorough introduction” added as a a reference material (guideline#2043)
- – 2016-02-24
  - -
  - 5.1.0 RELEASE version published
    * For details of change contents, refer 5.1.0 Issue List.
- –
  - General

  - Correction of errors in the guideline (typo mistakes and simple description errors)

    Description details modified

    * For details of modification, refer 5.1.0 Issue list (improvement).

- –
  - *In the Beginning*
  - Description details added
    * Description related to operation verification environment of the details described in the guideline added
- –
  - *Stack of TERASOLUNA Server Framework for Java (5.x)*
  - OSS version to be used (Spring IO Platform version) updated
    * Spring IO Platform version updated in 2.0.1.RELEASE
    * Spring Framework version updated in 4.2.4.RELEASE
    * Spring Security version updated in 4.0.3.RELEASE

    OSS version to be used along with Spring IO Platform version update is updated

    * OSS version to be used updated. For update details, refer version 5.1.0 migration guide.

    New project added

    * Descriptions for `terasoluna-gfw-string`, `terasoluna-gfw-codepoints`, `terasoluna-gfw-validator`, `terasoluna-gfw-web-jsp` projects added.

    New function of common library added

    `terasoluna-gfw-string`

    * Half width to full width conversion

    `terasoluna-gfw-codepoints`

    * Codepoint check
    * Bean Validation constraint annotation for code point check

    `terasoluna-gfw-validator`

    * Bean Validation constraint annotation for byte length check
    * Bean Validation constraint annotation for field value comparison correlation check

- –
  - *First application based on Spring MVC*
  - Description details modified
    - Modification of sample source corresponding to Spring Security 4 (guideline#1519)
    - `AuthenticationPrincipalArgumentResolver` package changed
- –
  - *Tutorial (Todo Application)*
  - Modifications corresponding to Spring Security 4
    - Modification of source corresponding to Spring Security 4 (guideline#1519)
    - `AuthenticationPrincipalArgumentResolver` package changed
    - Since the specification is true by default, `<use-expressions="true">` deleted from sample source
- –
  - *Create Web application development project*
  - Modification of description details
    - A method wherein mvn command is used in the offline environment is added (guideline#1197)
- –
  - *Implementation of Application Layer*
  - Description details modified
    - A method to create a request URL using EL function is added (guideline#632)
- –
  - *Database Access (Common)*
  - Description details added
    - Precautions for `Log4jdbcProxyDataSource` overhead added (guideline#1471)
- –
  - *Database Access (MyBatis3)*
  - Description details corresponding to MyBatis 3.3 added

    - Setup method of `defaultFetchSize` added (guideline#965)
    - "Changed the default at the time of delayed reading to `JAVASSIST`" added (guideline#1384)
    - Sample code which assigns Genrics to `ResultHandler` modified (guideline#1384)
    - Source example which use newly added `@Flush` annotation, and precautions added (guideline#915)

- –
  - *Database Access (JPA)*
  - Bug correction for the guideline
    - Utility which use Like condition modified appropriately (guideline#1464)
    - Incorrect implementation of true value in JPQL corrected (guideline#1525)
    - Incorrect implementation of pagination corrected (guideline#1463)
    - Incorrect implementation of sample code corrected which implements `DateTimeProvider` (guideline#1327)
    - Incorrect implementation in Factory class for generating an instance of implementation class for common Repository interface corrected (guideline#1327)

    Description details modified
    - Default value of `hibernate.hbm2ddl.auto` corrected (guideline#1282)

- –
  - *Input Validation*
  - Description details modified
    - Description for MethodValidation added (guideline#708)

- –
  - *Logging*
  - Description details modified
    - Description where `ServiceLoader` mechanism is used in Logback setting, is added (guideline#1275)
    - Sample source corresponding to Spring Security 4 modified (guideline#1519)
    - Since the specification is true by default, `<use-expressions="true">` deleted from the sample source

- –
  - *Session Management*
  - Description details modified
    - Description of session scope reference which use SpEL expression is added (guideline#1306)
- –
  - *Internationalization*
  - Description details modified
    - Description for appropriately reflecting locale in JSP is added (guideline#1439)
    - Description of `defaultLocale` of `SessionLocalResolver` corrected (guideline#686)
- –
  - *File Upload*
  - Description details added
    - Description for directory traversal attack added (guideline#2010)
- –
  - *Codelist*
  - Description details added
    - Description which recommends a pattern wherein `JdbcTemplate` is specified in JdbcCodeList, is added (guideline#501)
- –
  - *Health Check*
  - New
    - Health check added (guideline#1698)
- –
  - *RESTful Web Service*
  - Description details modified
    - Creation of ObjectMapper which use `Jackson2ObjectMapperFactoryBean` added (guideline#1022)

        * Modified to a format where MyBatis3 is used as a prerequisite in the implementation of domain layer of REST API application (guideline#1323)

- –
  - *REST Client (HTTP Client)*
  - Added new
    * REST client (HTTP client) added (guideline#1307)
- –
  - *SOAP Web Service (Server/Client)*
  - Added new
    * SOAP Web Service (Server / Client) added (guideline#1340)
- –
  - *File Upload*
  - Description details modified
    * Basic flow of uploading process and its description modified to description which use `MultipartFilter` of Spring (guideline#193)
    * "A method which sends CSRF token by query parameter" deleted due to issues like security issues, variation in the operation according to AP server etc. Precaution - "when allowable size for file upload exceeds, CSRF token check is not carried out appropriately in some AP servers" added (guideline#1602)
- –
  - *File Download*
  - Description details corresponding to Spring Framework4.2 added
    * `AbstractXlsxView` which manages xlsx format, is added (guideline#996)

    Description details modified
    * Source example which use `com.lowagie:itext:4.2.1` modified to a format which uses `com.lowagie:itext:2.1.7` for the specification change of the iText
- –
  - *Sending E-mail (SMTP)*
  - Added new
    * E-mail sending (SMTP) added (guideline#1165)

- –
  - *Date operations (JSR-310 Date and Time API)*
  - Added new
    - Date and time operation (JSR-310 Date and Time API) added (guideline#1450)
- –
  - *Date Operations (Joda Time)*
  - Description details added and modified
    - The object of sample code which handles the date that does not use Timezone modified to `LocalDate` (guideline#1283)
    - A method to handle Japanese calendar in Java8 and earlier versions is added (guideline#1450)
- –
  - *Logging*
  - Description details added
    - Extension method to output log message with ID is described (guideline#1928)
- –
  - *String Processing*
  - Added new
    - String processing added (guideline#1451)
- –
  - *JMS(Java Message Service)*
  - Added new
    - JMS added (guideline#1407)
- –
  - *Security countermeasures*
  - Configuration review
    - `Password hashing`moved in *Authentication*
    - Session management items are separated as *Session Management* from *Authentication*
- –

  - *Spring Security Overview*
  - Modify corresponding to Spring Security 4
    - Restructuring overall description
    - `spring-security-test` introduction
    - Since the specification is true by default, `<use-expressions="true">` deleted from sample source
    - Description related to `RedirectAuthenticationHandler`deprecation deleted
- –
  - *Spring Security Tutorial*
  - Modified corresponding to Spring Security 4
    - Modified tutorial source to a format corresponding to Spring Security 4 (guideline#1519)
- –
  - *Authentication*
  - Modified corresponding to Spring Security 4 (guideline#1519)
    - Restructuring of overall description
    - Deleted `auto-config="true"`
    - Authentication event listener modified to `@org.springframework.context.event.EventListene`
    - Modified `AuthenticationPrincipal` package
    - Since prefix is assigned by default, `ROLE_` prefix deleted from sample source
- –
  - *Authorization*
  - Modified corresponding to Spring Security 4 (guideline#1519)
    - Restructuring of overall description
    - Since the prefix is assigned by default, `ROLE_` prefix deleted from sample source
    - Since the specification is true by default, `<use-expressions="true">` deleted from sample source
    - Definition example of `@PreAuthorize` added
- –

  - *CSRF Countermeasures*
  - Modified corresponding to Spring Security 4
    - Restructuring of overall description
    - CSRF invalidation settings modified `<sec:csrf disabled="true"/>`
    - Description details modified
    - Items related to multi-part request moved to *File Upload* (guideline#1602)
- –
  - *Encryption*
  - Added new
    - Encryption guidelines added (guideline#1106)
- –
  - *Implementation Example of Typical Security Requirements*
  - Added new
- –
  - *Implementation Example of Typical Security Requirements*
  - Description details added
    - “Input check for security” added
    - “Audit log output” added
    - Typical implementation example of security requirements added (guideline#1604)
- –
  - *Session tutorial*
  - Added new
    - Session tutorial added (guideline#1599)
- –
  - *Tutorial (Todo Application REST)*
  - Modified corresponding to Spring Security 4
    - Modified source corresponding to Spring Security 4 (guideline#1519)
    - CSRF invalidation settings modified `<sec:csrf disabled="true"/>`

    - Since the specification is true by default, `<use-expressions="true">` deleted from sample source

- – 2015-08-05
  - –
  - Released “5.0.1 RELEASE” version
    - For update details, refer to Issue list of 5.0.1
- –
  - Overall modifications
  - Fixed guideline errors (corrected typos, mistakes in description, etc.)
    - For modification details, refer to Issue list of 5.0.1 (clerical error)

  Improved the description
    - For improvement details, Issue list of 5.0.1 (improvement)

  Fixed the description about application server
    - Removed the description for the Resin
    - Updated the link of reference page
- –
  - *In the Beginning*
  - Added the description
    - Added description about tested environments for contents described in this guideline
- –
  - *Stack of TERASOLUNA Server Framework for Java (5.x)*
  - Updated the OSS version(Spring IO Platform version) to protect security vulnerability
    - Spring IO Platform version updated to 1.1.3.RELEASE
    - Spring Framework version updated to 4.1.7.RELEASE (CVE-2015-3192)
    - JSTL version updated to 1.2.5 (CVE-2015-0254)

  Updated the OSS version by the Spring IO Platform version update
    - Updated the OSS version to be used. For update details, refer to Migration guide of version 5.0.1

Improved the description (guideline#1148)

* Added the description of `terasoluna-gfw-recommended-dependencies`,`terasoluna-gfw-rec` and `terasoluna-gfw-parent`
* Modified the description for some project
* Added the illustration to indicate project dependencies

- –
  - *Create Web application development project*
  - Added the description
    * Added how to build a war file (guideline#1146)
- –
  - *Database Access (Common)*
  - Added the description
    * Added the description of `DataSource` switching functionality (guideline#1071)
- –
  - *Database Access (MyBatis3)*
  - Fixed the guideline bug
    * Modified the description about timing of batch execution (guideline#903)
- –
  - *Logging*
  - Improved the description
    * Added the description about `additivity` attribute of `<logger>` tag (guideline#977)
- –
  - *Session Management*
  - Improved the description
    * Modified the description about how to define a session scope bean (guideline#1082)
- –
  - *Double Submit Protection*
  - Added the description

        - Added the description about the transaction token check in case that response cache is disabled (guideline#1260)

- –
  - *Codelist*
  - Added the description
    - Added how to display a code name (guideline#1109)
- –
  - *Ajax*

    RESTful Web Service
  - Added the warning about CVE-2015-3192(XML security vulnerability)
    - Added the warning at the time of the StAX(Streaming API for XML) use (guideline#1211)
- –
  - *Pagination*

    JSP Tag Libraries and EL Functions offered by common library
  - Modified in accordance with bug fixes of common library
    - Modified the description about `f:query` specification , in accordance with bug fixes of common library (terasoluna-gfw#297) (guideline#1244)
- –
  - *Authentication*
  - Improved the description
    - Added the notes about handling with some properties of parent class of `ExceptionMappingAuthenticationFailureHandler` (guideline#812)
    - Modified the setting example for the `requiresAuthenticationRequestMatcher` property of `AbstractAuthenticationProcessingFilter` (guideline#1110)
- –
  - *Authorization*
  - Fixed the guideline bug
    - Modified the setting example for the `access` attribute of `<sec:authorize>` tag (JSP tag library) (guideline#1003)
- –

  - Elimination of environmental dependency
  - Added the description
    * Added how to apply the external classpath(alternative functionality of `VirtualWebappLoader` of Tomcat7) at the time of Tomcat8 use (guideline#1081)

- – 2015-06-12
  - Overall modifications
  - Released English version of "5.0.0 RELEASE"
- – 2015-03-06
  - *RESTful Web Service*
  - Guideline bug modification
    * Modification of sample code for exception handling (the code that contains the issue of generating `NullPointerException`). For improvement details, refer to guideline#918.
- –
  - *Tutorial (Todo Application REST)*
  - Guideline bug modification
    * Fixed a problem that generates ‘‘ NullPointerException‘‘ in the processing of exception handling. For improvement details, refer to guideline#918.
- – 2015-02-23
  - -
  - Released "5.0.0 RELEASE" version
    * For update details, refer to Issue list of 5.0.0 and Backport issue list of 1.0.2.
- –
  - Overall modifications
  - Fixed guideline errors (corrected typos, mistakes in description, etc.)
    * For modification details, refer to Backport issue list of 1.0.2 (clerical error).

    Improved the description
    * For improvement details, refer to Issue list of 5.0.0 (improvement) and Backport issue list of 1.0.2 (improvement).

    Added new

    - *Create Web application development project*
    - *Database Access (MyBatis3)*
    - *JSP Tag Libraries and EL Functions offered by common library*
    - *Reducing Boilerplate Code (Lombok)*

  Updated in accordance with version 5.0.0

    - Deleted MyBatis2

- –
  - *Stack of TERASOLUNA Server Framework for Java (5.x)*
  - Spring IO Platform compatible
    - Added a point that except for some libraries, the management of recommended libraries is changed to a structure delegating it to Spring IO Platform.

    Updated the OSS version

    - Updated the OSS version to be used. For update details, refer to Migration guide of version 5.0.0.

- –
  - *First application based on Spring MVC*
  - Updated in accordance with version 5.0.0
    - Used Spring Framework 4.1
    - Reviewed structure of document.

- –
  - *Application Layering*
  - Fixed bugs in English translation.
    - Fixed translation bugs related to domain layer and other layers. For modification details, refer to guideline#364 issue.

- –
  - *Tutorial (Todo Application)*
  - Updated in accordance with version 5.0.0
    - Use of Spring Framework 4.1.
    - MyBatis3 support as infrastructure layer.

    * Revised document structure.

- –
  - *Create Web application development project*
  - Added new
    * Added a method to create a project having multi project structure
- –
  - *Domain Layer Implementation*
  - Modified in accordance with Spring Framework 4.1
    * Added description about handling `@Transactional` of JTA 1.2. For modification details, refer to guideline#562 issue.
    * Modified description about handling `@Transactional(readOnly = true)` when using JPA (Hibernate implementation). With SPR-8959 (Spring Framework 4.1 and later versions) support, it has been improved so that instruction can be given so as to handle as "Read-only transactions" for JDBC driver.

    Added description
    * Added notes regarding the cases where "Read-only transactions" are not enabled. For added contents, refer to guideline#861 issue.
- –
  - *Implementation of Infrastructure Layer*
  - Modified in accordance with MyBatis3
    * Added a method to use MyBatis3 mechanism as implementation of RepositoryImpl.
- –
  - *Implementation of Application Layer*
  - Modified in accordance with Spring Framework 4.1
    * Added description about the attribute (attribute to filter the Controllers to be used) added in `@ControllerAdvice`. For modification details, refer to guideline#549 issue.
    * Added description about `<mvc:view-resolvers>`. For modification details, refer to guideline#609 issue.
- –
  - *Database Access (Common)*

  - Modified in accordance with bug fixes of common library
    * Added description about handling double byte wild card characters (%, _), in accordance with bug fixes of common library (terasoluna-gfw#78). For modification details, refer to guideline#712 issue.

    Modified in accordance with Spring Framework 4.1

    * Removed the description about the problem where pessimistic locking error of JPA (Hibernate implementation) is not converted into `PessimisticLockingFailureException` of Spring Framework. This problem is resolved in SPR-10815 (Spring Framework 4.0 and later versions).

    Modified in accordance with Apache Commons DBCP 2.0

    * Changed the sample code and its description to use component for Apache Commons DBCP 2.0.

- –
  - *Database Access (MyBatis3)*
  - Added new
    * Added method to implement an infrastructure layer using MyBatis3 as O/R Mapper.
- –
  - *Exclusive Control*
  - Fixed guideline bugs
    * Modified the sample code of optimistic locking of long transactions (processing when records cannot be fetched). For modification details, refer to guideline#450 issue.

    Modified in accordance with Spring Framework 4.1

    * Removed the description about the problem where pessimistic locking error of JPA (Hibernate implementation) is not converted into `PessimisticLockingFailureException` of Spring Framework. This problem is resolved in SPR-10815 (Spring Framework 4.0 and later versions).

    Modified in accordance with MyBatis3

    * Added methods to implement exclusive control when using MyBatis3.
- –
  - *Input Validation*
  - Fixed guideline bugs

* Modified the description of `@GroupSequence`. For modification details, refer to guideline#296 issue.

Modified in accordance with bug fixes of common library

* Added notes about `ValidationMessages.properties`, in accordance with bug fixes of common library (terasoluna-gfw#256). For modification details, refer to guideline#766 issue.

Added description

* Added a method to link with the mechanism of Group Validation of Bean Validation at the time of correlated item check using Spring Validator. For added contents, refer to guideline#320 issue.

Modified in accordance with Bean Validation 1.1 (Hibernate Validator 5.1)

* Added description about `inclusive` attribute of `@DecimalMin` and `@DecimalMax`.
* Added description about Expression Language.
* Described about deprecated API from Bean Validation 1.1.
* Added description about a bug related to `ValidationMessages.properties` of Hibernate Validator 5.1.x (HV-881) and methods to prevent the same.

- –
  - *Exception Handling*
  - Added description
    * Added a description that simple error page is likely to be displayed in Internet Explorer when an error having size lesser than 513 bytes is sent as response. For added contents, refer to guideline#189 issue.

    Modified in accordance with Spring Framework 4.1

    * Removed the description about the problem where pessimistic locking error of JPA (Hibernate implementation) is not converted into `PessimisticLockingFailureException` of Spring Framework. This problem is resolved in SPR-10815 (Spring Framework 4.0 and later versions).

- –
  - *Session Management*
  - Modified in accordance with Spring Security 3.2
    * Removed the description about a problem where CSRF token error occurs (SEC-2422) instead of session time out at the time of POST request. A mechanism to detect session time

out is included in formal version of Spring Security 3.2, hence the problem is resolved.

- –
  - *Message Management*
  - Reflected changes of common library
    - Added description about newly added message type (warning) and deprecated messages types (warn), in accordance with the improvement of common library (terasoluna-gfw#24). For modification details, refer to guideline#74 issue.
- –
  - *Pagination*
  - Reflected changes of common library
    - Changed description of page link in active state, in accordance with the improvement of common library (terasoluna-gfw#13). For modification details, refer to guideline#699 issue.
    - Changed description of page link in disabled state, in accordance with the improvement of common library (terasoluna-gfw#14). For modification details, refer to guideline#700 issue.

    Modified in accordance with Spring Data Common 1.9
    - Added notes for the classes where API specifications (`Page` interface, etc.) are changed due to version upgrade.
- –
  - *Codelist*
  - Modified in accordance with bug fixes of common library
    - Added notes about version upgrade and changing message key of `ExistInCodeList` in accordance with bug fixes of common library (terasoluna-gfw#16). For modification details, refer to guideline#638 issue.
    - Added notes about message definition of `@ExistInCodeList` in accordance with bug fixes of common library (terasoluna-gfw#256). For modification details, refer to guideline#766 issue.

    Reflected changes of common library
    - Added a method to use `EnumCodeList` class in accordance with addition of common library functions (terasoluna-gfw#25).
- –
  - *Ajax*

  - Modified in accordance with Spring Security 3.2
    * Changed the sample code for CSRF measures (method to create `<meta>` tag for CSRF measures).

    Modified in accordance with Jackson 2.4
    * Changed the sample code and description to use components for Jackson 2.4.

- –
  - *RESTful Web Service*
  - Improvement in description
    * Improve the method to build an URL to be set in location header and hypermedia link. For improvement details, refer to guideline#374 issue.

    Modified in accordance with Spring Framework 4.1
    * Added a description about `@RestController`. For modification details, refer to guideline#560 issue.
    * Changed the sample code to create `ResponseEntity` using builder style API.

    Modified in accordance with Jackson 2.4
    * Changed the sample code and description to use components for Jackson 2.4.

    Modified in accordance with Spring Data Common 1.9
    * Added notes for the classes where API specifications (`Page` interface, etc.) are changed due to version upgrade.

- –
  - *File Upload*
  - Fixed guideline bugs
    * Modified version of Apache Commons FileUpload with resolved CVE-2014-0050 (File Upload vulnerabilities). For modification details, refer to guideline#846 issue.

    Added description
    * File upload function of Servlet 3 has a problem of garbled characters on a part of application server. Therefore, added a method to use Apache Commons FileUpload as a measure to prevent this event. For added contents, refer to guideline#778 issue.

- –
  - *System Date*

  - Reflected changes of common library
    - Changed document structure, package name and class name in accordance with the improvement of common library (terasoluna-gfw#224). For modification details, refer to guideline#701 issue.

- –
  - *Screen Layout using Tiles*
  - Modified in accordance with Tiles 3.0
    - Changed the example of settings and description to use component for Tiles 3.0.

    Modified in accordance with Spring Framework 4.1
    - Added description about `<mvc:view-resolvers>`, `<mvc:tiles>`, `<mvc:definitions>`. For modification details, refer to guideline#609 issue.

- –
  - *Date Operations (Joda Time)*
  - Added description
    - Added method to use `LocalDateTime`. For added contents, refer to guideline#584 issue.

    Modified in accordance with Joda Time 2.5
    - Since `DateMidnight` class is deprecated in accordance with version upgrade, changed the method to fetch start time of specific date (0:00:00.000).

- –
  - *Spring Security Overview*
  - Modified in accordance with Spring Security 3.2
    - Added "Settings to create secure HTTP header" in appendix.

- –
  - *Spring Security Tutorial*
  - Updated in accordance with version 5.0.0
    - Made changes so as to use MyBatis3 as infrastructure layer.
    - Applied Spring Framework 4.1
    - Applied Spring Security 3.2
    - Revised document structure.

- –
  - *Authentication*
  - Fixed guideline bugs
    - Modified the erroneous and inadequate description of `<form-login>`, `<logout>`, `<session-management>` tag. For modification details, refer to guideline#754 issue.
    - Modified the sample code that indicates extension method of AuthenticationFilter (added settings to validate CSRF measures and session fixation attack measures). For details, refer to guideline#765 issue.

    Modified in accordance with Spring Security 3.2
    - Added notes about logout method when CSRF measures are validated.
    - Added description of `@AuthenticationPrincipal`, as a method to access `UserDetails` (authentication user information class) from Controller.
    - Added description of `changeSessionId`, as parameters of `session-fixation-protection` attribute of `<sec:session-management>`.
    - Added methods to detect session time-out and notes for same.
    - Changed setting method to validate concurrent session control of identical users (made changes so as to use `<sec:concurrency-control>`).
    - Added a point that a class of concurrent session control of identical users is deprecated and other class is provided.
- –
  - *CSRF Countermeasures*
  - Modified in accordance with Spring Security 3.2
    - Removed description about the component for CSRF measures of Spring Security 3.2.0 (provisional version before formal release) included in common library of version 1.0.x.
    - Changed setting method to validate CSRF measures by a proper method of Spring Security 3.2 (method using `<sec:csrf>`).
    - Added description about JSP tag library (`<sec:csrfInput>` and `<sec:csrfMetaTags>`) for CSRF measures.
    - Added methods to detect session time-out and precautions when CSRF measures are validated.

    Modified in accordance with Spring Framework 4.1

    - Changed description about the condition where CSRF token is output as hidden, when `<form:form>` is used.

- –
  - *Tutorial (Todo Application REST)*
  - Improved the description
    - Changed to the contents that do not depend on specific infrastructure layer (O/R Mapper), by adding REST API in the project created in *Tutorial (Todo Application)*. For modification details, refer to guideline#325 issue.

    Updated in accordance with version 5.0.0
    - Applied Spring Framework 4.1.
    - Applied Spring Security 3.2.
    - Applied Jackson 2.4.
- –
  - Create a new project from a blank project
  - Improved the description
    - Supported method to create a project having multi project structure.
    - Updated the method to create a project having single project structure.
- –
  - *JSP Tag Libraries and EL Functions offered by common library*
  - Added new
    - Added description about EL functions and JSP tag libraries provided by common libraries.
- –
  - *Reducing Boilerplate Code (Lombok)*
  - Added new
    - Added description about how to remove a boilerplate code where Lombok is used.
- –
  - English version
  - Added English version of the following.
    - *Create Web application development project*

        - *Database Access (Common)*
        - *Database Access (JPA)*
        - *Database Access (MyBatis3)*
        - *Exclusive Control*
        - *Logging*
        - *Properties Management*
        - *Pagination*
        - *Double Submit Protection*
        - *Internationalization*
        - *Codelist*
        - *Ajax*
        - *RESTful Web Service*
        - *File Upload*
        - *File Download*
        - *Screen Layout using Tiles*
        - *System Date*
        - *Bean Mapping (Dozer)*
        - *Spring Security Overview*
        - *Authentication*
        - *Authorization*
        - *CSRF Countermeasures*
        - Create a new project from a blank project
        - *Maven Repository Management using NEXUS*
        - Elimination of environmental dependency
        - Project Structure Standard
        - *Reducing Boilerplate Code (Lombok)*
        - *Spring Framework Comprehension Check*
- – 2014-08-27

  - -
  - Released "1.0.1 RELEASE" version

    Refer to Issue list of 1.0.1 for details.

- –
  - Overall modifications
  - Fixed guideline bugs (corrected typos, mistakes in description etc.)

    Refer to Issue list of 1.0.1 (bug & clerical error) for details.

- –
  - Japanese version
  - Added Japanese version of the following.
    * *Criteria based mapping of guideline*
    * *RESTful Web Service*
    * *Tutorial (Todo Application REST)*

- –
  - English version
  - Added English version of the following.
    * *In the Beginning*
    * *Architecture Overview*
    * *Tutorial (Todo Application)*
    * *Application Development*
    * *Input Validation*
    * *Exception Handling*
    * *Message Management*
    * *Date Operations (Joda Time)*
    * *XSS Countermeasures*
    * *Reference Books*

- –
  - *Stack of TERASOLUNA Server Framework for Java (5.x)*

  - Updated the OSS version in accordance with bug fixes.
    - GroupId (`org.springframework`) updated to 3.2.10.RELEASE from 3.2.4.RELEASE
    - GroupId (`org.springframework.data`)/ArtifactId(`spring-data-commons`) updated to 1.6.4.RELEASE from 1.6.1.RELEASE
    - GroupId (`org.springframework.data`)/ArtifactId(`spring-data-jpa`) updated to 1.4.3.RELEASE from 1.4.1.RELEASE
    - GroupId (`org.aspectj`) updated to 1.7.4 from 1.7.3
    - Deleted GroupId (`javax.transaction`)/ArtifactId(`jta`)
- –
  - *Implementation of Application Layer*
  - Added a warning about CVE-2014-1904 (XSS Vulnerability of `action` attribute in `<form:form>` tag)
- –
  - Japanese version

    Message Management
  - Added description about bug fix
    - Fixed bugs of `<t:messagesPanel>` tag of common library (terasoluna-gfw#10)
- –
  - Japanese version

    Pagination
  - Updated description about bug fix
    - Fixed bugs of `<t:pagination>` tag of common library (terasoluna-gfw#12)
    - Fixed bugs of Spring Data Commons (terasoluna-gfw#22)
- –
  - Japanese version

    Ajax
  - Updated description of countermeasures against XXE Injection
- –

  - Japanese version

    File Upload

  - Added a warning about CVE-2014-0050 (File Upload Vulnerability)

    Fixed guideline bugs.

    * Added how to handle `MultipartException` using error-page functionality of servlet container, because your application can't handle `MultipartException` using `SystemExceptionResolver` when used `MultipartFilter`. Refer to Issue of guideline#59 for details.

- –
  - Japanese version
  - Change how to create following projects to be carried out from `mvn archetype:generate`
    * *First application based on Spring MVC*
    * *Tutorial (Todo Application)*
    * *Tutorial (Todo Application)*
- –
  - Japanese version
  - Minor modifications in how to create following Maven archetype
    * *Spring Security Tutorial*
    * Create a new project from a blank project
- – 2013-12-17
  - Japanese version
  - Released "1.0.0 Public Review" version

2

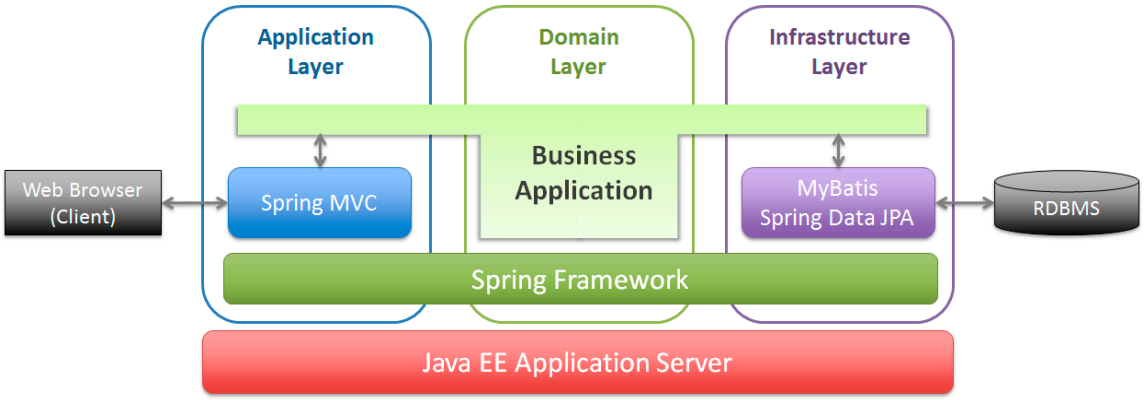
Architecture Overview

The architecture adopted in this guideline is explained here.

2.1 Stack of TERASOLUNA Server Framework for Java (5.x)

2.1.1 Summary of Software Framework of TERASOLUNA Server Framework for Java (5.x)

Software Framework being used in TERASOLUNA Server Framework for Java (5.x) is not a proprietary Framework but a combination of various OSS technologies around Spring Framework.

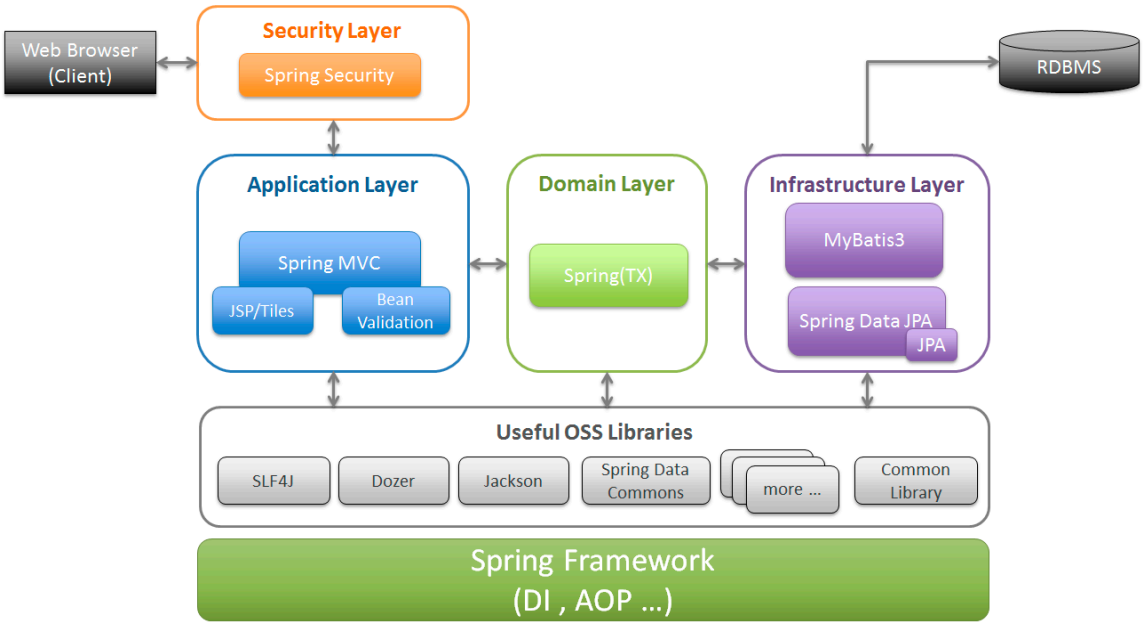


2.1.2 Main Structural Elements of Software Framework

Libraries which constitute TERASOLUNA Server Framework for Java (5.x) are as follows:

DI Container

Spring Framework is used as DI Container.



- Spring Framework 4.3

MVC Framework

Spring MVC is used as Web MVC Framework.

- Spring MVC 4.3

O/R Mapper

This guideline assumes the use of **any one of the below**.

- MyBatis 3.4
  - MyBatis-Spring is used as library for coordinating with Spring Framework.
- JPA2.1
  - Hibernate 5.0 is used as provider.

Note: To be precise MyBatis is a "SQL Mapper", but it is classified as "O/R Mapper" in this guidelines.

Warning: Every project must not adopt JPA. For situations in which table design has been done and "Most of the tables are not normalized", "The number of columns in the table is too large" etc, use of JPA is difficult.

Furthermore, this guideline does not explain the basic usage of JPA. Hence, it is pre-requisite to have JPA experience people in the team.

View

JSP is used as View.

Use the following to standardize the view layout.

- Apache Tiles 3.0

Security

Spring Security is used as the framework for Authentication and Authorization.

- Spring Security 4.1

Tip: In addition to providing a mechanism of authentication and authorization in Spring Security 3.2, the mechanism has been enhanced to protect a Web application from malicious attackers.

For mechanism to protect Web applications from malicious attackers, Refer,

- *CSRF Countermeasures*
- *Coordinating with browser security countermeasure function*

Validation

- For Single item input check, BeanValidation 1.1 is used.
  - For implementation, Hibernate Validator 5.2 is used.
- For correlated items check, BeanValidation or Spring Validation is used.
  - Refer to *Input Validation* for determining the proper use.

Logging

- The SLF4J API is used for Logger.
  - The Logback API is used for implementation of Logger.

Common Library

- https://github.com/terasolunaorg/terasoluna-gfw
- Refer to *Building blocks of Common Library* for details.

2.1.3 OSS Versions

List of OSS being used in version 5.3.0.RELEASE.

Tip: From version 5.0.0.RELEASE onwards, adopted the mechanism of importing `<dependencyManagement>` of Spring IO platform.

By importing the `<dependencyManagement>` of Spring IO platform,

- Spring Framework offering library
- Spring Framework dependent OSS library
- Spring Framework compatible OSS library

dependencies resolved and OSS version to be used in the TERASOLUNA Server Framework for Java (5.x) is following the rule of Spring IO platform definition.

Furthermore, Spring IO platform version is Athens-SR2 specified in version 5.3.0.RELEASE.

Type	GroupId	ArtifactId	Version	Spring IO platform	Remarks
Spring	org.springframework	spring-aop	4.3.5.RELEASE	*	
Spring	org.springframework	spring-aspects	4.3.5.RELEASE	*	
Spring	org.springframework	spring-beans	4.3.5.RELEASE	*	
Spring	org.springframework	spring-context	4.3.5.RELEASE	*	
Spring	org.springframework	spring-context-support	4.3.5.RELEASE	*	
Spring	org.springframework	spring-core	4.3.5.RELEASE	*	
Spring	org.springframework	spring-expression	4.3.5.RELEASE	*	
Spring	org.springframework	spring-jdbc	4.3.5.RELEASE	*	
Spring	org.springframework	spring-orm	4.3.5.RELEASE	*	
Spring	org.springframework	spring-tx	4.3.5.RELEASE	*	
Spring	org.springframework	spring-web	4.3.5.RELEASE	*	
Spring	org.springframework	spring-webmvc	4.3.5.RELEASE	*	

次のページに続く

Table. 2.1 – 前のページからの続き

Type	GroupId	ArtifactId	Version	Spring IO platform	Remarks
Spring	org.springframework	spring-jms	4.3.5.RELEASE	*	
Spring	org.springframework	spring-messaging	4.3.5.RELEASE	*	
Spring	org.springframework.data	spring-data-commons	1.12.6.RELEASE	*	
Spring	org.springframework.security	spring-security-acl	4.1.4.RELEASE	*	
Spring	org.springframework.security	spring-security-config	4.1.4.RELEASE	*	
Spring	org.springframework.security	spring-security-core	4.1.4.RELEASE	*	
Spring	org.springframework.security	spring-security-taglibs	4.1.4.RELEASE	*	
Spring	org.springframework.security	spring-security-web	4.1.4.RELEASE	*	
Spring	org.springframework.security.oauth	spring-security-oauth2	2.0.12.RELEASE	*	
MyBatis3	org.mybatis	mybatis	3.4.2		*1
MyBatis3	org.mybatis	mybatis-spring	1.3.1		*1
MyBatis3	org.mybatis	mybatis-typehandlers-jsr310	1.0.2		*1*6
JPA(Hibernate)	antlr	antlr	2.7.7	*	*2
JPA(Hibernate)	dom4j	dom4j	1.6.1	*	*2
JPA(Hibernate)	org.hibernate	hibernate-core	5.0.11.Final	*	*2
JPA(Hibernate)	org.hibernate	hibernate-entitymanager	5.0.11.Final	*	*2
JPA(Hibernate)	org.hibernate.common	hibernate-commons-annotations	5.0.1.Final	*	*2 *4
JPA(Hibernate)	org.hibernate.javax.persistence	hibernate-jpa-2.1-api	1.0.0.Final	*	*2 *4
JPA(Hibernate)	org.javassist	javassist	3.20.0-GA	*	*2
JPA(Hibernate)	org.jboss	jandex	2.0.0.Final	*	*2 *4
JPA(Hibernate)	org.springframework.data	spring-data-jpa	1.10.6.RELEASE	*	*2
JPA(Hibernate)	org.apache.geronimo.specs	geronimo-jta_1.1_spec	1.1.1	*	*2
DI	javax.inject	javax.inject	1	*	
AOP	aopalliance	aopalliance	1	*	
AOP	org.aspectj	aspectjrt	1.8.9	*	
AOP	org.aspectj	aspectjweaver	1.8.9	*	
Log output	ch.qos.logback	logback-classic	1.1.8	*	
Log output	ch.qos.logback	logback-core	1.1.8	*	*4
Log output	org.lazyluke	log4jdbc-remix	0.2.7		
Log output	org.slf4j	jcl-over-slf4j	1.7.22	*	
Log output	org.slf4j	slf4j-api	1.7.22	*	
JSON	com.fasterxml.jackson.core	jackson-annotations	2.8.5	*	

次のページに続く

Table. 2.1 – 前のページからの続き

Type	GroupId	ArtifactId	Version	Spring IO platform	Remarks
JSON	com.fasterxml.jackson.core	jackson-core	2.8.5	*	
JSON	com.fasterxml.jackson.core	jackson-databind	2.8.5	*	
JSON	com.fasterxml.jackson.datatype	jackson-datatype-joda	2.8.5	*	
JSON	com.fasterxml.jackson.datatype	jackson-datatype-jsr310	2.8.5	*	*6
Input check	javax.validation	validation-api	1.1.0.Final	*	
Input check	org.hibernate	hibernate-validator	5.2.4.Final	*	
Input check	org.jboss.logging	jboss-logging	3.3.0.Final	*	*4
Input check	com.fasterxml	classmate	1.3.3	*	*4
Bean conversion	commons-beanutils	commons-beanutils	1.9.3	*	*3
Bean conversion	net.sf.dozer	dozer	5.5.1		*3
Bean conversion	net.sf.dozer	dozer-spring	5.5.1		*3
Bean conversion	org.apache.commons	commons-lang3	3.4	*	*3
Date operation	joda-time	joda-time	2.9.6	*	
Date operation	joda-time	joda-time-jsptags	1.1.1		*3
Date operation	org.jadira.usertype	usertype.core	5.0.0.GA		*2
Date operation	org.jadira.usertype	usertype.spi	5.0.0.GA		*2
Connection pool	org.apache.commons	commons-dbcp2	2.1.1	*	*3
Connection pool	org.apache.commons	commons-pool2	2.4.2	*	*3
Tiles	commons-digester	commons-digester	2.1	*	*3
Tiles	org.apache.tiles	tiles-api	3.0.7	*	*3
Tiles	org.apache.tiles	tiles-core	3.0.7	*	*3
Tiles	org.apache.tiles	tiles-jsp	3.0.7	*	*3
Tiles	org.apache.tiles	tiles-servlet	3.0.7	*	*3
Tiles	org.apache.tiles	tiles-template	3.0.7	*	*3 *4
Tiles	org.apache.tiles	tiles-autotag-core-runtime	1.2	*	*3 *4
Tiles	org.apache.tiles	tiles-request-servlet	1.0.6	*	*3 *4
Tiles	org.apache.tiles	tiles-request-api	1.0.6	*	*3
Tiles	org.apache.tiles	tiles-request-jsp	1.0.6	*	*3 *4
Utility	com.google.guava	guava	17.0	*	
Utility	commons-collections	commons-collections	3.2.2	*	*3
Utility	commons-io	commons-io	2.5	*	*3
Servlet	org.apache.taglibs	taglibs-standard-jstlel	1.2.5	*	
Servlet	org.apache.taglibs	taglibs-standard-spec	1.2.5	*	*4
Servlet	org.apache.taglibs	taglibs-standard-impl	1.2.5	*	*4

1. Dependent libraries, when MyBatis3 is used for data access.

2. Dependent libraries, when JPA is used for data access.
3. Libraries which are not dependent on Common Library, but recommended in case of application development using TERASOLUNA Server Framework for Java (5.x).
4. Libraries that are supported by Spring IO platform, but library that relies individually.
   (Library is not managed as dependencies in Spring IO platform)
5. Library versions that are applied in the Spring IO platform is a Beta or RC (Release Candidate)
   (Library that explicitly specify the GA version at TERASOLUNA Server Framework for Java (5.x))
6. A library which considers API of Java SE 8 and further versions,as a prerequisite.
   Dependency relation to the library must be explicitly removed in case of versions earlier than Java SE 7

2.1.4 Building blocks of Common Library

Common Library includes `+ alpha` functionalities which are not available in Spring Ecosystem or other dependent libraries included in TERASOLUNA Server Framework for Java (5.x). Basically, application development is possible using TERASOLUNA Server Framework for Java (5.x) without this library but "convenient to have" kind of existence. With the default settings, provided two blank projects, Blank project of multi-project and Blank project of single-project, contains built-in Common Library as shown in the following listing.

No.	Project Name	Summary	Java source-code availability	Blank project of multi-project built-in	Blank project of single-project built-in
(1)	terasoluna-gfw-parent	Provide recommended settings for dependent library management and build plug-in.	No	Yes*1	Yes*1
(2)	terasoluna-gfw-common-libraries	Define structure of the project which includes Java source code, among the common libraries. It is not necessary to add to pom.xml as a dependency.	No	Yes*1	Yes*1
(3)	terasoluna-gfw-dependencies	Define structure of the project which provides only dependency definition (other than terasoluna-gfw-parent), among the common libraries. It is not necessary to add pom.xml as a dependency.	No	L*1	L*1
(4)	terasoluna-gfw-common	Provide a general purpose function that does not depend on Web. When this library is to be used, terasoluna-gfw-common-dependencies is added to pom.xml as a dependency.	Yes	Yes*2	Yes*2

次のページに続く

Table. 2.2 – 前のページからの続き

No.	Project Name	Summary	Java source-code availability	Blank project of multi-project built-in	Blank project of single-project built-in
(5)	terasoluna-gfw-common-dependencies	Provide definition for dependency while using the function offered by terasoluna-gfw-common project.	No	Yes	Yes
(6)	terasoluna-gfw-jodatime	Provide a function that depends on Joda Time. When the library is to be used, terasoluna-gfw-jodatime-dependencies is added to pom.xml as a dependency. (add from 5.0.0)	Yes	Yes*2	Yes*2
(7)	terasoluna-gfw-jodatime-dependencies	Provide dependency definition while using a function offered by terasoluna-gfw-jodatime project.	No	Yes	Yes
(8)	terasoluna-gfw-web	Provide a function used while creating a Web application. Functions that do not depend on View are consolidated. When the library is used, terasoluna-gfw-web-dependencies is added to pom.xml as a dependency.	Yes	Yes*2	Yes*2
(9)	terasoluna-gfw-web-dependencies	Provide dependency definition while using a function offered by terasoluna-gfw-web project.	No	Yes	Yes
(10)	terasoluna-gfw-web-jsp	Provide a function used while creating a Web application which adopts JSP in View. When this library is used, terasoluna-gfw-web-jsp-dependencies is added to pom.xml as a dependency.	Yes	Yes*2	Yes*2
(11)	terasoluna-gfw-web-jsp-dependencies	Provide dependency definition while using a function offered by terasoluna-gfw-web-jsp project.	No	Yes	Yes
(12)	terasoluna-gfw-security-web	Provide extended parts of Spring Security. When the library is used, terasoluna-gfw-security-web-dependencies is added to pom.xml as a dependency.	Yes	Yes*2	Yes*2

次のページに続く

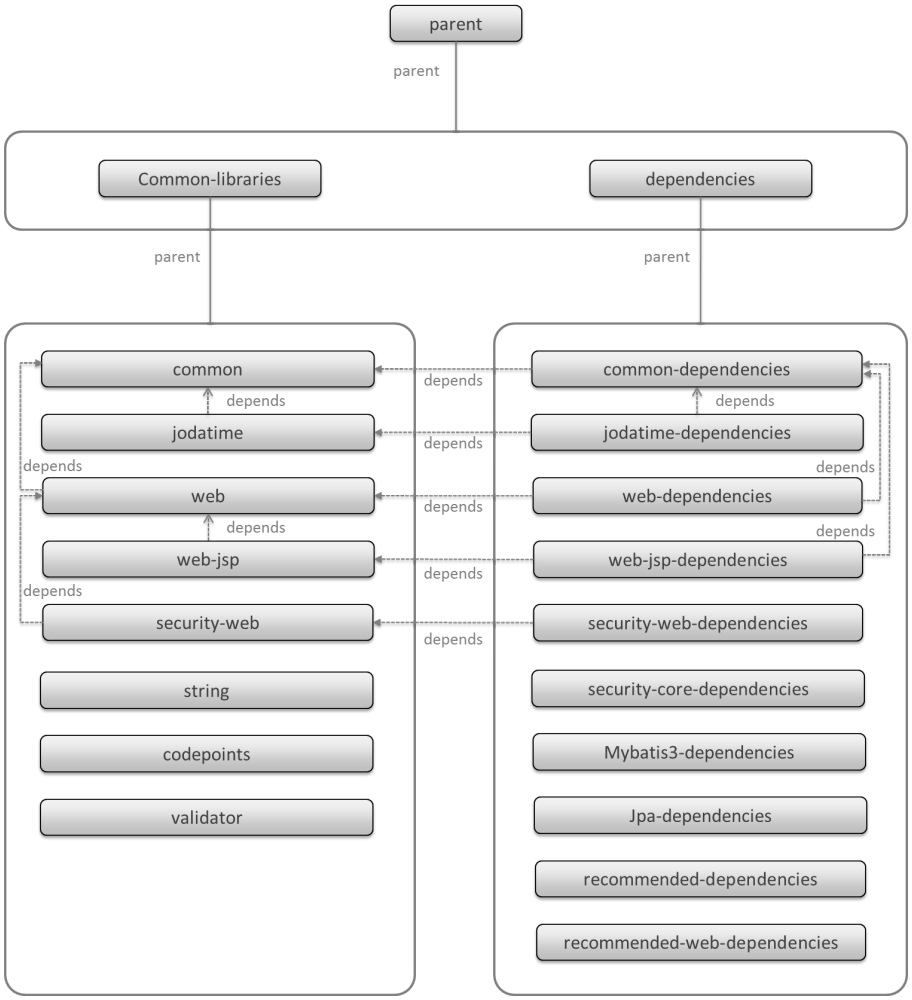
Table. 2.2 – 前のページからの続き

No.	Project Name	Summary	Java source-code availability	Blank project of multi-project built-in	Blank project of single-project built-in
(13)	terasoluna-gfw-security-web-dependencies	Provide dependency definition while using Spring Security (Web related) and dependency definition while using a function offered by terasoluna-gfw-security-web project.	No	Yes	Yes
(14)	terasoluna-gfw-string	Provide a function related to string processing. (added from 5.1.0)	Yes	No	No
(15)	terasoluna-gfw-codepoints	Provide a function to check whether the code points that constitute the targeted string are included in the code point set. (added from 5.1.0)	Yes	No	No
(16)	terasoluna-gfw-validator	Provide by adding a constraint annotation of general-purpose Bean Validation. (added from 5.1.0)	Yes	No	No
(17)	terasoluna-gfw-security-core-dependencies	Provide a dependency definition (other than Web) while using Spring Security.	No	Yes	Yes
(18)	terasoluna-gfw-mybatis3-dependencies	Provide a dependency definition while using MyBatis3.	No	Yes*3	Yes*3
(19)	terasoluna-gfw-jpa-dependencies	Provide a dependency definition while using JPA.	No	Yes*4	Yes*4
(20)	terasoluna-gfw-recommended-dependencies	Provide a dependency definition for recommended libraries that do not depend on web.	No	Yes	Yes
(21)	terasoluna-gfw-recommended-web-dependencies	Provide a dependency definition for web-dependent recommended libraries.	No	Yes	Yes

1. Incorporated as `<parent>` element of each project and not as a `<dependency>`element.
2. Incorporated as a transition dependency from `<dependency>` element and not as a `<dependency>`element.
3. With the default settings of Common Library, when MyBatis3 is used for data access.
4. With the default settings of Common Library, when JPA is used for data access.

The project which does not contain the Java source code, only defines library dependencies.

In addition, project dependencies are as follows:



Note: Excluding exceptions, a project assigned with "dependencies" at the end of project name exists in common library. (For example, terasoluna-gfw-common-dependencies etc corresponding to terasoluna-gfw-common)

In these projects, since a dependency definition for OSS library recommended for use is provided along with dependency definition for common library, it is recommended to add it to pom.xml as a dependency, for the

project assigned with “dependencies” while using a common library.

terasoluna-gfw-common

terasoluna-gfw-common provide following components.

Classification	Component Name	Description
Exception Handling	Exception Class	Provide general exception classes.
	Exception Logger	Provide logger class for logging the exception that occurred in application.
	Exception Code	Provide mechanism (classes) for resolving the exception code (message ID) that corresponds to the exception class.
	Exception Logging Interceptor	Provide interceptor class of AOP for logging the exception that occurred in domain layer.
System Date	System Date Time Factory	Provide classes for retrieving the system date time.
Codelist	CodeList	Provide classes for generating CodeList.
Database Access (Common)	Query Escape	Provide class for escape processing of value to bind into the SQL and JPQL.
	Sequencer	Provide classes for retrieving the sequence value.

terasoluna-gfw-string

terasoluna-gfw-string provides following components.

Classification	Component Name	Description
String Processing	Half width to full width conversion	Provide a class which carries out a process wherein half width character of input string is converted to full width and a process wherein full width character is converted to half width based on mapping table of half width string and full width string.

terasoluna-gfw-codepoints

terasoluna-gfw-codepoints provides following components.

Classification	Component Name	Description
String Processing	Codepoint check	Provide a class which checks whether the codepoint constituting target string is incorporated in the codepoint set.
Input Validation	Bean validation constraint annotation for codepoint check	Provide constraint annotation to carry out Bean validation for codepoint check.

terasoluna-gfw-validator

terasoluna-gfw-validator provides following components.

Classification	Component name	Description
Input Validation	Bean Validation constraint annotation for byte length check	Provide a constraint annotation to check whether the byte length in the character code of input value is less than or equal to specified maximum value, and greater than or equal to specified minimum value, using Bean Validation.
	Bean Validation constraint annotation for field value comparison correlation check	Provide a constraint annotation to check magnitude correlation of two fields using Bean Validation.

terasoluna-gfw-jodatime

terasoluna-gfw-jodatime provide following components.

Classification	Component Name	Description
System Date	System Date Time Factory for Joda Time	Provide classes for retrieving the system date time using the Joda Time API.

terasoluna-gfw-web

terasoluna-gfw-web provide following components.

Classification	Component Name	Description
Double Submit Protection	Transaction Token Check	Provide mechanism (classes) for protecting Web Application from double submitting of request.
Exception Handling	Exception Handler	Provide exception handler class(sub class of class that provided by Spring MVC) for integrating with exception handling components that provided from common library.
	Exception Logging Interceptor	Provide interceptor class of AOP for logging the exception that handled by Spring MVC.
Codelist	Populate CodeList interceptor	Provide interceptor class of Spring MVC for storing CodeList information into request scope, for the purpose of retrieving CodeList from View.
File Download	General Download View	Provide abstract class for retrieving data from input stream and writing to stream for download.
Logging	ServletFilter for storing Tracking ID	Provide Servlet Filter class for setting Tracking ID into MDC(Mapped Diagnostic Context) and request scope and response header, for the purpose of improving traceability. (If does not exist a Tracking ID in request header, generate a Tracking ID by this component)
	General ServletFilter for storing MDC	Provide abstract class for storing any value into Logger's MDC
	ServletFilter for clearing MDC	Provide ServletFilter class for clearing information that stored in Logger's MDC.

terasoluna-gfw-web-jsp

terasoluna-gfw-web-jsp provides following components.

Classification	Component name	Description
Double Submit Protection	JSP tag for transaction token output	Provide a JSP tag library to output transaction tokens as hidden items.
Pagination	JSP Tag for displaying Pagination Links	Provide JSP Tag Library for displaying Pagination Links using classes that provided by Spring Data Commons.
Message Management	JSP Tag for displaying Result Messages	Provide JSP Tag Library for displaying Result Messages.
EL Functions	EL Functions for XSS countermeasures	Provide EL Functions for XSS countermeasures.
	EL Functions for URL	Provide EL Functions for URL as URL encoding.
	EL Functions for DOM conversion	Provide EL Functions for DOM conversion.
	EL Functions for Utilities	Provide EL Functions for general utilities processing.

terasoluna-gfw-security-web

terasoluna-gfw-security-web provide following components.

Classification	Component Name	Description
Logging	ServletFilter for storing name of authenticated user	Provide ServletFilter class for setting name of authenticated user into MDC, for the purpose of improving traceability.

2.2 Overview of Spring MVC Architecture

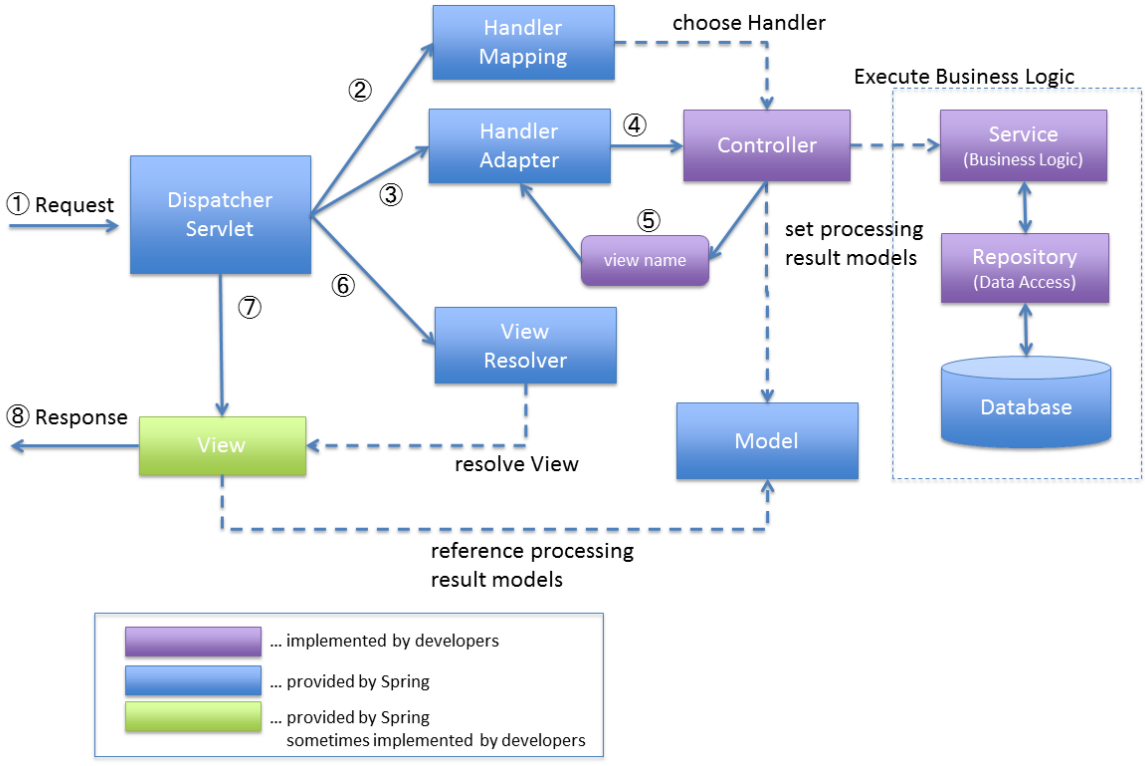
Official website of Spring MVC says the following

Spring Reference Document.

> Spring's web MVC framework is, like many other web MVC frameworks, request-driven, designed around a central Servlet that dispatches requests to controllers and offers other functionality that facilitates the development of web applications. Spring's DispatcherServlet however, does more than just that. It is completely integrated with the Spring IoC container and as such allows you to use every other feature that Spring has.

2.2.1 Overview of Spring MVC Processing Sequence

The processing flow of Spring MVC from receiving the request till the response is returned is shown in the following diagram.



1. `DispatcherServlet` receives the request.
2. `DispatcherServlet` dispatches the task of selecting an appropriate `controller` to `HandlerMapping`. `HandlerMapping` selects the controller which is mapped to the incoming request URL and returns the `(selected Handler)` and `Controller` to `DispatcherServlet`.

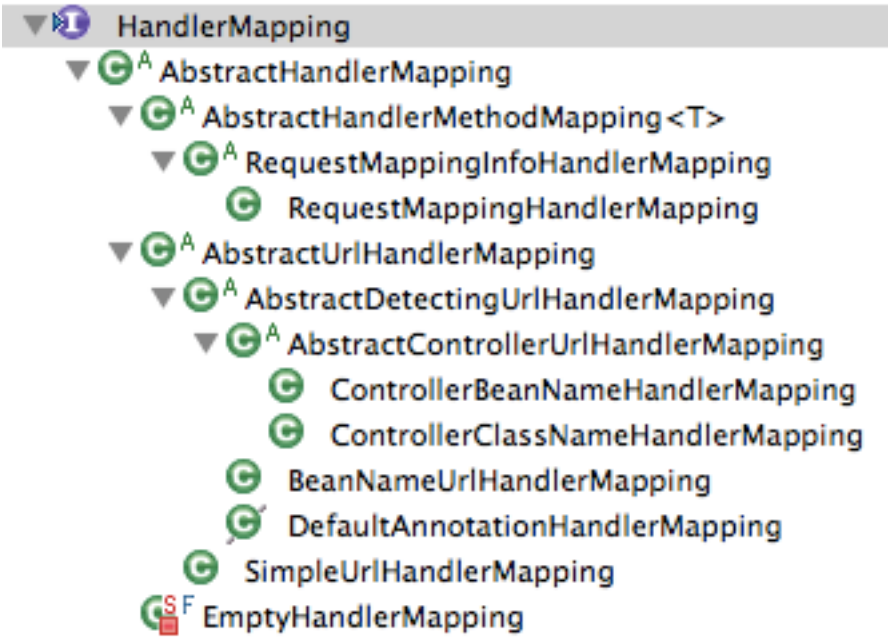
3. `DispatcherServlet` dispatches the task of executing of business logic of `Controller` to `HandlerAdapter`.
4. `HandlerAdapter` calls the business logic process of `Controller`.
5. `Controller` executes the business logic, sets the processing result in `Model` and returns the logical name of view to `HandlerAdapter`.
6. `DispatcherServlet` dispatches the task of resolving the `View` corresponding to the View name to `ViewResolver`. `ViewResolver` returns the `View` mapped to View name.
7. `DispatcherServlet` dispatches the rendering process to returned `View`.
8. `View` renders `Model` data and returns the response.

2.2.2 Implementation of each component

Among the components explained previously, the extendable components are implemented.

Implementation of HandlerMapping

Class hierarchy of `HandlerMapping` provided by Spring framework is shown below.

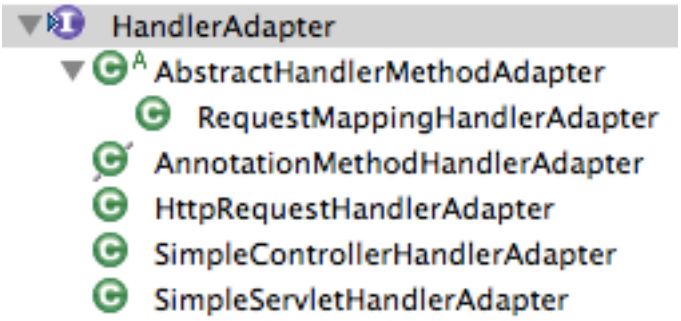


Usually `org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerMapping` is used. This class reads `@RequestMapping` annotation from the `Controller` and uses the method of `Controller` that matches with URL as Handler class.

In Spring3.1, `RequestMappingHandlerMapping` is enabled by default when `<mvc:annotation-driven>` is set in Bean definition file read by `DispatcherServlet`. (For the settings which get enabled with the use of `<mvc:annotation-driven>` annotation, refer Web MVC framework.)

Implementation of HandlerAdapter

Class hierarchy of `HandlerAdapter` provided by Spring framework is shown below.



Usually `org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter` is used. `RequestMappingHandlerAdapter` class calls the method of handler class (`Controller`) selected by `HandlerMapping`. In Spring 3.1, this class is also configured by default throught `<mvc:annotation-driven>`.

Implementation of ViewResolver

Classes that implement `ViewResolver` provided by Spring framework and dependent libraries are shown below.

Normally (When JSP is used),

- `org.springframework.web.servlet.view.InternalResourceViewResolver` is used,

however, when template engine Tiles is to be used

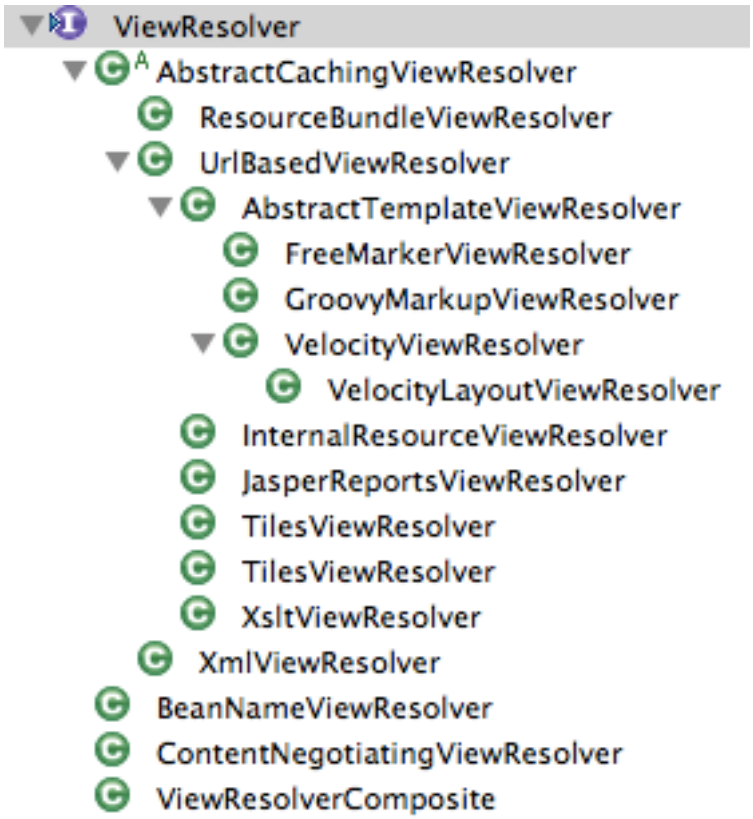
- `org.springframework.web.servlet.view.tiles3.TilesViewResolver`

and when stream is to be returned for file download

- `org.springframework.web.servlet.view.BeanNameViewResolver`

Thereby, it is required to use different viewResolver based on the type of the View that needs to be returned.

When `View` of multiple types is to be handled, multiple definitions of `ViewResolver` are required.



A typical example of using multiple `ViewResolver` is the screen application for which file download process exists.

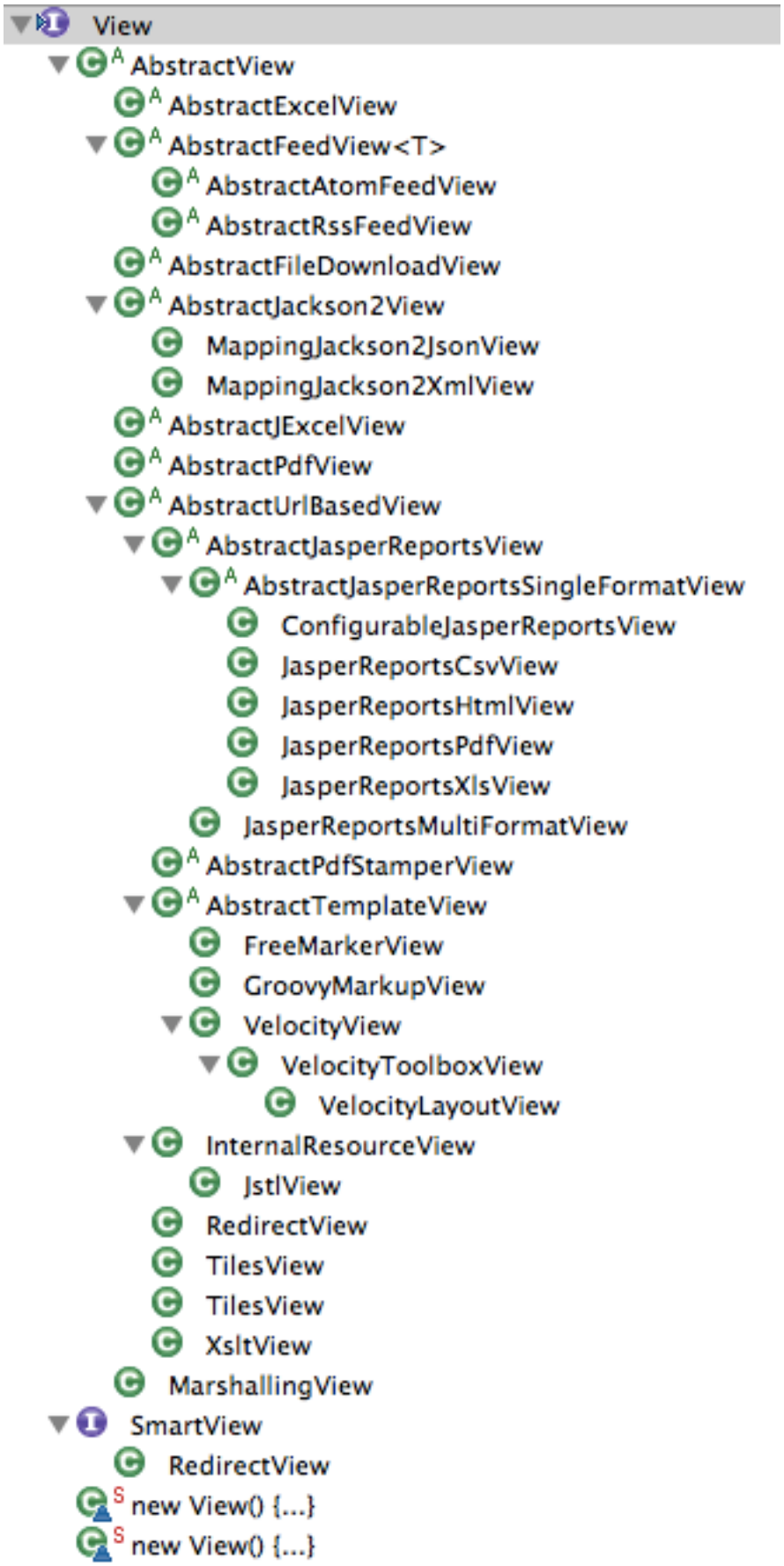
For screen (JSP), `View` is resolved using `InternalResourceViewResolver` and for File download `View` is resolved using `BeanNameViewResolver`.

For details, refer *File Download*.

Implementation of View

Classes that implement `View` provided by Spring framework and its dependent libraries are shown below.

`View` changes with the type of response to be returned. When JSP is to be returned, `org.springframework.web.servlet.view.JstlView` is used. When `View` not provided by Spring framework and its dependent libraries are to be handled, it is necessary to extend the class in which `View` interface is implemented. For details, refer *File Download*.


View
AbstractView
AbstractExcelView
AbstractFeedView<T>
AbstractAtomFeedView
AbstractRssFeedView
AbstractFileDownloadView
AbstractJackson2View
MappingJackson2JsonView
MappingJackson2XmlView
AbstractJExcelView
AbstractPdfView
AbstractUrlBasedView
AbstractJasperReportsView
AbstractJasperReportsSingleFormatView
ConfigurableJasperReportsView
JasperReportsCsvView
JasperReportsHtmlView
JasperReportsPdfView
JasperReportsXlsView
JasperReportsMultiFormatView
AbstractPdfStamperView
AbstractTemplateView
FreeMarkerView
GroovyMarkupView
VelocityView
VelocityToolboxView
VelocityLayoutView
InternalResourceView
JstlView
RedirectView
TilesView
TilesView
XsltView
MarshallingView
SmartView
RedirectView
new View() {...}
new View() {...}

2.3 First application based on Spring MVC

Before entering into the advanced usage of Spring MVC, it is better to understand Spring MVC by actually trying hands-on web application development using Spring MVC.

2.3.1 Prerequisites

The description of this chapter has been verified on the following environment. (For other environments, replace the contents mentioned here appropriately)

Classification	Product
OS	Windows 7
JVM	Java 1.8
IDE	Spring Tool Suite 3.6.4.RELEASE (Hereafter referred as [STS])
Build Tool	Apache Maven 3.3.9 (Hereafter referred as [Maven])
Application Server	Pivotal tc Server Developer Edition v3.1 (enclosed in STS)
Web Browser	Google Chrome 46.0.2490.80 m

Note: To connect the internet via proxy server, STS Proxy settings and Maven Proxy settings are required for the following operations.

2.3.2 Create a New Project

Create project using *mvn archetype:generate* on internet.

```
mvn archetype:generate -B^
 -DarchetypeGroupId=org.terasoluna.gfw.blank^
 -DarchetypeArtifactId=terasoluna-gfw-web-blank-archetype^
 -DarchetypeVersion=5.3.0.RELEASE^
 -DgroupId=com.example.helloworld^
 -DartifactId=helloworld^
 -Dversion=1.0.0-SNAPSHOT
```

Here, created a project on Windows environment.

```
C:\work>mvn archetype:generate -B^
More?  -DarchetypeGroupId=org.terasoluna.gfw.blank^
More?  -DarchetypeArtifactId=terasoluna-gfw-web-blank-archetype^
More?  -DarchetypeVersion=5.3.0.RELEASE^
More?  -DgroupId=com.example.helloworld^
More?  -DartifactId=helloworld^
More?  -Dversion=1.0.0-SNAPSHOT
[INFO] Scanning for projects...
```

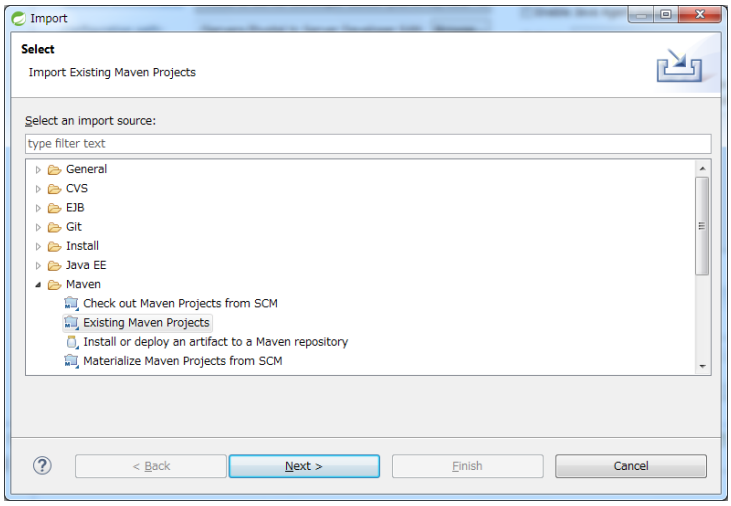
```
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building Maven Stub Project (No POM) 1
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] >>> maven-archetype-plugin:2.4:generate (default-cli) > generate-sources @␣
↪standalone-pom >>>
[INFO]
[INFO] <<< maven-archetype-plugin:2.4:generate (default-cli) < generate-sources @␣
↪standalone-pom <<<
[INFO]
[INFO] --- maven-archetype-plugin:2.4:generate (default-cli) @ standalone-pom ---
[INFO] Generating project in Batch mode
[INFO] Archetype repository not defined. Using the one from [org.terasoluna.gfw.
↪blank:terasoluna-gfw-web-blank-archetype:5.3.0.RELEASE] found in catalog remote
[INFO] ----------------------------------------------------------------------------
[INFO] Using following parameters for creating project from Archetype: terasoluna-
↪gfw-web-blank-archetype:5.3.0.RELEASE
[INFO] ----------------------------------------------------------------------------
[INFO] Parameter: groupId, Value: com.example.helloworld
[INFO] Parameter: artifactId, Value: helloworld
[INFO] Parameter: version, Value: 1.0.0-SNAPSHOT
[INFO] Parameter: package, Value: com.example.helloworld
[INFO] Parameter: packageInPathFormat, Value: com/example/helloworld
[INFO] Parameter: package, Value: com.example.helloworld
[INFO] Parameter: version, Value: 1.0.0-SNAPSHOT
[INFO] Parameter: groupId, Value: com.example.helloworld
[INFO] Parameter: artifactId, Value: helloworld
[INFO] project created from Archetype in dir: C:\work\helloworld
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 9.266 s
[INFO] Finished at: 2017-02-10T11:46:01+09:00
[INFO] Final Memory: 13M/188M
[INFO] ------------------------------------------------------------------------
C:\work>
```

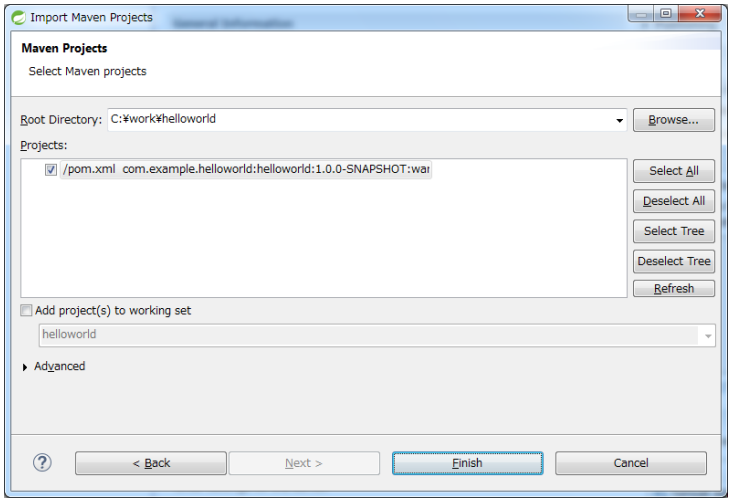
Select the archetype created project from STS menu [File] -> [Import] -> [Maven] -> [Existing Maven Projects] -> [Next].

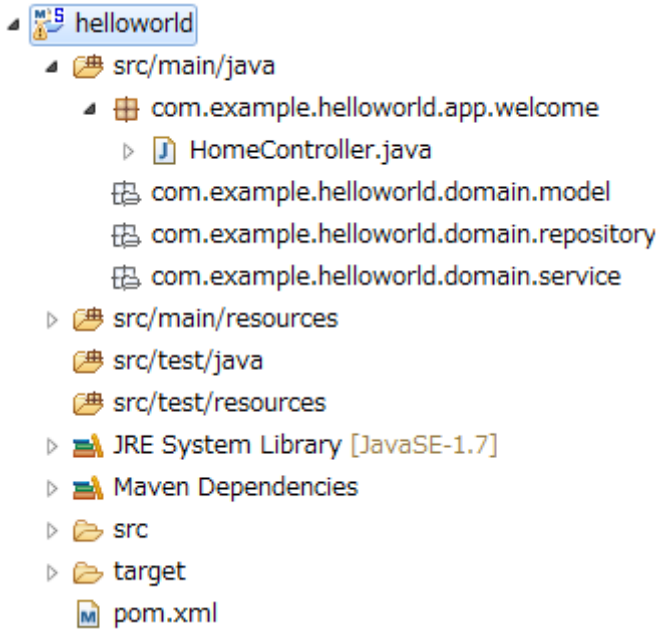
Click on [Finish] by selecting `C:\work\helloworld` in Root Directory and selecting pom.xml of helloworld in Projects.

Following project is generated in the Package Explorer.

To understand the configuration of Spring MVC, the generated Spring MVC configuration file (src/main/resources/META-INF/spring/spring-mvc.xml) is described briefly.


Import
Select
Import Existing Maven Projects
Select an import source:
type filter text
General
CVS
EJB
Git
Install
Java EE
Maven
Check out Maven Projects from SCM
Existing Maven Projects
Install or deploy an artifact to a Maven repository
Materialize Maven Projects from SCM
< Back
Next >
Finish
Cancel


Import Maven Projects
Maven Projects
Select Maven projects
Root Directory:
C:¥work¥helloworld
Browse...
Projects:
/pom.xml com.example.helloworld:helloworld:1.0.0-SNAPSHOT:war
Select All
Deselect All
Select Tree
Deselect Tree
Refresh
Add project(s) to working set
helloworld
Advanced
< Back
Next >
Finish
Cancel


helloworld
src/main/java
com.example.helloworld.app.welcome
HomeController.java
com.example.helloworld.domain.model
com.example.helloworld.domain.repository
com.example.helloworld.domain.service
src/main/resources
src/test/java
src/test/resources
JRE System Library [JavaSE-1.7]
Maven Dependencies
src
target
pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:mvc="http://www.springframework.org/schema/mvc"
    xmlns:util="http://www.springframework.org/schema/util"
    xmlns:aop="http://www.springframework.org/schema/aop"
    xsi:schemaLocation="http://www.springframework.org/schema/mvc http://www.
↪springframework.org/schema/mvc/spring-mvc.xsd
        http://www.springframework.org/schema/beans http://www.springframework.org/
↪schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/util http://www.springframework.org/
↪schema/util/spring-util.xsd
        http://www.springframework.org/schema/context http://www.springframework.
↪org/schema/context/spring-context.xsd
        http://www.springframework.org/schema/aop http://www.springframework.org/
↪schema/aop/spring-aop.xsd
    ">

    <context:property-placeholder
        location="classpath*:/META-INF/spring/*.properties" />

    <!-- (1) Enables the Spring MVC @Controller programming model -->
    <mvc:annotation-driven>
        <mvc:argument-resolvers>
            <bean
                class="org.springframework.data.web.
↪PageableHandlerMethodArgumentResolver" />
            <bean
                class="org.springframework.security.web.method.annotation.
↪AuthenticationPrincipalArgumentResolver" />
        </mvc:argument-resolvers>
        <!-- workaround to CVE-2016-5007. -->
        <mvc:path-matching path-matcher="pathMatcher" />
    </mvc:annotation-driven>

    <mvc:default-servlet-handler />

    <!-- (2) -->
    <context:component-scan base-package="com.example.helloworld.app" />

    <mvc:resources mapping="/resources/**"
        location="/resources/,classpath:META-INF/resources/"
        cache-period="#{60 * 60}" />

    <mvc:interceptors>
        <mvc:interceptor>
            <mvc:mapping path="/**" />
            <mvc:exclude-mapping path="/resources/**" />
            <mvc:exclude-mapping path="/**/*.html" />
```

```
            <bean
                class="org.terasoluna.gfw.web.logging.TraceLoggingInterceptor" />
        </mvc:interceptor>
        <mvc:interceptor>
            <mvc:mapping path="/**" />
            <mvc:exclude-mapping path="/resources/**" />
            <mvc:exclude-mapping path="/**/*.html" />
            <bean
                class="org.terasoluna.gfw.web.token.transaction.
↪TransactionTokenInterceptor" />
        </mvc:interceptor>
        <mvc:interceptor>
            <mvc:mapping path="/**" />
            <mvc:exclude-mapping path="/resources/**" />
            <mvc:exclude-mapping path="/**/*.html" />
            <bean class="org.terasoluna.gfw.web.codelist.CodeListInterceptor">
                <property name="codeListIdPattern" value="CL_.+" />
            </bean>
        </mvc:interceptor>
        <!--  REMOVE THIS LINE IF YOU USE JPA
        <mvc:interceptor>
            <mvc:mapping path="/**" />
            <mvc:exclude-mapping path="/resources/**" />
            <mvc:exclude-mapping path="/**/*.html" />
            <bean
                class="org.springframework.orm.jpa.support.
↪OpenEntityManagerInViewInterceptor" />
        </mvc:interceptor>
            REMOVE THIS LINE IF YOU USE JPA  -->
    </mvc:interceptors>

    <!-- (3) Resolves views selected for rendering by @Controllers to .jsp␣
↪resources in the /WEB-INF/views directory -->
    <!-- Settings View Resolver. -->
    <mvc:view-resolvers>
        <mvc:jsp prefix="/WEB-INF/views/" />
    </mvc:view-resolvers>

    <bean id="requestDataValueProcessor"
        class="org.terasoluna.gfw.web.mvc.support.
↪CompositeRequestDataValueProcessor">
        <constructor-arg>
            <util:list>
                <bean
                    class="org.springframework.security.web.servlet.support.csrf.
↪CsrfRequestDataValueProcessor" />
                <bean
                    class="org.terasoluna.gfw.web.token.transaction.
↪TransactionTokenRequestDataValueProcessor" />
            </util:list>
```

```
        </constructor-arg>
    </bean>

    <!-- Setting Exception Handling. -->
    <!-- Exception Resolver. -->
    <bean id="systemExceptionResolver"
        class="org.terasoluna.gfw.web.exception.SystemExceptionResolver">
        <property name="exceptionCodeResolver" ref="exceptionCodeResolver" />
        <!-- Setting and Customization by project. -->
        <property name="order" value="3" />
        <property name="exceptionMappings">
            <map>
                <entry key="ResourceNotFoundException" value="common/error/
↪resourceNotFoundError" />
                <entry key="BusinessException" value="common/error/businessError" /
↪>
                <entry key="InvalidTransactionTokenException" value="common/error/
↪transactionTokenError" />
                <entry key=".DataAccessException" value="common/error/
↪dataAccessError" />
            </map>
        </property>
        <property name="statusCodes">
            <map>
                <entry key="common/error/resourceNotFoundError" value="404" />
                <entry key="common/error/businessError" value="409" />
                <entry key="common/error/transactionTokenError" value="409" />
                <entry key="common/error/dataAccessError" value="500" />
            </map>
        </property>
        <property name="defaultErrorView" value="common/error/systemError" />
        <property name="defaultStatusCode" value="500" />
    </bean>
    <!-- Setting AOP. -->
    <bean id="handlerExceptionResolverLoggingInterceptor"
        class="org.terasoluna.gfw.web.exception.
↪HandlerExceptionResolverLoggingInterceptor">
        <property name="exceptionLogger" ref="exceptionLogger" />
    </bean>
    <aop:config>
        <aop:advisor advice-ref="handlerExceptionResolverLoggingInterceptor"
            pointcut="execution(* org.springframework.web.servlet.
↪HandlerExceptionResolver.resolveException(..))" />
    </aop:config>

    <!-- Setting PathMatcher. -->
    <bean id="pathMatcher" class="org.springframework.util.AntPathMatcher">
        <property name="trimTokens" value="false" />
    </bean>
```

```
</beans>
```

Sr. No.	Description
(1)	Default settings of Spring MVC are configured by defining <mvc:annotation-driven>. Refer to the official website Enabling the MVC Java Config or the MVC XML Namespace for default configuration of Spring framework.
(2)	Define the package which will be target of searching components used in Spring MVC.

(3)	Define the location of the JSP by specifying ViewResolver for JSP.
	Tip: <mvc:view-resolvers> element is a XML element that added from Spring Framework 4.1. By using <mvc:view-resolvers> element, it is possible to define ViewResolver simply. The definition example of using the conventional <bean> element is shown below.
	`<bean id="viewResolver" class="org.springframework.web.servlet.view.InternalResourceViewReso`
	`<property name="prefix" value="/WEB-INF/views/" />`
	`<property name="suffix" value=".jsp" />`
	`</bean>`

Next, Controller (com.example.helloworld.app.welcome.HelloController) for displaying the Welcome page is described briefly.

```
package com.example.helloworld.app.welcome;

import java.text.DateFormat;
import java.util.Date;
import java.util.Locale;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

/**
 * Handles requests for the application home page.
```

```
 */
@Controller // (4)
public class HelloController {

    private static final Logger logger = LoggerFactory
            .getLogger(HelloController.class);

    /**
     * Simply selects the home view to render by returning its name.
     */
    @RequestMapping(value = "/", method = {RequestMethod.GET, RequestMethod.POST}
↪) // (5)
    public String home(Locale locale, Model model) {
        logger.info("Welcome home! The client locale is {}.", locale);

        Date date = new Date();
        DateFormat dateFormat = DateFormat.getDateTimeInstance(DateFormat.LONG,
                DateFormat.LONG, locale);

        String formattedDate = dateFormat.format(date);

        model.addAttribute("serverTime", formattedDate); // (6)

        return "welcome/home"; // (7)
    }

}
```

Sr. No.	Description
(4)	It can be read automatically by DI container if `@Controller` annotation is used. As stated earlier in [explanation of Spring MVC configuration files (2)], it is the target of component-scan.
(5)	It gets executed when the HTTP method is GET or POST and the Resource is (or request URL) is "/".
(6)	Set `Model` object to be delivered to View.
(7)	Return View name. "WEB-INF/views/welcome/home.jsp" is rendered as per the configuration [Explanation of Spring MVC configuration files (3)] .

Lastly, JSP (`src/main/webapp/WEB-INF/views/welcome/home.jsp`) for displaying the Welcome page is described briefly.

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>Home</title>
<link rel="stylesheet"
    href="${pageContext.request.contextPath}/resources/app/css/styles.css">
</head>
<body>
    <div id="wrapper">
        <h1>Hello world!</h1>
        <p>The time on the server is ${serverTime}.</p> <%-- (8) --%>
    </div>
</body>
</html>
```

Sr. No.	Description
(8)	As stated earlier in [explanation of Controller (6)] the object (serverTime) set in Model is stored in HttpServletRequest. Therefore the value passed from Controller can be output by mentioning `${serverTime}` in JSP. **However about ${XXX}, it is necessary to perform HTML escaping since there is possibility of XSS attack.**

2.3.3 Run on Server

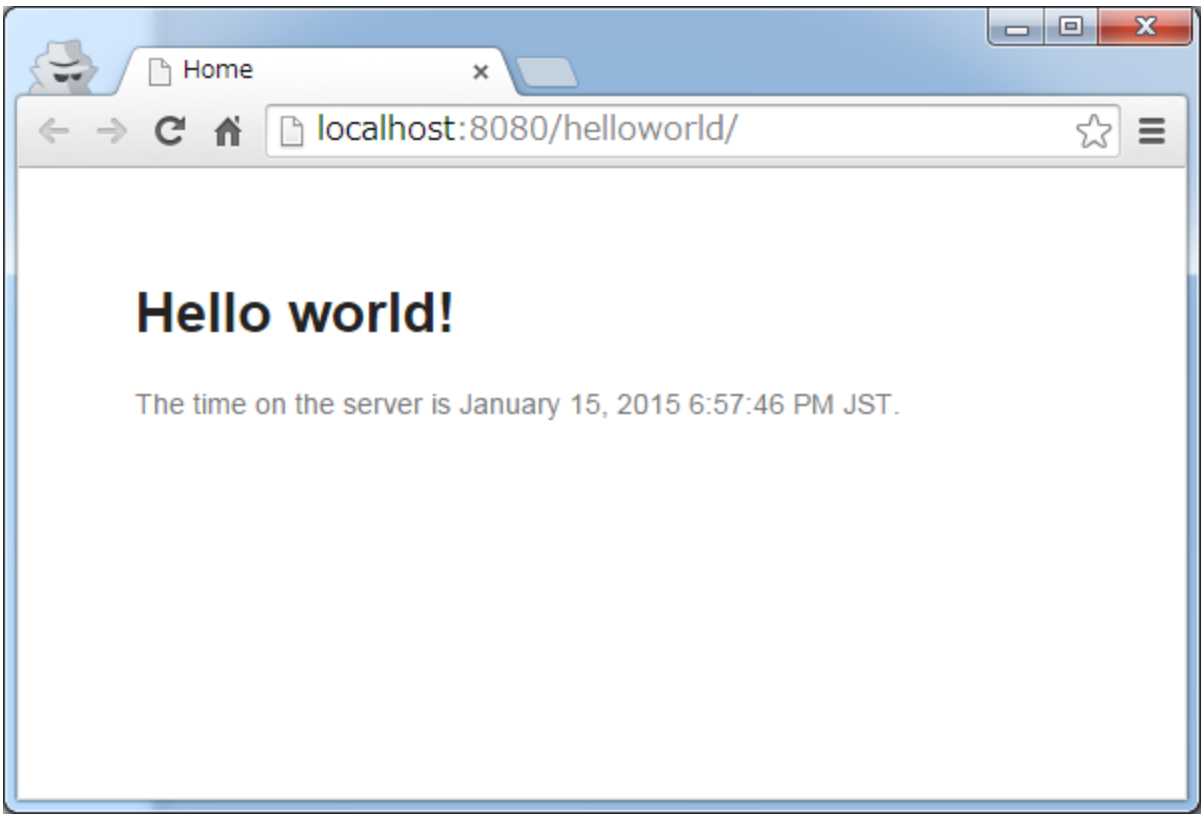
Right click “helloworld” project in STS, and start helloworld project by executing “Run As” -> “Run On Server” -> “localhost” -> “Pivotal tc Server Developer Edition v3.0” -> “Finish”.

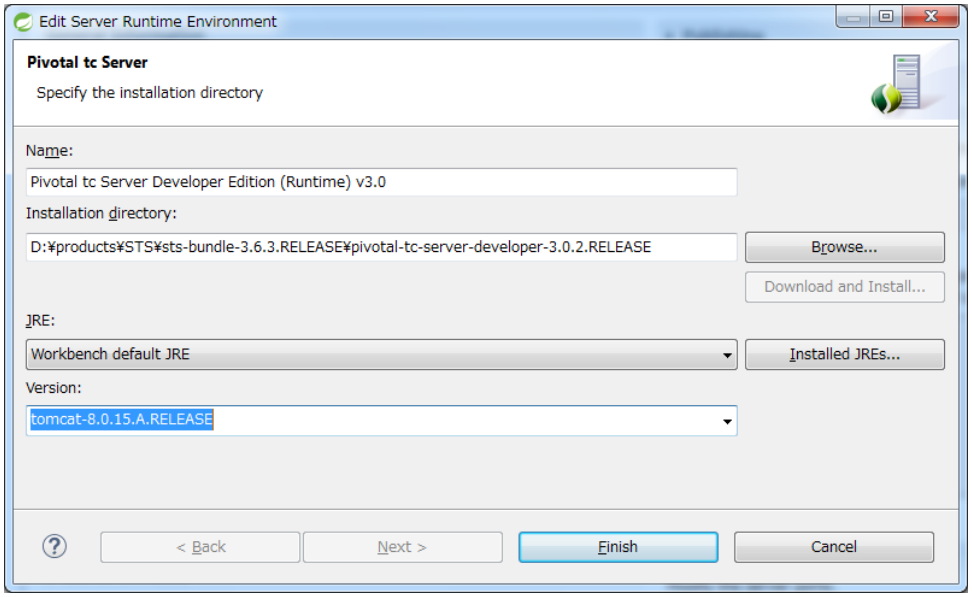
Enter “http://localhost:8080/helloworld/” in browser to display the following screen.

Note: Since tc Server internally uses the Tomcat, it is possible to choose below two versions in the STS.

- tomcat-8.0.15.A.RELEASE (version used by default)
- tomcat-7-0.57.A.RELEASE

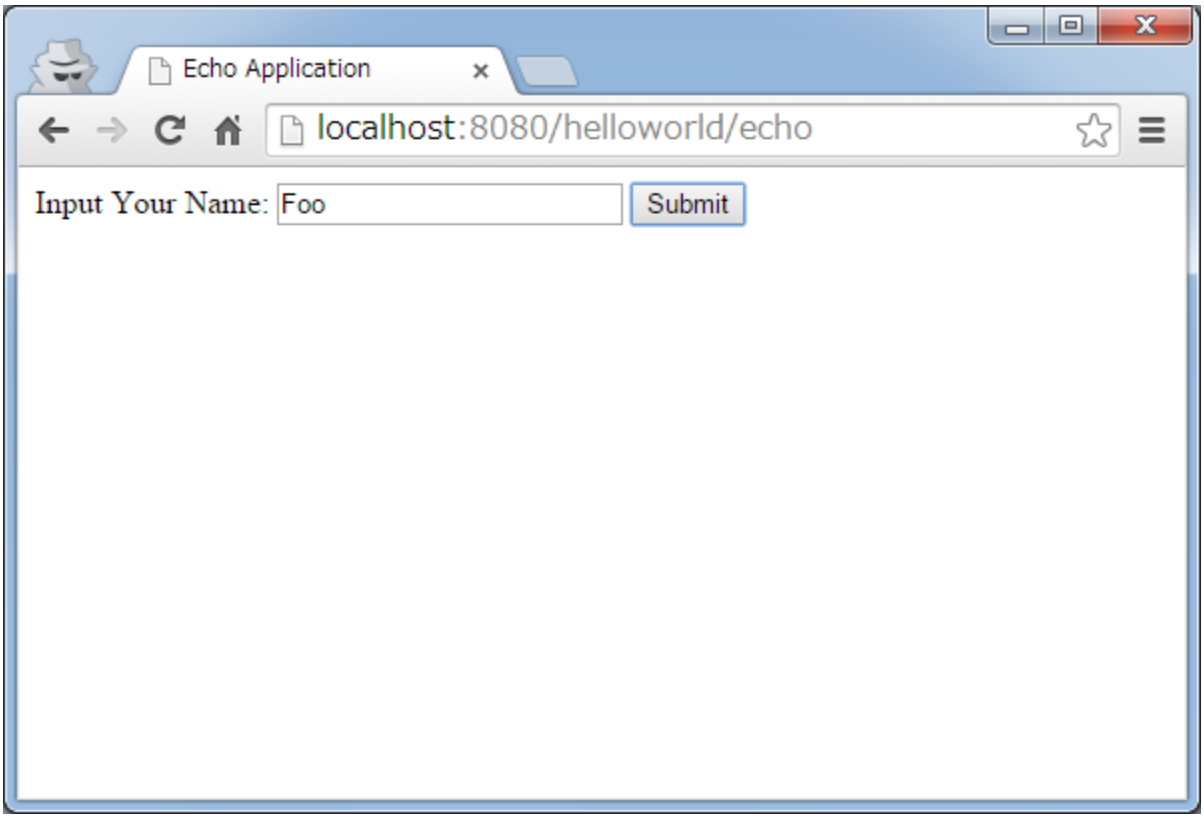
If you want to switch the Tomcat to be used, change the [Version] field of ts Server by opening the [Edit Server Runtime Environment] dialog box. Java (JRE) version can also be changed from this dialog box.


Home
localhost:8080/helloworld/
Hello world!
The time on the server is January 15, 2015 6:57:46 PM JST.


Edit Server Runtime Environment
Pivotal tc Server
Specify the installation directory
Name:
Pivotal tc Server Developer Edition (Runtime) v3.0
Installation directory:
D:¥products¥STS¥sts-bundle-3.6.3.RELEASE¥pivotal-tc-server-developer-3.0.2.RELEASE
Browse...
Download and Install...
JRE:
Workbench default JRE
Installed JREs...
Version:
tomcat-8.0.15.A.RELEASE
< Back
Next >
Finish
Cancel

2.3.4 Create an Echo Application

Lets go ahead and create a simple application. It is a typical eco application in which message will be displayed if name is entered in the text field as given below.

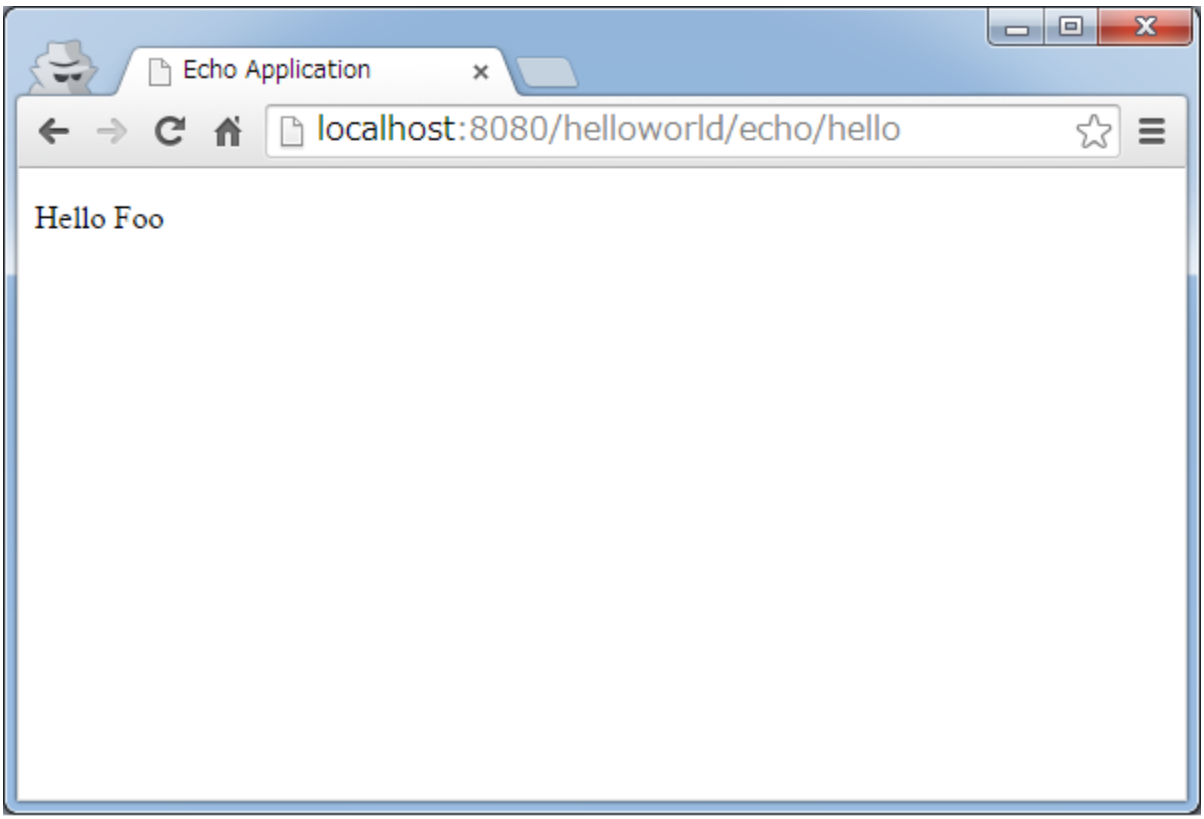


Creating a form object

First create a form object to accept the value of text field.

Create `EchoForm` class under `com.example.helloworld.app.echo` package. It is a simple JavaBean that has only one1 property.

```
package com.example.helloworld.app.echo;

import java.io.Serializable;

public class EchoForm implements Serializable {
    private static final long serialVersionUID = 2557725707095364445L;
```




```
    private String name;

    public void setName(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }
}
```

Create a Controller

Next, create the Controller class.

create the `EchoController` class under `com.example.helloworld.app.echo` package.

```
package com.example.helloworld.app.echo;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

@Controller
@RequestMapping("echo")
public class EchoController {

    @ModelAttribute // (1)
    public EchoForm setUpEchoForm() {
        EchoForm form = new EchoForm();
        return form;
    }

    @RequestMapping // (2)
    public String index(Model model) {
        return "echo/index"; // (3)
    }

    @RequestMapping(value = "hello", method = RequestMethod.POST) // (4)
    public String hello(EchoForm form, Model model) {// (5)
        model.addAttribute("name", form.getName()); // (6)
        return "echo/hello";
    }
}
```

Sr. No.	Description
(1)	Add `@ModelAttribute` annotation to the method. Return value of such a method is automatically added to the Model. Attribute name of the Model can be specified in `@ModelAttribute`, but the class name with the first letter in lower case is the default attribute name. In this case it will be "echoForm". This attribute name must match with the value of `modelAttribute` of `form:form tag`.
(2)	When nothing is specified in `value` attribute of `@RequestMapping` annotation at the method level, it is mapped to `@RequestMapping` added at class level. In this case, `index` method is called, if "<contextPath>/echo" is accessed. When nothing is set in `method` attribute, mapping is done for any HTTP method.
(3)	Since "echo/index" is returned as View name, "WEB-INF/views/echo/index.jsp" is rendered by the ViewResolver.
(4)	Since "hello" is specified in `value` attribute and `RequestMethod.POST` is specified in `method` attribute of the `@RequestMapping` annotation method, if "<contextPath>/echo/hello" is accessed with POST method, `hello` method is called.
(5)	EchoForm object added to the model in (1) is passed as argument.
(6)	`name` entered in form is passed as it is to the View.

Note: The value specified in the `method` attribute of `@RequestMapping` annotation is generally varied by how the data transmitted from the client.

- POST method in case of storing data to the server (in case of updating process).
- GET method or unspecified (any method) in case of not storing data to the server (in case of referring process).

In echo application,

- `index` method is not going to save data to the server, it is unspecified (any method)

- `hello` method is going to save data into `Model` object, it is POST method

etc are specified.

Create JSP Files

Finally create JSP for input screen and output screen. Each file path must match with View name as follows.

Create input screen (src/main/webapp/WEB-INF/views/echo/index.jsp).

```
<!DOCTYPE html>
<html>
<head>
<title>Echo Application</title>
</head>
<body>
  <%-- (1) --%>
  <form:form modelAttribute="echoForm" action="${pageContext.request.contextPath}/
↪echo/hello">
    <form:label path="name">Input Your Name:</form:label>
    <form:input path="name" />
    <input type="submit" />
  </form:form>
</body>
</html>
```

Sr. No.	Description
(1)	HTML form is constructed using tag library. Specify the name of form object created by Controller in `modelAttribute` attribute. Refer here for tag library.

Note: POST method is used if `method` attribute of `<form:form>` tag is omitted.

The generated HTML is as follows

```
<!DOCTYPE html>
<html>
<head>
<title>Echo Application</title>
```

```
</head>
<body>
  <form id="echoForm" action="/helloworld/echo/hello" method="post">
    <label for="name">Input Your Name:</label>
    <input id="name" name="name" type="text" value=""/>
    <input type="submit" />
  <input type="hidden" name="_csrf" value="43595f38-3edd-4c08-843b-3c31a00d2b15" />
</form>
</body>
</html>
```

Create output screen (src/main/webapp/WEB-INF/views/echo/hello.jsp).

```
<!DOCTYPE html>
<html>
<head>
<title>Echo Application</title>
</head>
<body>
  <p>
    Hello <c:out value="${name}" /> <%-- (2) --%>
  </p>
</body>
</html>
```

Sr. No.	Description
(2)	Output “name” passed from Controller. Take countermeasures for XSS using `c:out` tag.

Note: Here, XSS countermeasure taken using `c:out` standard tag, however, `f:h()` function provided in common library can be used easily. Refer to *XSS Countermeasures* for details.

Implementation of Eco application is completed here.

Start the server and form will be displayed by accessing “http://localhost:8080/helloworld/echo”.

Implement Input Validation

Till this point Input validation is not implemented in this application. In Spring MVC, Bean Validation and annotation based input validation can be easily implemented. For example name input validation is performed in Eco Application.

Add annotation for Input check rule to the `name` of `EchoForm`.

```
package com.example.helloworld.app.echo;

import java.io.Serializable;

import javax.validation.constraints.NotNull;
import javax.validation.constraints.Size;

public class EchoForm implements Serializable {
    private static final long serialVersionUID = 2557725707095364445L;

    @NotNull // (1)
    @Size(min = 1, max = 5) // (2)
    private String name;

    public void setName(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }
}
```

Sr. No.	Description
(1)	Whether `name` parameter exists in HTTP request or not can be checked by `@NotNull` annotation.
(2)	Whether the size of `name` is more than or equal to 1 and less than or equal to 5 can be checked by `@Size(min = 1,max = 5)`.

Implement the input check and the error handling when an error occurs in the input check.

```
package com.example.helloworld.app.echo;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.BindingResult;
import org.springframework.validation.annotation.Validated;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

@Controller
@RequestMapping("echo")
public class EchoController {

    @ModelAttribute
    public EchoForm setUpEchoForm() {
        EchoForm form = new EchoForm();
        return form;
    }

    @RequestMapping
    public String index(Model model) {
        return "echo/index";
    }

    @RequestMapping(value = "hello", method = RequestMethod.POST)
    public String hello(@Validated EchoForm form, BindingResult result, Model␣
↪model) { // (1)
        if (result.hasErrors()) { // (2)
            return "echo/index";
        }
        model.addAttribute("name", form.getName());
        return "echo/hello";
    }
}
```

Sr. No.	Description
(1)	In Controller, add `@Validated` annotation to the argument on which validation needs to be executed. Also add `BindingResult` object to the arguments. Input validation is automatically performed using Bean Validation and the result is stored in `BindingResult` object.
(2)	Error can be checked by executing `hasErrors` method. If there is an input error, it returns View name to display the input screen.

Add the implementation of displaying input error message on input screen (src/main/webapp/WEB-INF/views/echo/index.jsp).

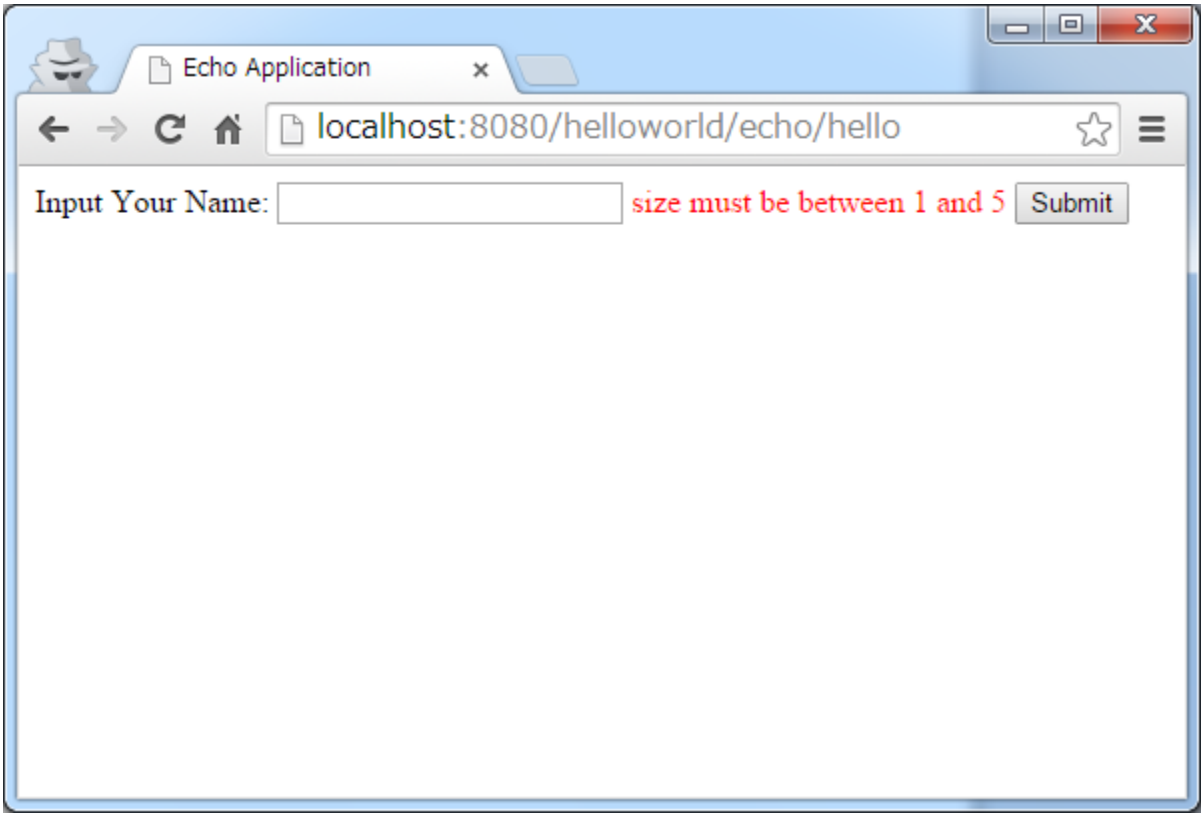
```
<!DOCTYPE html>
<html>
<head>
<title>Echo Application</title>
</head>
<body>
  <form:form modelAttribute="echoForm" action="${pageContext.request.contextPath}/
↪echo/hello">
    <form:label path="name">Input Your Name:</form:label>
    <form:input path="name" />
    <form:errors path="name" cssStyle="color:red" /><%-- (1) --%>
    <input type="submit" />
  </form:form>
</body>
</html>
```

Sr. No.	Description
(1)	Add `form:errors` tag for displaying error message when an error occurs on input screen.

Implementation of input validation is completed.

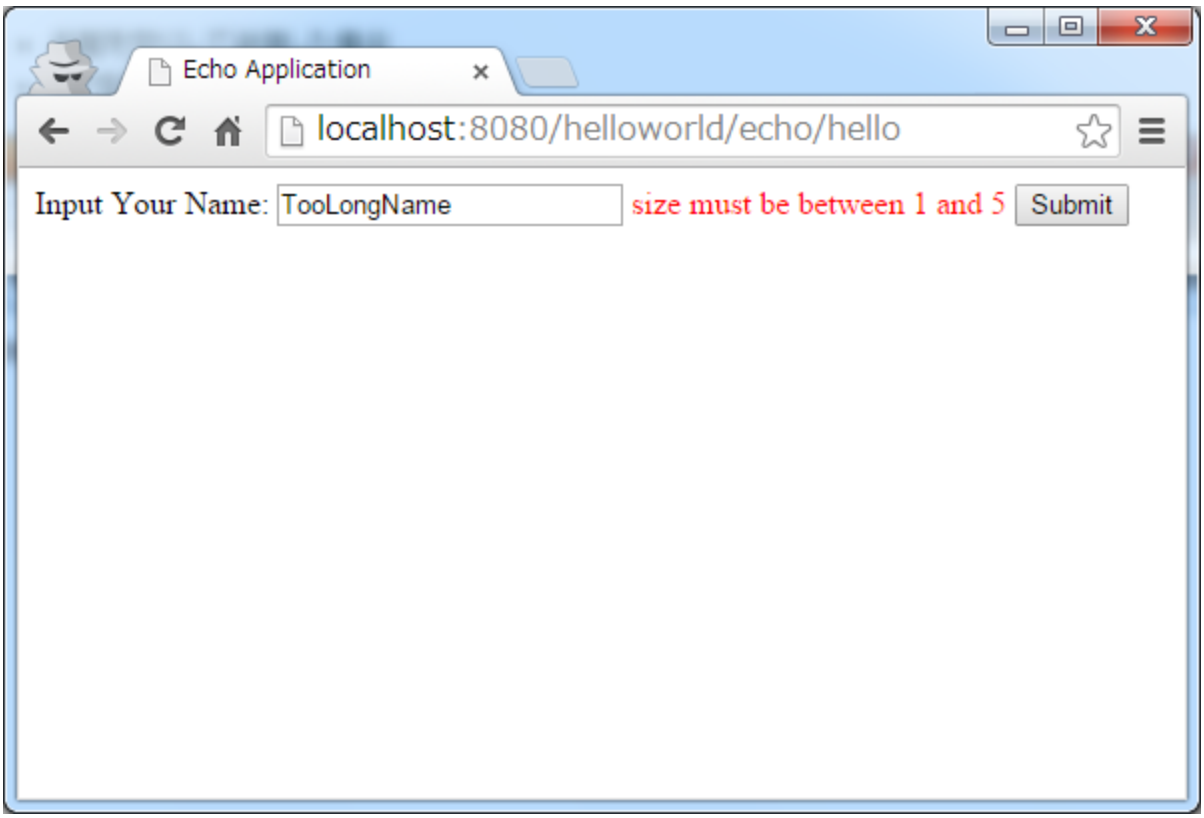
Error message is displayed in the following conditions:

- When an empty name is sent
- Size is more than 5 characters.



The generated HTML is as follows.

```
<!DOCTYPE html>
<html>
<head>
<title>Echo Application</title>
</head>
<body>
  <form id="echoForm" action="/helloworld/echo/hello" method="post">
    <label for="name">Input Your Name:</label>
    <input id="name" name="name" type="text" value=""/>
    <span id="name.errors" style="color:red">size must be between 1 and 5</span>
    <input type="submit" />
  <input type="hidden" name="_csrf" value="6e94a78d-4a2c-4a41-a514-0a60f0dbedaf" />
</form>
</body>
</html>
```



Summary

The following are the learnings from this chapter.

1. How to create blank project using `mvn archetype:generate`.
2. Basic Spring MVC configuration set-up
3. Simplified level screen transition
4. Way to pass values between screens
5. Simple input validation

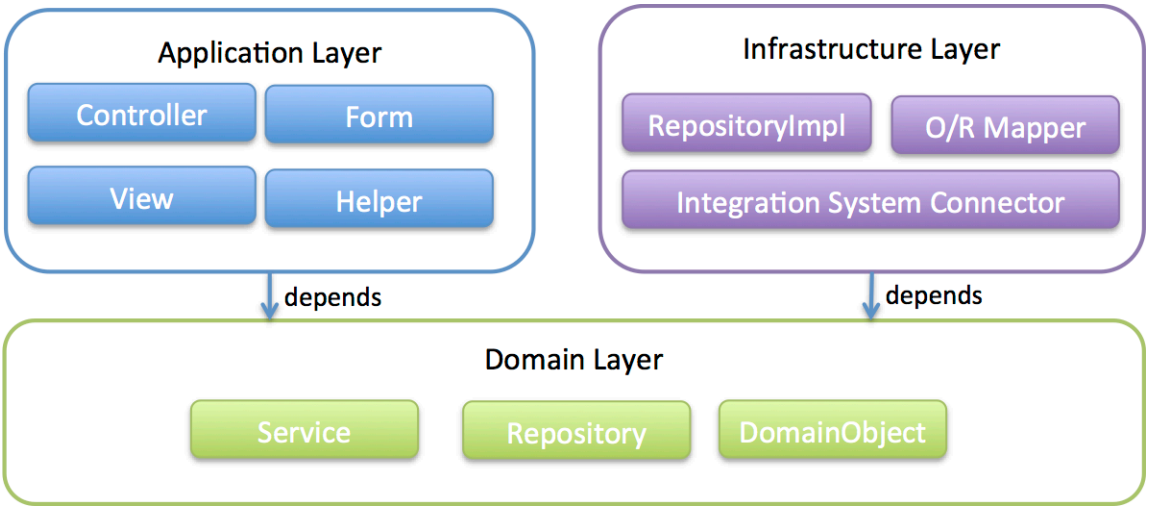
If above points are still not understood, it is recommended to read this chapter again and start again from building the environment.

2.4 Application Layering

In this guideline, the application is classified into the following 3 layers.

- Application layer
- Domain layer
- Infrastructure layer

Each layer has the following components.



Both application layer and infrastructure layer depend on domain layer, however **domain layer should not depend on other layers.** There can be changes in the application layer with the changes in domain layer, However, there should be no changes in domain layer with the changes in application layer.

Each layer is explained.

Note: Application layer, domain layer, infrastructure layer are the terms explained in "Domain-Driven Design (2004, Addison-Wesley)" of Eric Evans. However, though the terms are used, they are not necessarily as per the concept used in Domain Driven Design.

2.4.1 Layer definition

Flow of data from input to output is Application layer → Domain layer → Infrastructure layer, hence explanation is given in this sequence.

Application layer

Application layer does the wiring part of the application.

In this layer, it provides the following implementations:

- Provides UI(User Interface) to input and output information.
- Handles the request from clients.
- Validates the input data from a clients
- Calls components in the domain layer corresponding to the request from clients.

This layer should be as thin as possible and should not contain any business rules.

Controller

Controller is mainly responsible for the below:

- Controls the screen flow (mapping the request and returning the View corresponding to the result)
- Calls services in the domain layer (executing main logic corresponding to the request)

In Spring MVC, POJO class having `@Controller` annotation becomes the Controller class.

Note: When storing the client input data in the session, controllers also have a role to control the life cycle of data to be stored in the session.

View

View's responsibilities is generating output (including the UI provision) to the client. Returns output results in various formats such as HTML/PDF/Excel/JSON.

In Spring MVC, all this is done by `View` class.

Tip: If use the JSON or XML format output results for the REST API or Ajax request, `HttpMessageConverter` class plays View's responsibility.

Details refer to "*RESTful Web Service*".

Form

Form's main responsibilities are the following:

- Represents the form of HTML. (sending form information to the Controller, or outputting the processing result data to form of HTML)
- Declaration of input check rules. (by adding Bean Validation annotation)

In Spring MVC, form object are the POJO class that store request parameters. It is called as form backing bean.

Note: To retain the domain layer independent from the application layer, need to perform following processing at the application layer.

- Conversion from Form to DomainObject(such as Entity)
- Conversion from DomainObject to Form

If conversion code is too many, recommend to perform of either or both of the following measures and keep Controller's source code in simple state.

- Create helper class and delegate conversion logic to helper classes.
- Use the *Dozer*.

Tip: If use the JSON or XML as input data for REST API or Ajax request, `Resource` class plays Form's responsibility. Also, `HttpMessageConverter` class plays responsibility to convert JSON or XML input data into `Resource` class.

Details refer to "*RESTful Web Service*".

Helper

It plays the role of assisting the Controller.

Creating Helper is an option. Please create as POJO class if required.

Note: The main duty of Controller is routing (URL mapping and sepcifying the destination). If there is any processing required (converting JavaBean etc), then that part must be cut-off from controller and must be delegated

to helper classes.

The existance of Helper class improves the visibility of Controller; hence, it is OK to think as a part of Controller. (Similar to private methods of Controller)

Domain layer

Domain layer is the core of the application, and execute business rules.

In this layer, it provides the following implementations.

- DomainObject
- Checking business rules corresponding to the DomainObject (like whether there is sufficient balance when crediting amount into the account)
- Executing business rules for the DomainObject (reflects values corresponding to business rules)
- Executing CRUD operations for the DomainObject

Domain layer is independent from other layers and is reusable.

DomainObject

DomainObject is a model that represents resources required for business and items generated in the business process.

Models are broadly classified into following 3 categories.

- Resource models such as Employee, Customer, Product etc. (generally indicated by a noun)
- Event models such as Order, Payment etc.(generally indicated by a verb)
- Summary model such as YearlySales, MonthlySales etc.

Domain Object is the Entity that represents an object which indicates 1 record of database table.

Note: Mainly Models holding state only are handled in this guideline.

In "Patterns of Enterprise Application Architecture (2002, Addison-Wesley)" of Martin Fowler, Domain Model is defined as Item having state and behavior. We will not be touching such models in detail.

In this guideline, Rich domain model proposed by Eric Evans is also not included. However, it is mentioned here for classification.

Repository

It can be thought of as a collection of DomainObjects and is responsible for CRUD operations such as create, read, update and delete of DomainObject.

In this layer, only interface is defined.

It is implemented by RepositoryImpl of infrastructure layer. Hence, in this layer, there is no information about how data access is implemented.

Service

Provides the business logic.

In this guideline, it is recommended to draw the transaction boundary at the method of Service.

Note: Information related to Web such as Form and HttpRequest should not be handled in service.

This information should be converted into domain object in Application layer before calling the Service.

Infrastructure layer

Implementation of domain layer (Repository interface) is provided in infrastructure layer.

It is responsible for storing the data permanently (location where data is stored such as RDBMS, NoSQL etc.) as well as transmission of messages.

RepositoryImpl

Represents implementation of Repository interface of domain layer. It covers life cycle management of Domain-Object.

With the help of this structure, it is possible to hide implementation details from domain layer.

This layer also can be the transaction boundary depending on the requirements.

Tip: When using MyBatis3 or Spring Data JPA, most RepositoryImpls are created automatically.

O/R Mapper

It is responsible for mapping database with Entity.

This function is provided by MyBatis, JPA and Spring JDBC.

Specifically, the following classes are the O/R Mapper.

- Mapper interface or `SqlSession` in case of using MyBatis3
- `EntityManager` in case of using JPA,
- `JdbcTemplate` in case of using plain Spring JDBC

O/R Mapper used for implementation of Repository.

Note: It is more correct to classify MyBatis and Spring JDBC as "SQL Mapper" and not "O/R Mapper"; however, in this guideline it is treated as "O/R Mapper".

Integration System Connector

It integrates a data store other than the database(such as messaging system, Key-Value-Store, Web service, existing legacy system, external system etc.).

It is also used for implementation of Repository.

2.4.2 Dependency between layers

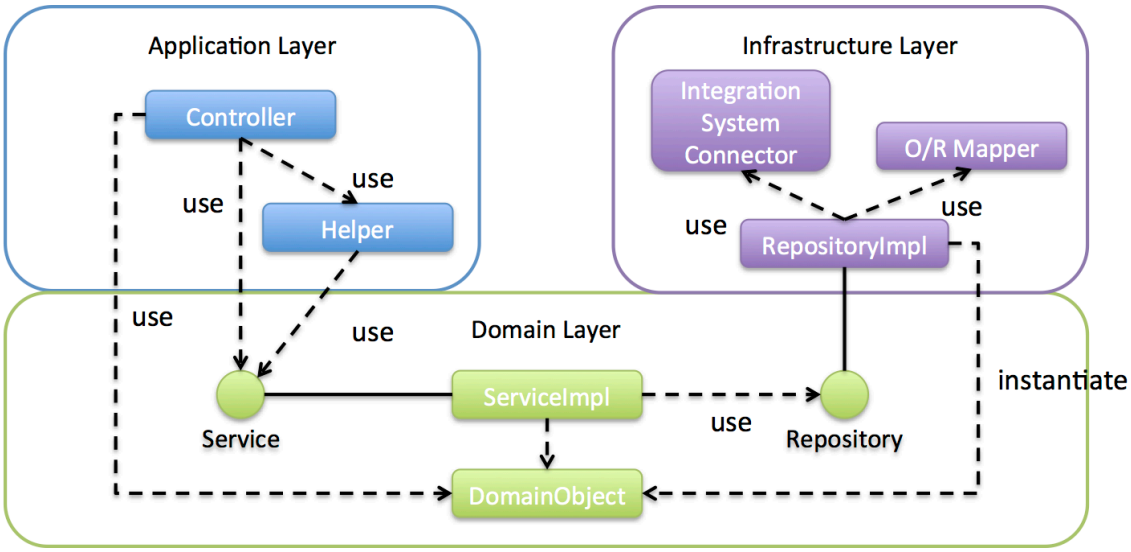
As explained earlier, domain layer is the core of the application, and application layer and infrastructure layer are dependent on it.

In this guideline, it is assumed that,

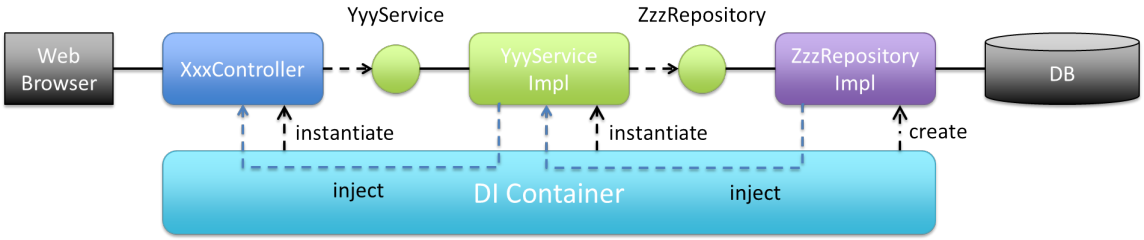
- Spring MVC is used in application layer
- MyBatis and Spring Data JPA are used in infrastructure layer

Even if there is change in implementation technology, the differences can be absorbed in respective layers and there should not be any impact on domain layer. Coupling between layers is done by using interfaces and hence they can be made independent of implementation technology being used in each layer.

It is recommended to make loosely-coupled design by understanding the layering.

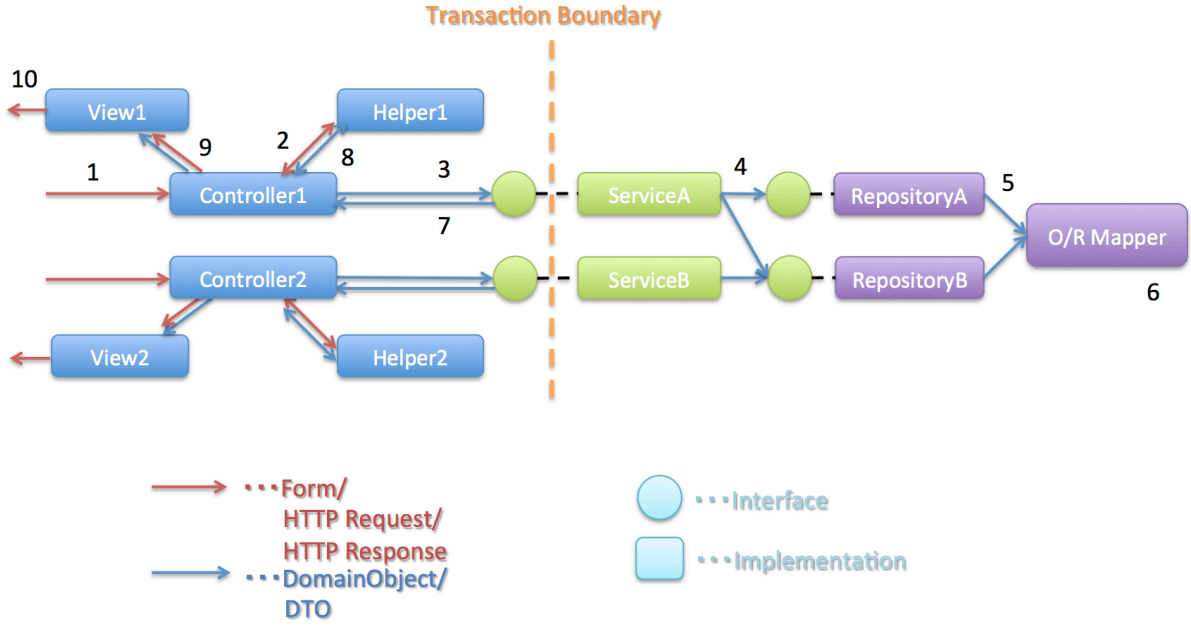


Object dependency in each layer can be resolved by DI container.



Processing and data flow with Repository

The flow from input to output is given in the following figure.



Sequence is explained using the example of update operation.

No.	Description
1.	Controller receives the Request.
2.	(Optional) Controller calls Helper and converts the Form information to DomainObject or DTO.
3.	Controller calls the Service by using DomainObject or DTO.
4.	Service calls the Repository and executes business logic.
5.	Repository calls the O/R Mapper and persists DomainObject or DTO.
6.	(Implementation dependency) O/R Mapper stores DomainObject or DTO information in DB.
7.	Service returns DomainObject or DTO which is the result of business logic execution to Controller.
8.	(Optional) Controller calls the Helper and converts DomainObject or DTO to Form.
9.	Controller returns View name of transition destination.
10.	View outputs Response.

Please refer to the below table to determine whether it is OK to call a component from another component.

Table.2.3 **Possibility of calling between components**

Caller/Callee	Controller	Service	Repository	O/R Mapper
Controller	✗	✓	✗	✗
Service	✗	⚠	✓	✗
Repository	✗	✗	✗	✓

Note that **calling a Service from another Service is basically prohibited**.

If services that can be used even from other services are required, SharedService should be created in order to

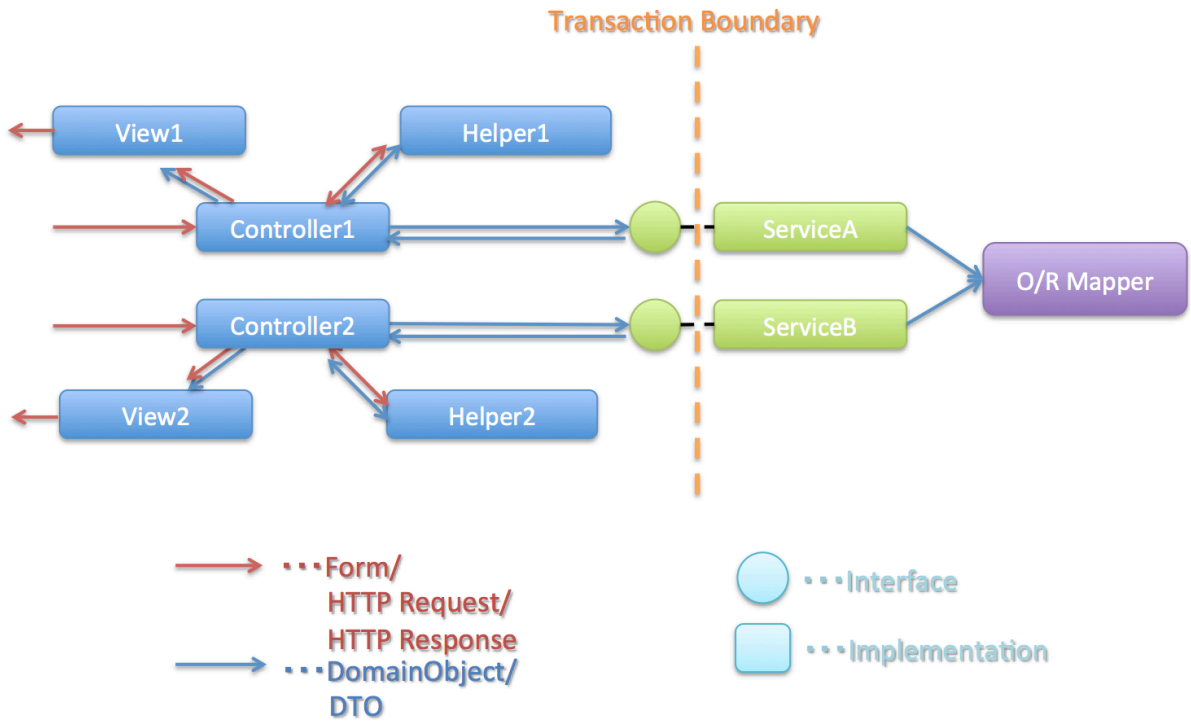
clarify such a possibility. Refer to *Domain Layer Implementation* for the details.

Note: It may be difficult to follow the above rules at the initial phase of application development. If looking at a very small application, it can be created quickly by directly calling the Repository from Controller. However, when rules are not followed, there will be many maintainability issues when development scope becoms larger. It may be difficult to understand the impact of modifications and to add common logic which is cross-cutting in nature. It is strongly recommended to pay attention to dependency from the beginning of development so that there will be no problem later on.

Processing and data flow without Repository

Hiding the implementation details and sharing of data access logic are the merits of creating a Repository.

However, it is difficult to share data access logic depending on the project team structure (multiple companies may separately implement business processes and control on sharing may be difficult etc.) In such cases, if abstraction of data access is not required, O/R Mapper can be called directly from Service as shown in the following diagram, without creating the repository.



Inter-dependency between components in this case must be as shown below:

Table.2.4 **Inter-dependency between components (without Repository)**

Caller/Callee	Controller	Service	O/R Mapper
Controller			
Service			

2.4.3 Project structure

Here, recommended project structure is explained when application layering is done as described earlier.

The standard maven directory structure is pre-requisite.

Basically, it is recommended to create the multiple projects with the following configuration.

Project Name	Description
[projectName]-domain	Project for storing classes/configuration files related to domain layer
[projectName]-web	Project for storing classes/configuration files related to application layer
[projectName]-env	Project for storing files dependent on environment

([projectName] is the target project name)

Note: Classes of infrastructure layer such as RepositoryImpl are also included in project-domain. Originally, [projectName]-infra project should be created separately; however, normally there is no need to conceal the implementation details in the infra project and development becomes easy if implementation is also stored in domain project. Moreover, when required, [projectName]-infra project can be created.

Tip: For the example of multi-project structure, refer to Sample Application or Test Application of Common

Library.

[projectName]-domain

Recommended structure of [projectName]-domain project is as below:

```
[projectName]-domain
  └src
      └main
          ├java
          |  └com
          |      └example
          |          └domain ...(1)
          |              ├model ...(2)
          |              |  ├Xxx.java
          |              |  ├Yyy.java
          |              |  └Zzz.java
          |              ├repository ...(3)
          |              |  ├xxx
          |              |  |  └XxxRepository.java
          |              |  ├yyy
          |              |  |  └YyyRepository.java
          |              |  └zzz
          |              |      ├ZzzRepository.java
          |              |      └ZzzRepositoryImpl.java
          |              └service ...(4)
          |                  ├aaa
          |                  |  ├AaaService.java
          |                  |  └AaaServiceImpl.java
          |                  └bbb
          |                      ├BbbService.java
          |                      └BbbServiceImpl.java
          └resources
              └META-INF
                  └spring
                      ├[projectName]-codelist.xml ...(5)
                      ├[projectName]-domain.xml ...(6)
                      └[projectName]-infra.xml ...(7)
```

No.	Details
(1)	Package to store configuration elements of domain layer.
(2)	Package to store DomainObjects classes.
(3)	Package to store Repository interfaces. Create a separate package for each Entity. If there are associated Entities to the main entity, then Repository interfaces of associated Entities must also be placed in the same package as main Entity. (For example, Order and OrderLine). If DTO(holds such as search criteria) is also required, it too must be placed in this package. RepositoryImpl belongs to Infrastructure layer; however, there is no problem in keeping it in this project. In case of using different data stores or existance of multiple persistence platforms, RepositoryImpl class must be kept in separate project or separate package so that implementation related details are concealed.
(4)	Package to store Service classes. Package must be created based on Entity Model or other functional unit. Interface and Implementation class must be kept at the same level of package. If input/output classes are also required, then they must be placed in this package.
(5)	Bean definition for CodeList.
(6)	Bean definition pertaining to domain layer.
(7)	Bean definition pertaining to infrastructure layer.

[projectName]-web

Recommended structure of [projectName]-web

```
[projectName]-web
  └src
      └main
```

```
        ├ java
        |   └ com
        |       └ example
        |           └ app ...(1)
        |               ├ abc
        |               |   ├ AbcController.java
        |               |   ├ AbcForm.java
        |               |   └ AbcHelper.java
        |               └ def
        |                   ├ DefController.java
        |                   ├ DefForm.java
        |                   └ DefOutput.java
        ├ resources
        |   ├ META-INF
        |   |   └ spring
        |   |       ├ applicationContext.xml ...(2)
        |   |       ├ application.properties ...(3)
        |   |       ├ spring-mvc.xml ...(4)
        |   |       └ spring-security.xml ...(5)
        |   └ i18n
        |       └ application-messages.properties ...(6)
        └ webapp
            ├ resources ...(7)
            └ WEB-INF
                ├ views ...(8)
                |   ├ abc
                |   |   ├ list.jsp
                |   |   └ createForm.jsp
                |   └ def
                |       ├ list.jsp
                |       └ createForm.jsp
                └ web.xml ...(9)
```

No,	Details
(1)	Package to store configuration elements of application layer.
(2)	Bean definited related to the entire application.
(3)	Define the properties to be used in the application.
(4)	Bean definitions related to Spring MVC.
(5)	Bean definitions related to Spring Security
(6)	Define the messages and other contents to be used for screen display (internationalization).
(7)	Store static resources(css、js、image, etc)
(8)	Store View(jsp) files.
(9)	Servlet deployment definitions

[projectName]-env

The recommended structure of [projectName]-env project is below:

```
[projectName]-env
  ├configs ...(1)
```

```
|   └[envName] ...(2)
|       └resources ...(3)
└src
    └main
        └resources ...(4)
            ├META-INF
            |   └spring
            |       ├[projectName]-env.xml ...(5)
            |       └[projectName]-infra.properties ...(6)
            ├dozer.properties
            ├log4jdbc.properties
            └logback.xml ...(7)
```

No.	Details
(1)	Directory to define configurations depends on the environment for all environments.
(2)	Directory to define configurations depeands on each environment. The directory name is used as the name to identify the environment.
(3)	Directory to define configurations depeands on each environment. The sub directory structure and files are same as (4).
(4)	Directory to define configurations depeands on the local development environment.
(5)	Bean definitions that depend on the local development environment (like dataSource etc).
(6)	Property definitions which depend on the local development environment.
(7)	Log output definitions which depend on the local development environment.

Note: The purpose of separating [projectName]-domain and [projectName]-web into different projects is to prevent reverse dependency among them.

It is natural that [projectName]-web uses [projectName]-domain; however, [projectName]-domain must not refer projectname]-web.

If configruation elements of both, [projectName]-web and [projectName]-domain, are kept in a single project, it may lead to an incorrect reference by mistake. It is strongly recommended to prevent reference to [projectName]-web from [projectName]-domain by separating the project and setting order of reference.

Note: The reason for creating [projectName]-env is to separate the information depending on the environment and thereby enable switching of environment.

For example, set local development environment by default and at the time of building the application, create war file without including [projectName]-env. By creating a separate jar for integration test environment or system test environment, deployment becomes possible just by replacing the jar of corresponding environment.

Further, it is also possible to minimize the changes in case of project where RDBMS being used changes.

If the above point is not considered, contents of configuration files have to changed according to the target environment and the project has to be re-build.

3

Application Development

Description of various rules as well as recommended implementation on the use of TERASOLUNA Server Framework for Java (5.x).

The flow of development in this guideline with be as follows.

3.1 Create Web application development project

In this section, how to create a web application development project is described.

In this guideline, it is recommended to adopt multi-project configuration. For description of the recommended multi-project configuration, refer [*Project structure*].

3.1.1 Types of Blank Project

Following two types of blank projects are provided depending on usage.

Type	Usage
Blank project of multi-project structure	This should be used when developing a real application that is to be released in commercial environment. Following types of project templates are provided as Archetype of Maven. • Template that includes settings for MyBatis3 • Template that includes settings for JPA (Spring Data JPA) **This guideline recommends using a project having multi-project structure.**
Blank project of single-project structure	This should be used when creating simple applications such as POC (Proof Of Concept), prototype, sample, etc. Following types of project templates are provided as Archetype of Maven. (Projects for Eclipse WTP are also provided; however, their description is omitted in this chapter) • Template that includes settings for MyBatis3 • Template that includes settings for JPA (Spring Data JPA) • Template that does not depend on O/R Mapper This guideline follows a procedure wherein various tutorials are performed using a single project.

3.1.2 Create development project

The multi-project structured development project will be created using the archetype:generate of the Maven Archetype Plugin .

Note: Prerequisite

In the following description,

- Maven (`mvn` command) is used
- Internet connection is used
- If internet is connected via proxy, Maven proxy setting needs to be done

are prerequisites.

If prerequisite conditions are not satisfied, it is necessary to perform these setups first.

As an Archetype following two types are provided for creating multi-project.

Sr. No.	Archetype(ArtifactId)	Description
1.	terasoluna-gfw-multi-web-blank-mybatis3-archetype	Archetype for generating a project to use MyBatis3 as O/R Mapper.
2.	terasoluna-gfw-multi-web-blank-jpa-archetype	Archetype for generating a project to use JPA(with Spring Data JPA and Hibernate) as O/R Mapper.

Move to the folder where you want to create project.

```
cd C:\work
```

Create project using archetype:generate of Maven Archetype Plugin.

```
mvn archetype:generate -B^
 -DarchetypeGroupId=org.terasoluna.gfw.blank^
 -DarchetypeArtifactId=terasoluna-gfw-multi-web-blank-mybatis3-archetype^
 -DarchetypeVersion=5.3.0.RELEASE^
 -DgroupId=com.example.todo^
 -DartifactId=todo^
 -Dversion=1.0.0-SNAPSHOT
```

Parameter	Description
-B	batch mode (skips interaction)
-DarchetypeGroupId	Specify groupId of the blank project.(Fixed)
-DarchetypeArtifactId	Specify archetypeId (ID to identify the template) of the blank project. **(Customization required)** specify one of the following archetypeId. • `terasoluna-gfw-multi-web-blank-mybatis3-archetype` • `terasoluna-gfw-multi-web-blank-jpa-archetype` In above example, `terasoluna-gfw-multi-web-blank-mybatis3-archetype` is specified.
-DarchetypeVersion	Specify version of the blank project.(Fixed)
-DgroupId	Specify groupId of the project that you want to create. **(Customization required)** In above example, `"com.example.todo"` is specified.
-DartifactId	Specify artifactId of the project that you want to create. **(Customization required)** In above example, `"todo"` is specified.
-Dversion	Specify version of the project that you want to create. **(Customization required)** In above example, `"1.0.0-SNAPSHOT"` is specified.

If the project creation successes, following type of log will be printed. (The following output is an example when project is created using the MyBatis3 Archetype)

```
(... omit)
[INFO] ----------------------------------------------------------------------------
[INFO] Using following parameters for creating project from Archetype: terasoluna-
↪gfw-multi-web-blank-mybatis3-archetype:5.3.0.RELEASE
[INFO] ----------------------------------------------------------------------------
[INFO] Parameter: groupId, Value: com.example.todo
[INFO] Parameter: artifactId, Value: todo
```

```
[INFO] Parameter: version, Value: 1.0.0-SNAPSHOT
[INFO] Parameter: package, Value: com.example.todo
[INFO] Parameter: packageInPathFormat, Value: com/example/todo
[INFO] Parameter: package, Value: com.example.todo
[INFO] Parameter: version, Value: 1.0.0-SNAPSHOT
[INFO] Parameter: groupId, Value: com.example.todo
[INFO] Parameter: artifactId, Value: todo
[INFO] Parent element not overwritten in C:\work\todo\todo-env\pom.xml
[INFO] Parent element not overwritten in C:\work\todo\todo-domain\pom.xml
[INFO] Parent element not overwritten in C:\work\todo\todo-web\pom.xml
[INFO] Parent element not overwritten in C:\work\todo\todo-initdb\pom.xml
[INFO] Parent element not overwritten in C:\work\todo\todo-selenium\pom.xml
[INFO] project created from Archetype in dir: C:\work\todo
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 9.929 s
[INFO] Finished at: 2015-07-31T12:03:21+00:00
[INFO] Final Memory: 10M/26M
[INFO] ------------------------------------------------------------------------
```

If the project creation successes, Maven multi-project gets created. For detail description of the project that you have created in the Maven Archetype, Refer [*Structure of the development project*].

```
todo
├- pom.xml
├- todo-domain
├- todo-env
├- todo-initdb
├- todo-selenium
└- todo-web
```

3.1.3 Customization of development project

Depending upon the application, there are several locations where customization is required in the Maven Archetype created project.

The customization required locations are described below.

- *POM file project information*
- *x.xx.fw.9999 format message ID*
- *Message wording*
- *Error screen*
- *Screen footer copyright*
- *In-memory database (H2 Database)*
- *DataSource configuration*

Note: The customization points other than the above are,

- Settings of *Authentication* · *Authorization*
- Settings to enable *File Upload*
- Setting to activate *Internationalization*
- Definition of *Logging*
- Definition of *Exception Handling*
- Apply settings of *RESTful Web Service*

For these customizations, Refer to "How to use" of each section and customize if required.

Note: Part that is expressed as `artifactId` in the following description needs to be read by replacing the `artifactId` which is specified at the time of creating a project.

POM file project information

In the POM file of Maven Archetype created project,

- Project name (`name` element)
- Project description (`description` element)
- Project URL (`url` element)

- Project inception year (`inceptionYear` element)
- Project license (`licenses` element)
- Project organization (`organization` element)

such information set in Archetype projects. The actual settings contents indicated below.

```
<!-- ... -->

<name>TERASOLUNA Server Framework for Java (5.x) Web Blank Multi Project</name>
<description>Web Blank Multi Project using TERASOLUNA Server Framework for Java (5.
↪x)</description>
<url>http://terasoluna.org</url>
<inceptionYear>2014</inceptionYear>
<licenses>
    <license>
        <name>Apache License, Version 2.0</name>
        <url>http://www.apache.org/licenses/LICENSE-2.0.txt</url>
        <distribution>manual</distribution>
    </license>
</licenses>
<organization>
    <name>TERASOLUNA Framework Team</name>
    <url>http://terasoluna.org</url>
</organization>

<!-- ... -->
```

Note: Set the appropriate values in the project information.

Customization method and customization targeted files are indicated below.

Sr. No.	Targeted File	Customization method
1.	POM (Project Object Model) file that defines the overall configuration of multi-project `artifactId/pom.xml`	Set the appropriate values in the project information.

x.xx.fw.9999 format message ID

In the Maven Archetype created project, the `x.xx.fw.9999` format message ID used at the time of,

- Message to be displayed on the error screen
- Error log to be output when an exception occurs

Actual point-of-use (sampling) indicated below.

[application-messages.properties]

```
e.xx.fw.5001 = Resource not found.
```

[JSP]

```
<div class="error">
    <c:if test="${!empty exceptionCode}">[${f:h(exceptionCode)}]</c:if>
    <spring:message code="e.xx.fw.5001" />
</div>
```

[applicationContext.xml]

```
<bean id="exceptionCodeResolver"
    class="org.terasoluna.gfw.common.exception.SimpleMappingExceptionCodeResolver">
    <!-- ... -->
            <entry key="ResourceNotFoundException" value="e.xx.fw.5001" />
    <!-- ... -->
</bean>
```

The `x.xx.fw.9999` format message ID is a message ID system that is introduced in [*Message Management*] of this guideline but, the value of the project division is in the state of provisional value [`xx`].

Note:

- **If the message ID system introduced in this guideline is used, specify the appropriate values to the project classification.** For the message ID system introduced in this guideline, Refer [*Result messages*].
- If the message ID system introduced in this guideline is not used, it is necessary to replace all the message IDs those are used in the customization targeted file indicated below.

Customization method and customization targeted files are indicated below.

Sr. No.	Targeted File	Customization method
1.	Message definition file `artifactId/artifactId-web/src/main/resources/i18n/application-messages.proper`	The provisional value [xx] of project classification message ID specified in the property key needs to be modified by appropriate value.
2.	Error screen JSP `artifactId/artifactId-web/src/main/webapp/WEB-INF/views/common/error/`	The provisional value [xx] of project classification message ID specified in the `code` attribute of the element `<spring:message>` needs to be modified by appropriate value.
3.	Bean definition file to create an application context for Web applications `artifactId/artifactId-web/src/main/resources/META-INF/spring/applicationConte`	The provisional value [xx] of project classification exception code (message ID) specified in the Bean definition of `ExceptionCodeResolver` needs to be modified by appropriate value.

Message wording

In the Maven Archetype created project, number of message definitions are provided but, message wordings are simple messages. Actual messages (sampling) are indicated below.

[application-messages.properties]

```
e.xx.fw.5001 = Resource not found.

# ...

# typemismatch
typeMismatch="{0}" is invalid.

# ...
```

Note: Modify the message wording depending upon the application requirements (such as message terms)

Customization method and customization targeted files are indicated below.

Sr. No.	Targeted File	Customization method
1.	Message definition file `artifactId/artifactId-web/src/main/resources/i18n/application-messages.proper`	Modify the messages in accordance with the application requirements. The message to be displayed (Bean Validation messages) when there is an error in input check needs to be modified (override default messages) depending upon the application requirement. For overriding the default messages, Refer [*Definition of error messages*].

Error screen

In the Maven Archetype created project, JSP and HTML are provided for displaying an error screen for every kind of errors but,

- screen layout
- screen title
- wording of the message

etc are simple implementation. Actual JSP implementation (sampling) is indicated below.

[JSP]

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>Resource Not Found Error!</title>
<link rel="stylesheet"
    href="${pageContext.request.contextPath}/resources/app/css/styles.css">
</head>
<body>
    <div id="wrapper">
        <h1>Resource Not Found Error!</h1>
        <div class="error">
            <c:if test="${!empty exceptionCode}">[${f:h(exceptionCode)}]</c:if>
            <spring:message code="e.xx.fw.5001" />
        </div>
        <t:messagesPanel />
    <br>
    <!-- ... -->
```

```
    <br>
    </div>
</body>
</html>
```

Note: Modify the JSP and HTML depending upon the application requirements (such as UI terms) used for displaying an error screen.

Customization method and customization targeted files are indicated below.

Sr. No.	Targeted File	Customization method
1.	JSP for the error screen `artifactId/artifactId-web/src/main/webapp/WEB-INF/views/common/error/*.jsp`	Modify depending upon the application requirements (such as UI terms). Refer [*Coding points (JSP)* of *Exception Handling*] for customizing the JSP to display an error screen.
2.	HTML for the error screen `artifactId/artifactId-web/src/main/webapp/WEB-INF/views/common/error/unhandle`	Modify depending upon the application requirements (such as UI terms).

Screen footer copyright

In the Maven Archetype created project, screen layouts are configured using Tiles but, the copyright of the screen footer portion is in a state of provisional value [`Copyright © 20XX CompanyName`]. Actual JSP implementation (sampling) is indicated below.

[template.jsp]

```
<div class="container">
  <tiles:insertAttribute name="header" />
  <tiles:insertAttribute name="body" />
  <hr>
  <p style="text-align: center; background: #e5eCf9;">Copyright
    &copy; 20XX CompanyName</p>
</div>
```

Note: If screen layouts are configured using Tiles, specify appropriate value to the copyright.

Customization method and customization targeted files are indicated below.

Sr. No.	Targeted File	Customization method
1.	Template JSP for Tiles `artifactId/artifactId-web/src/main/webapp/WEB-INF/views/layout/template.jsp`	Modify the provisional value [`Copyright © 2013 Company Name`] of the copyright with an appropriate value.

In-memory database (H2 Database)

In the Maven Archetype created project, in-memory database (H2 Database) setting is configured but, these settings are done for the small operation (Prototyping and POC (Proof Of Concept)) verification. Therefore, these could be unnecessary settings while having regular application development.

[artifactId-env.xml]

```
<jdbc:initialize-database data-source="dataSource"
    ignore-failures="ALL">
    <jdbc:script location="classpath:/database/${database}-schema.sql" encoding=
↪"UTF-8" />
    <jdbc:script location="classpath:/database/${database}-dataload.sql" encoding=
↪"UTF-8" />
</jdbc:initialize-database>
```

```
└- src
    └- main
        └- resources
            ├- META-INF
          (...)
            ├- database
            |  ├  - H2-dataload.sql
            |  └  - H2-schema.sql
```

Note: While having regular application development, remove the directory which is maintained for defini-

tion and SQL files for setting up a In-memory database (H2 Database)

Customization method and customization targeted files are indicated below.

Sr. No.	Targeted File	Customization method
1.	Bean definition file for defining environment dependent components `artifactId-env/src/main/resources/META-INF/spring/artifactId-env.xml`	Remove the `<jdbc:initialize-database>` element.
2.	Directory that contains the SQL for configuring In-memory database (H2 Database) `artifactId/artifactId-env/src/main/resources/database/`	Remove the directory.

DataSource configuration

In the Maven Archetype created project, DataSource setting is done for accessing in-memory database (H2 Database) but, these settings are done for the small operation (Prototyping and POC (Proof Of Concept)) verification. Therefore it is necessary to change the DataSource settings for accessing the actual running database application while having regular application development.

[artifactId/artifactId-domain/pom.xml]

```
<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <scope>runtime</scope>
</dependency>
```

Note: In the above setting example, since it is assumed that the dependent library version is managed by the parent project terasoluna-gfw-parent , specifying the version in pom.xml is not necessary. The above dependent library used by terasoluna-gfw-parent is defined by Spring IO Platform.

[artifactId-infra.properties]

```
database=H2
database.url=jdbc:h2:mem:todo;DB_CLOSE_DELAY=-1
```

```
database.username=sa
database.password=
database.driverClassName=org.h2.Driver
# connection pool
cp.maxActive=96
cp.maxIdle=16
cp.minIdle=0
cp.maxWait=60000
```

[artifactId-env.xml]

```
<bean id="realDataSource" class="org.apache.commons.dbcp2.BasicDataSource"
    destroy-method="close">
    <property name="driverClassName" value="${database.driverClassName}" />
    <property name="url" value="${database.url}" />
    <property name="username" value="${database.username}" />
    <property name="password" value="${database.password}" />
    <property name="defaultAutoCommit" value="false" />
    <property name="maxTotal" value="${cp.maxActive}" />
    <property name="maxIdle" value="${cp.maxIdle}" />
    <property name="minIdle" value="${cp.minIdle}" />
    <property name="maxWaitMillis" value="${cp.maxWait}" />
</bean>
```

Note: Change the DataSource settings for accessing the actual running database application while having regular application development.

In the Maven Archetype created project, the use of Apache Commons DBCP is configured but, there are many cases that adopting a method of accessing a DataSource via JNDI (Java Naming and Directory Interface) by use of DataSource provided by the application server.

Again there are some cases where Apache Commons DBCP is used on development environment and DataSource provided by the application server is used on test as well as production environment.

For how to set-up the DataSource, Refer [*Datasource settings* of *Database Access (Common)*].

Customization method and customization targeted files are indicated below.

Sr. No.	Targeted File	Customization method
1.	POM file • `artifactId/pom.xml` • `artifactId/artifactId-web/pom.xml`	Remove in-memory database (H2 Database) JDBC driver from the dependency library. Add the JDBC driver in dependency library for accessing the actual running application database.
2.	Property file for defining environment dependent setting `artifactId/artifactId-env/src/main/resources/META-INF/spring/artifactId-infra.properties`	If Apache Commons DBCP is used as a DataSource, specify the connection information for accessing the actual running application database in below property. • `database` • `database.url` • `database.username` • `database.password` • `database.driverClassName` Remove unnecessary property except the following property if DataSource provided by the application server is used. • `database`
3.	Bean definition file for defining environment dependent components `artifactId/artifactId-env/src/main/resources/META-INF/spring/artifactId-env.xml`	If DataSource provided by the application server is used, change the configuration to use the DataSource that is obtained via JNDI. For how to set-up the DataSource, Refer [*Datasource settings* of *Database Access (Common)*].

Note: About the database property of the property file for defining environment dependent setting

The `database` property is unnecessary property if MyBatis is used as O/R Mapper. You may remove this but you may leave the settings in order to specify the database being used.

Tip: How to add the JDBC driver

It is fine to remove the comment out of POM file in case of PostgreSQL or Oracle database is used.. Modify the JDBC driver version by actual use of the corresponding database version.

However, if Oracle is used, it is necessary to install the Oracle JDBC driver in the local repository of Maven before removing the comment.

The following is an example of setting in case of PostgreSQL is used.

- `artifactId/pom.xml`

```
                <dependency>
                    <groupId>org.postgresql</groupId>
                    <artifactId>postgresql</artifactId>
                    <version>${postgresql.version}</version>
                </dependency>
<!--            <dependency> -->
<!--                <groupId>com.oracle</groupId> -->
<!--                <artifactId>ojdbc7</artifactId> -->
<!--                <version>${ojdbc.version}</version> -->
<!--            </dependency> -->

    <!-- ... -->

    <postgresql.version>9.4-1206-jdbc41</postgresql.version>
    <ojdbc.version>12.1.0.2</ojdbc.version>
```

- `artifactId/artifactId-web/pom.xml`

```
            <dependency>
                <groupId>org.postgresql</groupId>
                <artifactId>postgresql</artifactId>
                <scope>runtime</scope><!-- (1) -->
            </dependency>
<!--        <dependency> -->
<!--            <groupId>com.oracle</groupId> -->
<!--            <artifactId>ojdbc7</artifactId> -->
<!--            <scope>runtime</scope> -->
<!--        </dependency> -->
```

Sr. No.	Description
(1)	JDBC driver is not need when compile, it is only need when run time, so the `runtime` scope is specified. In case of using JDBC driver in unit test, change to appropriate scope and use it.

3.1.4 Structure of the development project

Explained the structure of the project created in Maven Archetype.

Below is the structure of the project created in Maven Archetype.

- Project structure of each layer that is recommended in this guideline
- Project structure that takes into account the exclusion of environmental dependency introduced in this guideline
- Project structure that conscious the CI (Continuous Integration)

In addition, various settings have been included that is recommended in this guideline

- Web application configuration file (web.xml)
- Spring Framework Bean definition file
- Bean definition file for the Spring MVC
- Bean definition file for the Spring Security
- O/R Mapper configuration file
- Tiles configuration file
- Property file (such as message definition file)

and, as a simple component implementation of low (=necessary to develop every kind of application) dependency on the application requirements,

- Controller and JSP for displaying Welcome page
- JSP to display an error screen (HTML)
- Template JSP for Tiles
- Include JSP for reading configuration such as JSP tag library
- CSS file that defines the screen style of entire application

etc are provided.

Warning: Components provided as a simple implementation

Components provided as a simple implementation can be corresponding to one of the following.

- Modification to meet the application requirements
- Removal of unnecessary components

Note: Procedure to create the REST API project

In the Maven Archetype created project, the recommended settings are done which are required for building a traditional Web application (application that receives the request parameters and respond the HTML).

Therefore, unnecessary setting exists in building a REST API for handling JSON or XML. If you want to create a project for building REST API, need to apply the REST API related settings by referring to the [*Application settings* of *RESTful Web Service*].

Note: Part that is expressed as `artifactId` in the following description needs to be read by replacing the `artifactId` which is specified at the time of creating a project.

Multi-project structure

Initially entire multi-project structure is explained.

```
artifactId
    ├- pom.xml  ... (1)
    ├- artifactId-web  ... (2)
    ├- artifactId-domain  ... (3)
    ├- artifactId-env  ... (4)
    ├- artifactId-initdb  ... (5)
    └- artifactId-selenium  ... (6)
```

Sr. No.	Description
(1)	The entire multi-project configuration is defined in POM (Project Object Model) file. Mainly following definitions are done in this file. • Version of the dependent libraries • Build plug-ins settings (setting of how to build) Refer [*Hierarchical structure of the project*] for the hierarchical relationship of multi-project.

次のページに続く

Table. 3.1 – 前のページからの続き

Sr. No.	Description
(2)	Module that manages the application layer (Web layer) components. Mainly following components and files are managed in this module. • Controller class • Validator class for relational check • Form class (the Resource class in case of REST API) • View (JSP) • CSS file • JavaScript file • JUnit for the application layer components • Bean definition file for defining the application layer components • Web application configuration file (web.xml) • Message definition file
(3)	Module that manages the domain layer components. Mainly following components and files are managed in this module. • Domain object such as Entity • Repository • Service • DTO • JUnit for the domain layer components • Bean definition file for defining the domain layer components
(4)	Module that manages the environmental dependency settings files. Mainly following files are managed in this module. • Bean definition file for defining the environment dependent components • Property file for defining the environment dependent properties value
(5)	Module that manages the database initialization SQL files. Mainly following files are managed in this module. • SQL file to create the database objects such as tables • SQL file to insert the initial data such as master data • SQL file to insert the test data used for E2E (End To End) test

次のページに続く

Table. 3.1 – 前のページからの続き

Sr. No.	Description
(6)	Module that manages the Selenium used E2E testing components Mainly following files are managed in this module. • JUnit testing using Selenium operation • Expected value file used while Assert (if necessary)

Note: About a terminology definition of [multi-project] in this guideline

The project created in Maven Archetype is the exact multi-module structured project.

This is supplement that the multi-module and multi-project is being used as the same meaning in this guideline.

Note: Development projects required for two Web applications and one common library

- bar-parent
- bar-initdb
- bar-common
- bar-common-web
- bar-domain-a
- bar-domain-b
- bar-web-a
- bar-web-b
- bar-env
- bar-web-a-selenium
- bar-web-b-selenium

The contents of each project are as follows.

- bar-parent

  Project called as a parent-pom (parent POM). A simple project consisting of only pom.xml file. It never contains other source code or configuration files. Common setting information specified in the parent POM can be reflected in other project by specifying this bar-parent project into <parent> tag.

- bar-initdb

  Stores RDBMS table definitions (DDL) and SQL statements for INSERT the initial data. This also managed as a maven project. By defining sql-maven-plugin in pom.xml, it is possible to automate the execution of DDL statements and initial data INSERT statements for any RDBMS in the course of the build lifecycle.

- bar-common

  Stores common library in the project. Web related classes are placed in the bar-common-web by making it as a web-independent.

- bar-common-web

  Stores common web library in the project.

- bar-domain-a

  Stores unit test cases and domain layer java classes related to “a” domain. Finally *.jar file is created.

- bar-domain-b

  Stores domain layer java classes related to “b” domain.

- bar-web-a

  Stores application layer java classes, jsps, configuration files, unit test cases. Finally created *.war file is created as the Web application. bar-web-a having dependency on bar-common and bar-env.

- bar-web-b

  This is a Web application as one more subsystem. Structure is the same as the bar-web-a.

- bar-env

  Collects only the configuration files having environment dependency.

- bar-web-a-selenium

  Project that stores test cases using Selenium WebDriver for web-a project.

- bar-web-b-selenium

  Project that stores test cases using Selenium WebDriver for web-b project.

Structure of Web module

Module that manages the application layer (Web layer) components are explained.

```
artifactId-web
    ├- pom.xml  ... (1)
```

Sr. No.	Description
(1)	The web module configuration is defined in POM (Project Object Model) file. Following definitions are done in this file. • Definition of dependent libraries and build plug-ins • Definition to create a war file

Note: About the module name of the web module while creating a project for REST API

The application type can be easily distinguished, if the module name is assigned the name of `artifactId-api` while building a REST API.

```
└- src
    ├- main
    |    ├- java
    |    |   └  - com
    |    |         └- example
    |    |             └- project
    |    |                 └- app  ... (2)
    |    |                     └- welcome
    |    |                         └- HelloController.java  ... (3)
    |   ├  - resources
    |    |   ├  - META-INF
    |    |    |   ├  - dozer  ... (4)
    |    |    |   └  - spring  ... (5)
    |    |    |        ├- application.properties  ... (6)
    |    |    |        ├- applicationContext.xml  ... (7)
    |    |    |        ├- spring-mvc.xml  ... (8)
    |    |    |        └- spring-security.xml  ... (9)
    |    |   └  - i18n  ... (10)
    |    |        └- application-messages.properties  ... (11)
```

Sr. No.	Description
(2)	Package for storing the application layer classes. The component type can be easily distinguished, if the package name is assigned the name of `api` while building a REST API.
(3)	The controller class for receiving a request to display the Welcome page.
(4)	The directory in which a mapping definition file of Dozer (Bean Mapper) is stored, Refer to [*Bean Mapping (Dozer)*] for Dozer. It is an empty directory at the time of creation. If the mapping file is required (if high mapping is required), it gets automatically read in case of stored under this directory. **Note:** Following files are stored under this directory. • Definition file for mapping the application layer JavaBean with domain layer JavaBean • Definition file for each other mapping of application layer JavaBean It is recommended to store each other mapping of domain layer JavaBean in domain layer directory.
(5)	Directory contains the property file and Spring Framework bean definition file.
(6)	Properties file that defines the settings to be used in the application layer. It is an empty file at the time of creation.
(7)	Bean definition file to create an application context for Web applications. Following beans are defined in this file. • Components to be used in the entire Web application • Domain layer components (Import the bean definition file in which domain layer components are defined)
(8)	Bean definition file to create an application context for the `DispatcherServlet`. Following beans are defined in this file. • Spring MVC components • application layer components The application type can be easily distinguished, if the file name is assigned the name of `spring-mvc-api.xml` while building a REST API.

次のページに続く

Table. 3.2 – 前のページからの続き

Sr. No.	Description
(9)	Bean definition file for defining the Spring Security components. This file is read when you create an application context for the Web application.
(10)	Directory that contains the message definition file to be used in the application layer.
(11)	Property file that defines the messages to be used in the application layer. Some of the generic messages are defined at the time of creation. **Note: Messages should be modified according to the application requirements (Such as message Terms).** For the message definition, Refer [*Message Management*].

Note: Refer [*Relationship of bean definition file and application context structure*] for the application context and bean definition file related.

```
|   └  - webapp
|         ├- WEB-INF
|         |  ├  - tiles  ... (12)
|         |    |  └  - tiles-definitions.xml
|         |  ├  - views  ... (13)
|         |    |  ├  - common
|         |    |    |  ├  - error  ... (14)
|         |    |    |    |  ├  - accessDeniedError.jsp
|         |    |    |    |  ├  - businessError.jsp
|         |    |    |    |  ├  - dataAccessError.jsp
|         |    |    |    |  ├  - invalidCsrfTokenError.jsp
|         |    |    |    |  ├  - missingCsrfTokenError.jsp
|         |    |    |    |  ├  - resourceNotFoundError.jsp
|         |    |    |    |  ├  - systemError.jsp
|         |    |    |    |  ├  - transactionTokenError.jsp
|         |    |    |    |  └  - unhandledSystemError.html
|         |    |    |  └  - include.jsp  ... (15)
|         |    |  ├  - layout  ... (16)
|         |    |    |  ├  - header.jsp
|         |    |    |  └  - template.jsp
|         |    |  └  - welcome
|         |    |       └- home.jsp  ... (17)
|         |  └  - web.xml  ... (18)
|         └- resources  ... (19)
|             └- app
|                 └- css
|                     └- styles.css  ... (20)
└- test
    ├- java
    └- resources
```

Sr. No.	Description
(12)	Directory that contains the Tiles configuration files. Refer [*Screen Layout using Tiles*] for the Tiles configuration files.

次のページに続く

Table. 3.3 – 前のページからの続き

Sr. No.	Description
(13)	Directory that contains the View generation templates (jsp etc).
(14)	Directory that contains the JSP and HTML for displaying error screens. At the time of creation, JSPs (HTMLs) are stored corresponding to the errors that may occur during application execution. **Note: Error screen JSP and HTML should be modified according to the application requirements (Such as UI Terms).**
(15)	Common JSP files for include. This file is included at the beginning of all JSP files. Refer [*Creating common JSP for include*] for common JSP files for include.
(16)	Directory that contains the JSP files for the Tiles layout. Refer [*Screen Layout using Tiles*] for JSP files for the Tiles layout.
(17)	JSP file that displays the Welcome page.
(18)	Configuration definition file for the Web application.
(19)	Directory that contains the static resource files. This directory contains such files whose response contents are not going to change depending upon the request contents. Specifically following files are stored. • JavaScript files • CSS files • Image files • HTML files Here adopted a dedicated directory mechanism for managing static resources offered by Spring MVC.
(20)	CSS file that defines the screen style applied to the entire application.

Structure of Domain module

Module that manages the domain layer components are explained.

```
artifactId-domain
   ├  - pom.xml  ... (1)
```

Sr. No.	Description
(1)	The domain module configuration is defined in POM (Project Object Model) file. Following definitions are done in this file. • Definition of dependent libraries and build plug-ins • Definition to create a jar file

```
└- src
    ├- main
    |  ├  - java
    |    |  └  - com
    |    |        └- example
    |    |            └- project
    |    |                └- domain  ... (2)
    |    |                    ├- model
    |    |                    ├- repository
    |    |                    └- service
    |  └  - resources
    |        └- META-INF
    |           ├  - dozer  ... (3)
    |           └  - spring  ... (4)
    |                ├- artifactId-codelist.xml  ... (5)
    |                ├- artifactId-domain.xml  ... (6)
    |                └- artifactId-infra.xml  ... (7)
```

Sr. No.	Description
(2)	Package for storing the domain layer classes.
(3)	The directory in which a mapping definition file of Dozer (Bean Mapper) is stored, Refer to [*Bean Mapping (Dozer)*] for Dozer. It is an empty directory at the time of creation. If the mapping file is required (if high mapping is required), it gets automatically read in case of stored under this directory. **Note:** Following files are stored under this directory. • Definition file for each other mapping of domain layer JavaBean
(4)	Directory contains the property file and Spring Framework bean definition file.
(5)	Bean definition file for defining the code list.
(6)	Bean definition file for defining the domain layer components. Following beans are defined in this file. • Domain layer components (Service, Repository etc) • Infrastructure layer components (Import the bean definition file that the component has been defined in the Infrastructure layer) • Components for transaction management provided from Spring Framework.
(7)	Bean definition file for defining the Infrastructure layer components. O/R Mapper etc beans are defined in this file.

```
└- test
    ├- java
    |   └   - com
    |         └- example
    |             └- project
```

```
    |                └- domain
    |                    ├- repository
    |                    └- service
    └- resources
        └- test-context.xml  ... (8)
```

Sr. No.	Description
(8)	Bean definition file for defining the domain layer unit test components.

In case of project created for MyBatis3

```
└- src
    ├- main
    |  ├  - java
   (...)
    |  └  - resources
    |        ├- META-INF
    |        |  ├  - dozer
    |        |  ├  - mybatis  ... (9)
    |        |    |  └  - mybatis-config.xml  ... (10)
    |        |  └  - spring
   (...)
    |        └- com
    |            └- example
    |                └- project
    |                    └- domain
    |                        └- repository  ... (11)
    |                            └- sample
    |                                └- SampleRepository.xml  ... (12)
```

Sr. No.	Description
(9)	Directory that contains the MyBatis3 configuration files
(10)	MyBatis3 configuration files. Some of the recommended settings are defined at the time of creation.
(11)	Directory that contains the MyBatis3 Mapper files.
(12)	Sample file of MyBatis3 Mapper file. Sample implementation is in commented out state at the time of creation **Lastly, these files will not required.**

Structure of Env module

Module that manages the environment dependent configuration files are explained.

```
artifactId-env
   ├  - configs  ... (1)
     |  ├  - production-server  ... (2)
     |    |  └  - resources
     |  └  - test-server
     |        └- resources
   ├  - pom.xml  ... (3)
```

Sr. No.	Description
(1)	Directory for managing the environment dependent configuration files. Manage environment dependent configuration file by creating subdirectories of each environment.
(2)	Directory for managing the each environment configuration file. At the time of creation, following directories (directory template) are provided as most simple configuration. • production-server (Directory that contains the production environment configuration files) • test-server (Directory that contains the test environment configuration files)
(3)	The env module configuration is defined in POM (Project Object Model) file. Following definitions are done in this file. • Definition of dependent libraries and build plug-ins • Definition of Profile to create a jar file for each environment

```
└- src
    └- main
        └- resources  ... (4)
            ├- META-INF
            |  └  - spring
            |        ├- artifactId-env.xml  ... (5)
            |        └- artifactId-infra.properties  ... (6)
            ├- database  ... (7)
            |  ├  - H2-dataload.sql
            |  └  - H2-schema.sql
            ├- dozer.properties  ... (8)
            ├- log4jdbc.properties  ... (9)
            └- logback.xml  ... (10)
```

Sr. No.	Description
(4)	Directory for managing configuration files of the development.
(5)	Bean definition file that defines the environment dependent components. Following beans are defined in this file. • Datasource • `JodaTimeDateFactory` offered by common library (In case of different implementations depending on the environment) • Components for transaction management provided by Spring Framework (In case of different implementations depending on the environment)
(6)	Property file that defines the environment dependent settings. At the time of creation, the DataSource settings are defined (Setting of the connection and connection pool)
(7)	Directory that contains the SQL to set up an in-memory database (H2 Database). This directory is prepared while performing small operation verification . **Basically remove this directory because this directory is not intended to use in the actual application development.**
(8)	Property file for carrying out the Dozer (Bean Mapper) global settings. For Dozer refer [*Bean Mapping (Dozer)*]. It is an empty file at the time of creation. (The warning log appears at the start-up time if file is not exist, the empty file is prepared in order to prevent it)
(9)	Property file for carrying out the Log4jdbc-remix (library to perform the JDBC-related log output) global settings. For Log4jdbc-remix, refer [*JDBC debug log settings*]. At the time of creation, new line character related setting are specified for those SQLs which are going to be printed in log.
(10)	Configuration file of the Logback (log output). For the log output refer [*Logging*].

Structure of Initdb module

Module that manages the SQL file to initialize the database is explained.

```
artifactId-initdb
   ├  - pom.xml  ... (1)
   └  - src
         └- main
             └- sqls  ... (2)
```

Sr. No.	Description
(1)	The initdb module configuration is defined in POM (Project Object Model) file. Following definitions are done in this file. • Definition of build plug-ins (SQL Maven Plugin) Simple configuration for PostgreSQL is defined at the time of creation.
(2)	Directory for storing the database initialization SQL files. It is an empty directory at the time of creation. For how to create, Refer Sample application of initdb project.

Note: Can be executed SQL using sql:execute of SQL Maven Plugin.

```
mvn sql:execute
```

Structure of Selenium module

Module that manages the E2E (End To End) testing components used in Selenium explained.

```
artifactId-selenium
   ├  - pom.xml  ... (1)
   └  - src
         └- test  ... (2)
             ├- java
            |   └  - com
            |         └- example
            |             └- project
            |                 └- selenium
            |                     └- welcome
            |                         └- HelloTest.java  ... (3)
             └- resources
                └- META-INF
                     └- spring
                         ├- selenium.properties  ... (4)
```

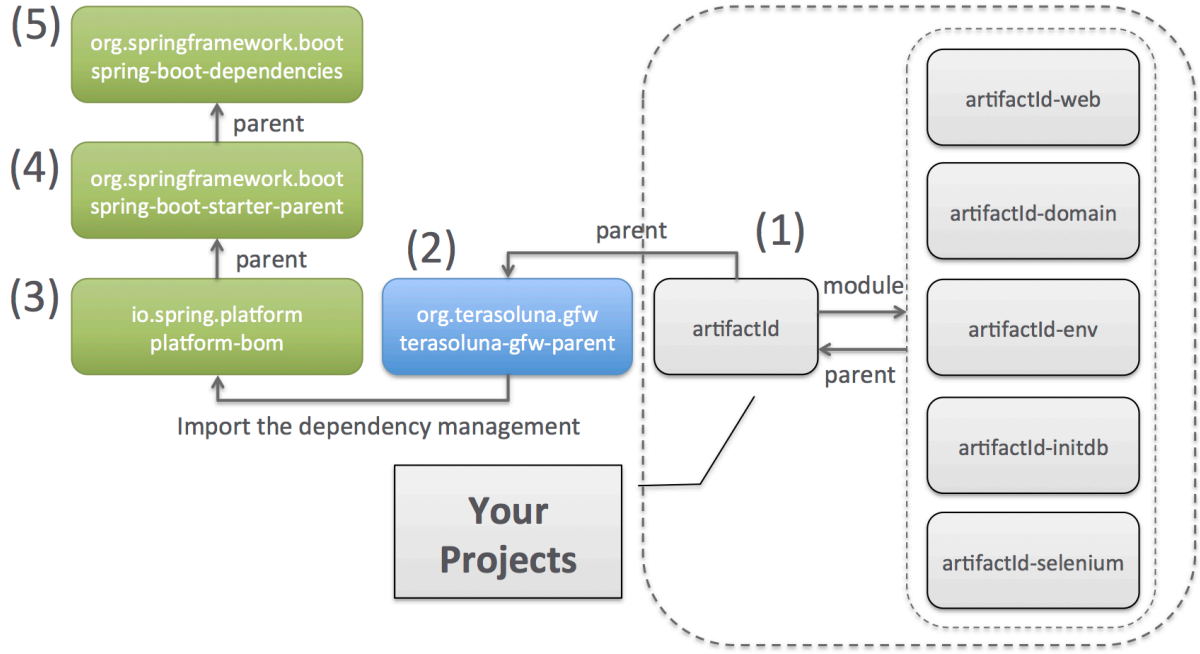
```
└- seleniumContext.xml  ... (5)
```

Sr. No.	Description
(1)	The selenium module configuration is defined in POM (Project Object Model) file. Following definitions are done in this file. • Definition of dependent libraries and build plug-ins • Definition to create a war file
(2)	Directory that contains the configuration files and testing components. For how to create, Refer Sample application of selenium project.
(3)	Sample test class using Selenium WebDriver. At the time of creation, it has the test method for asserting a title of the Welcome page.
(4)	Properties file that defines the settings to be used in the test. The URL of the application server is `http://localhost:8080/`at the time of creation.
(5)	Bean definition file for defining the test components. At the time of creation, it defines required settings for executing the sample test.

3.1.5 Appendix

Hierarchical structure of the project

The hierarchical structure of the project indicated below which is created in Maven Archetype.



Sr. No.	Description
(1)	Project created in Maven Archetype. The project created in Maven Archetype has become a multi-module configuration, parent project and each sub-module have a cross-reference relationship. In the project created in version 5.3.0.RELEASE Maven Archetype, [org.terasoluna.gfw:terasoluna-gfw-parent:5.3.0.RELEASE] is specified as a parent project.
(2)	TERASOLUNA Server Framework for Java (5.x) Parent project. In the TERASOLUNA Server Framework for Java (5.x) Parent project, • Plug-ins settings for build • Customization of libraries that is managed through Spring IO Platform (adjusted version) • Version management of recommended libraries that is not managed by Spring IO Platform are performed. Furthermore, in order to version management of the dependent libraries via Spring IO Platform, imported the [Spring IO Platform] into `<dependencyManagement>` of the project. Spring IO Platform Version (the current project using) is described in *OSS Versions*.
(3)	Spring IO Platform project. Since [org.springframework.boot:spring-boot-starter-parent:1.2.5.RELEASE] is specified as a parent project, the definition of `<dependencyManagement>` defined into pom file of the spring-boot-starter-parent also imported into pom file of the terasoluna-gfw-parent.

次のページに続く

Table. 3.4 – 前のページからの続き

Sr. No.	Description
(4)	Spring Boot Starter Parent project. Since [org.springframework.boot:spring-boot-dependencies:1.2.5.RELEASE] is specified as a parent project, the definition of `<dependencyManagement>` defined into pom file of the spring-boot-dependencies also imported into pom file of the terasoluna-gfw-parent.
(5)	Spring Boot Dependencies project.

Tip: The configuration has been changed like `<dependencyManagement>` of Spring IO Platform is imported from version 5.0.0.RELEASE, we have adopted a style that version management of recommended libraries are done in Spring IO Platform.

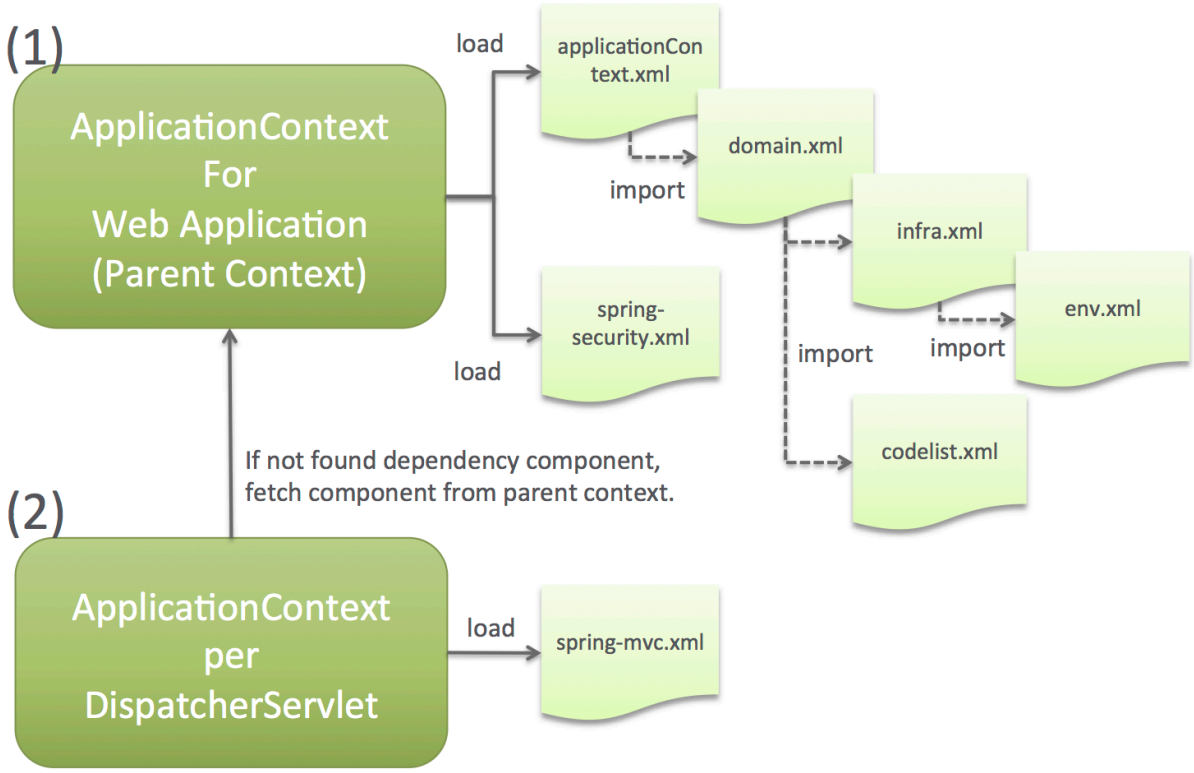
Warning: Since the configuration has been changed like `<dependencyManagement>` of Spring IO Platform is imported from version 5.0.0.RELEASE, You are no longer able to access the version management properties from the child project.

Therefore, if property values are referring or overwriting at the child project, pom file should be modified while upgrading from version 1.0.x.

Furthermore, it is possible to access the conventional version management properties for recommended libraries (TERASOLUNA Server Framework for Java (5.x) recommended library) which are not managed by the Spring IO Platform.

Relationship of bean definition file and application context structure

Relationship of bean definition file and structure of the Spring Framework application context (DI container) indicated below.



Sr. No.	Description
(1)	Application context for the Web application. As shown in above diagram, Components defined in • artifactId-web/src/main/resource/META-INF/spring/applicationContext.xml • artifactId-domain/src/main/resource/META-INF/spring/artifactId-domain.xml • artifactId-domain/src/main/resource/META-INF/spring/artifactId-infra.xml • artifactId-env/src/main/resource/META-INF/spring/artifactId-env.xml • artifactId-domain/src/main/resource/META-INF/spring/artifactId-codelist.xml • artifactId-web/src/main/resource/META-INF/spring/spring-security.xml are registered in the application context (DI container) for the Web application.. Components registered in the application context for the Web application are mechanized such a way that it can be referred by the application context of each `DispatcherServlet`.
(2)	Application context for `DispatcherServlet`. As shown in above diagram, Components defined in • artifactId-web/src/main/resource/META-INF/spring/spring-mvc.xml are registered in the application context (DI container) for the `DispatcherServlet`. Components not stored in the application context for the `DispatcherServlet` are mechanized such a way that it can be obtained by referring the application context of the Web application (parent context), hence it is possible to inject domain layer components for the application layer component.

Note: About the operation when registered the same components in both application contexts.

If same components are registered in both application context for web application and application context for `DispatcherServlet`, injected component will be the registered component in the same application context(Application context for `DispatcherServlet`) and this point is supplemented here.

In particular, it is necessary to be careful that do not register the domain layer component (such as Service and Repository) to application context for the `DispatcherServlet`.

If domain layer components are registered to the application context for the `DispatcherServlet`, trouble like the database operations are not committed occurs due to component that performs the transaction control (AOP) is not enabled.

Furthermore, the settings are done in the project created using Maven Archetype so that the above events don't occur. It is necessary to be careful while performing modification or addition of the settings.

Description of the configuration file

Todo

In order to increase the understanding of various settings, planning to add explanation of a configuration file.

- If functional description is explained somewhere, Reference to the functional description will be noted down.
- If functional description is explained anywhere, description will be done here.

Specific time-line is not decided yet.

Application development in offline environment

In "*Create development project*", a method to create a development project of multi-project configuration by using archetype:generate of Maven Archetype Plugin is described. Although Maven is used for the operations in the online environment, a method is described below for how to use it in offline environment as well.

To continue project development in the offline environment, the files like libraries and plugins necessary for development must be copied in advance. The operation below should be performed in **online environment**.

Move to root directory of development project. Here, the project created using "*Create development project*" is used for the explanation.

```
cd C:\work\todo
```

Copy the files like libraries and plugins necessary for project development. Files are copied by executing dependency:go-offline of Maven Archetype Plugin.

```
mvn dependency:go-offline -Dmaven.repo.local=repository
```

Parameter	Description
–Dmaven.repo.local	Specify copy destination. A new destination is created if a copy destination does not exist. At present, copy destination is specified as a repository.

Create a war file or a jar file in order to facilitate the distribution of deliverables. At that time, files like libraries and plugins necessary for build are copied.

```
mvn package -Dmaven.repo.local=repository
```

When build is successful, the log shown below is output.

```
(... omit)
[INFO] ------------------------------------------------------------------------
[INFO] Reactor Summary:
[INFO]
[INFO] TERASOLUNA Server Framework for Java (5.x) Web Blank Multi Project (MyBa
tis3) SUCCESS [  0.006 s]
[INFO] todo-env ........................................... SUCCESS [ 46.565 s]
[INFO] todo-domain ........................................ SUCCESS [  0.684 s]
[INFO] todo-web ........................................... SUCCESS [ 12.832 s]
[INFO] todo-initdb ........................................ SUCCESS [  0.067 s]
[INFO] todo-selenium ...................................... SUCCESS [01:13 min]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 02:14 min
[INFO] Finished at: 2015-10-01T10:32:34+09:00
[INFO] Final Memory: 36M/206M
[INFO] ------------------------------------------------------------------------
```

Above, files like libraries and plugins necessary for project development are copied. Operation is completed when the repository is copied to ${HOME}/.m2 of offline environment machine. If a process which has not been executed even once in online environment is executed in offline environment, necessary files like libraries and plugins cannot be fetched resulting in the process failure. However, by copying the files, the development can be continued uninterrupted even after moving to offline environment.

Warning: Precautions for the development in offline environment

Since it is not possible to fetch a new dependency relation from internet in the offline environment, POM (Project Object Model) file should not be edited. It is necessary to return to online environment again for editing POM file.

3.2 Domain Layer Implementation

3.2.1 Roles of domain layer

Domain layer **implements business logic**to be provided to the application layer.

Implementation of domain layer is classified into the following.

Sr. No.	Classification	Description
1.	*Implementation of Entity*	Creation of classes (Entity class) to hold business data.
2.	*Implementation of Repository*	Implementation of the methods to operate on business data. These methods are provided to Service classes. These are in particular the CRUD operations on Entity object.
3.	*Implementation of Service*	Implementation of the methods for executing business logic. These methods are provided to the application layer. Business data required by the business logic is fetched as the Entity object through the Repository.

This guideline recommends the structure of creating Entity classes and Repository for the following reasons.

1. **By splitting the overall logic into business logic (Service) and the logic to access business data, the implementation scope of business logic gets limited to the implementation of business rules**,
2. **Access logic to business data is standardized** by consolidating the operations of business data in the Repository.

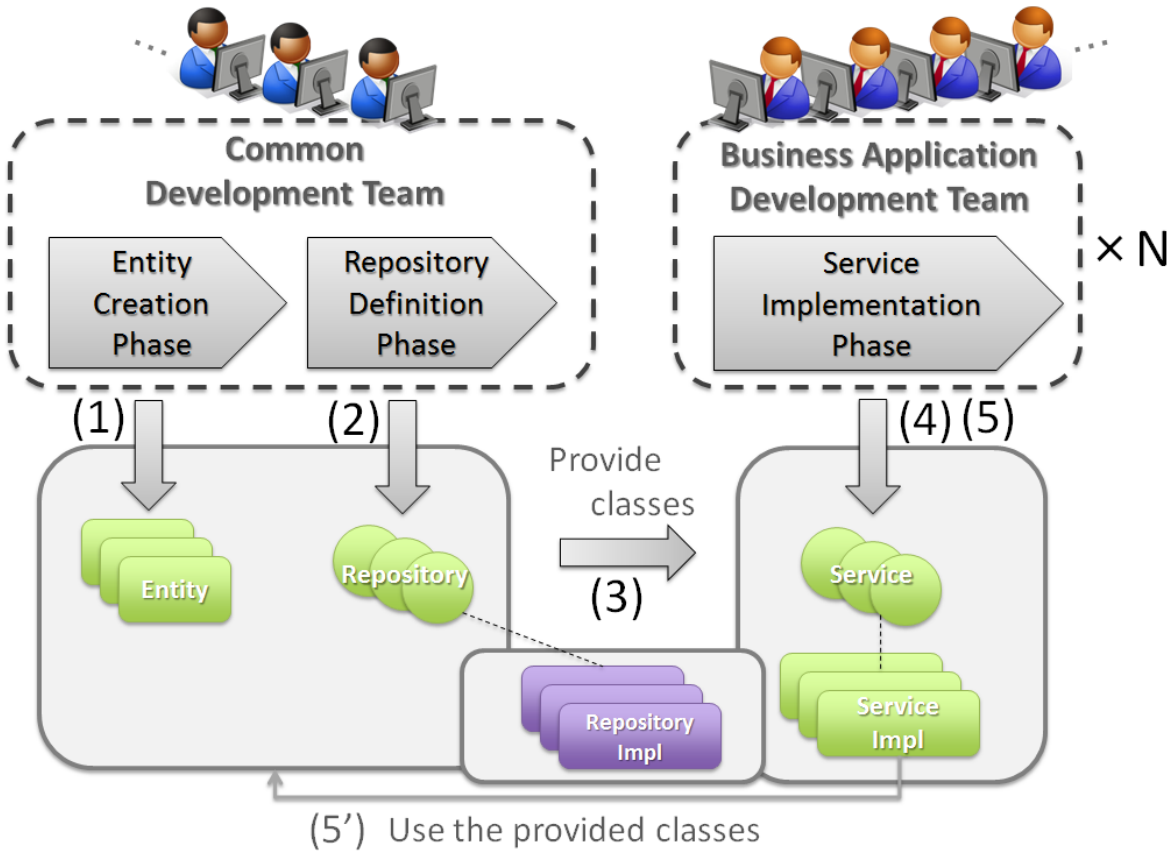
Note: Though this guideline recommends a structure to create Entity classes and Repository, it is not mandatory to perform development in this structure.

Decide a structure by taking into account the characteristics of the application as well as the project (structure of development team and development methodology).

3.2.2 Flow of development of domain layer

Flow of development of domain layer and allocation of roles is explained here.

A case where application is created by multiple development teams is assumed; however, the flow itself remains same even if developed by a single team.



Sr. No.	Team in-charge	Description
(1)	Common development team	Common development team designs and creates Entity classes.
(2)	Common development team	Common development team works out access pattern for the Entity classes extracted in (1) and designs methods of Repository interface. Common development team should implement the methods to be shared by multiple development teams.
(3)	Common development team	Common development team provides Entity classes and Repository created in (1) and (2) to the business application development team. At this time, it requests each business application development team to implement the Repository interface.
(4)	Business application development team	Business application development team takes charge of the implementation of Repository interface.
(5)	Business application development team	Business application development team develops Service interface and Service class using the Entity class and Repository provided by the common development team and the Repository implementation class created by the team itself.

Warning: A system having a large development scope is often developed by assigning the application to multiple teams. In that case, it is strongly recommended to provide a common team to design Entity classes and Repository.

When there is no common team, O/R Mapper(MyBatis, etc.) should be called directly from Service and a method to access business data should be adopted without creating Entity classes and Repository .

3.2.3 Implementation of Entity

Policy of creating Entity class

Create an Entity using the following method.

Specific creation method is shown in *Example of creating Entity class*.

Sr. No.	Method	Supplementary
1.	Create Entity class for each table.	However, Entity class is not required for mapping tables which represent the relationship between the tables. Further, when the tables are not normalized, Entity class for each table rule may not be applicable. Refer to the *Warning as well as Note outside this table* for the approach related to not-normalized tables.
2.	When there is a FK (Foreign Key) in the table, the Entity class of FK destination table must be defined as one of the properties of this Entity.	When there is 1:N relationship with FK destination table, use either `java.util.List<E>` or `java.util.Set<E>`. The Entity corresponding to the FK destination table is called as the related Entity in this guideline.
3.	Treat the code related tables as `java.lang.String` rather than as an Entity.	Code related tables are to manage the pairs of code value and name. When there is a need to bifurcate the process as per code values, `enum` class corresponding to code value should be created and it must be defined as property.

Warning: When table is not normalized, **check whether to use the method of creating the Entity classes and Repository** by considering the following points. Since the unnormalized tables do not have good compatibility with JPA, it is better not to use JPA.

- Creating an appropriate Entity class may often not be possible because of increased difficulty in creating entities if the tables are not normalized.

  In addition, efforts to create an Entity classes also increases.

  Two viewpoints must be taken into consideration here. Firstly “Can we assign an engineer

who can perform normalization properly?" and secondly "Is it worth taking efforts for creating normalized Entity classes?".

- If the tables are not normalized, the logic to fill the gap of differences between the Entity class and structure of table is required in data access.

  Here the viewpoint to be considered is, "Is it worth taking efforts to fill the gap of differences between the Entity class and structure of table ?".

The method of creating Entity classes and Repository is recommended; however, the characteristics of the application as well as the project (structure of development team and development methodology) must also be taken into account.

Note: If you want to operate business data as application, and as normalized Entity even if the tables are not normalized, it is recommended to use MyBatis as an implementation of RepositoryImpl of the infrastructure layer.

MyBatis is the O/R Mapper developed to map the SQL with object and not to map the database table record with object. Therefore, mapping to the object independent of table structure is possible depending on the implementation of SQL, .

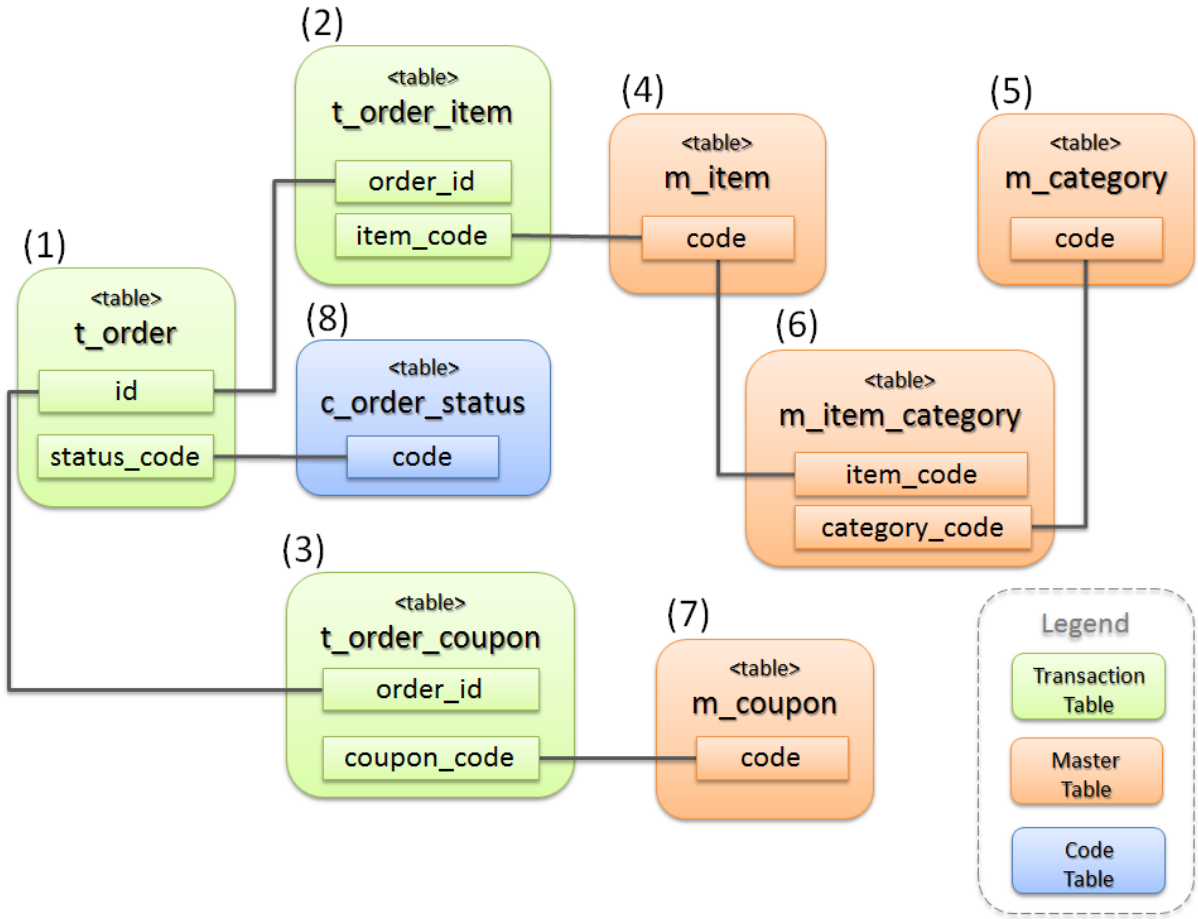
Example of creating Entity class

The creation of Entity class is explained using specific examples.

Following is an example of creating the business data of Entity classes required for purchasing a product on some shopping site.

Table structure

The table structure is as given below:



Sr. No.	Classification	Table name	Description
(1)	Transaction related	t_order	Table to store orders. 1 record is stored for 1 order.
(2)		t_order_item	Table to store the products purchased in 1 order. Record of each product is stored when multiple products are purchased in 1 order.
(3)		t_order_coupon	Table to store the coupon used in a single order. Record of each coupon is stored when multiple coupons are used in 1 order. No record is stored when coupon is not used.
(4)	Master related	m_item	Master table to define products.

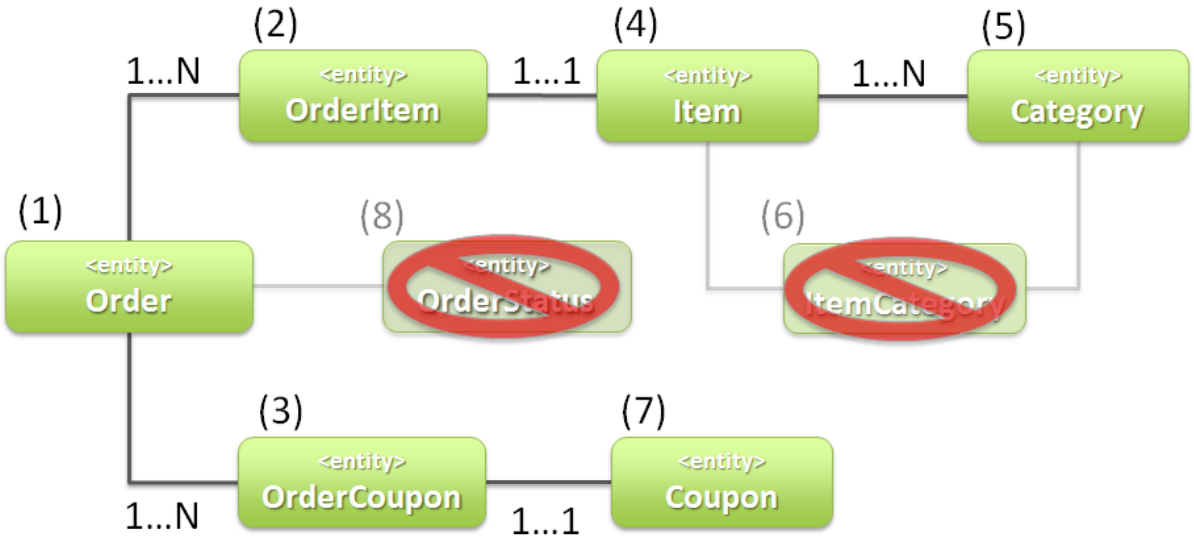
次のページに続く

Table. 3.5 – 前のページからの続き

Sr. No.	Classification	Table name	Description
(5)		m_category	Master table to define product category.
(6)		m_item_category	Master table to define the category of the product. Mapping between product and category is maintained. Model where 1 product belongs to multiple categories.
(7)		m_coupon	Master table to define coupons.
(8)	Code related	c_order_status	Code table to define order status.

Entity structure

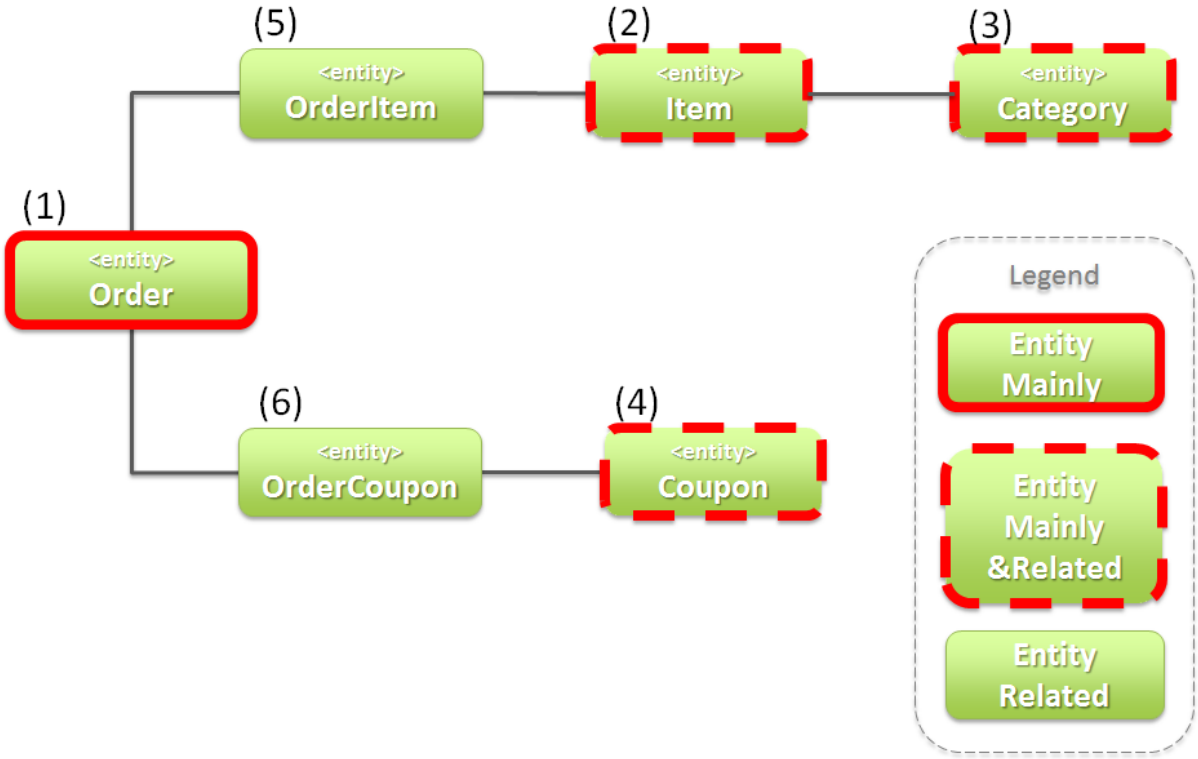
If Entity classes are created with the help of policy defined by the above table, it results into the following structure.



Sr. No.	Class name	Description
(1)	Order	Entity class indicating 1 record of t_order table. Multiple `OrderItem` and `OrderCoupon` are stored as the related Entity.
(2)	OrderItem	Entity class indicating 1 record of t_order_item table. `Item` is stored as the related Entity.
(3)	OrderCoupon	Entity class indicating 1 record of t_order_coupon table. `Coupon` is stored as the related Entity.
(4)	Item	Entity class indicating 1 record of m_item table. Multiple `Category` are stored as the related Entity. The association between `Item` and `Category` is done using m_item_category table.
(5)	Category	Entity class indicating 1 record of m_category table.
(6)	ItemCategory	Entity class is not created since m_item_category table is the mapping table to store the relationship between m_item table and m_category table.
(7)	Coupon	Entity class indicating 1 record of m_coupon table.
(8)	OrderStatus	Entity class is not created since c_order_status table is code table.

As it can be observed from the above entity diagram, it might first seem that Order class is the only main entity class in the shopping site application; however, there are other main entity class as well other than Order class.

Below is the classification of main Entity classes as well as Entity class which are not main.



The following 4 Entities are treated as the main Entity for creating shopping site application.

Sr. No.	Entity class	Reasons for treating as the main Entity.
(1)	Order class	It is one of the most important Entity class in the shopping site. Order class is the Entity indicating the order itself and a shopping site cannot be created without the Order class.
(2)	Item class	It is one of the most important Entity class in the shopping site. Item class is the Entity indicating the products handled in the shopping site and a shopping site cannot be created without Item class.
(3)	Category class	Product categories are displayed usually on the top page or as a common menu in shopping sites. In such shopping sites, Category becomes a main entity. Usually operations like 'search category list' can be expected.
(4)	Coupon class	Often discounts through coupons are offered in the shopping sites as a measure of promoting sales of the products. In such shopping sites, Coupon becomes a main entity. Usually operations like 'search coupon list' can be expected.

The following are not main Entities for creating shopping site application.

Sr. No.	Entity class	Reason of not treating Entity as main Entity
(5)	OrderItem class	This class indicates 1 product purchased in 1 order and exists only as the related Entity of Order class. So OrderItem class should not be considered as main Entity.
(6)	OrderCoupon	This class indicates 1 coupon used in 1 order and exists only as the related Entity of Order class. So, OrderCoupon class should not be considered as main Entity.

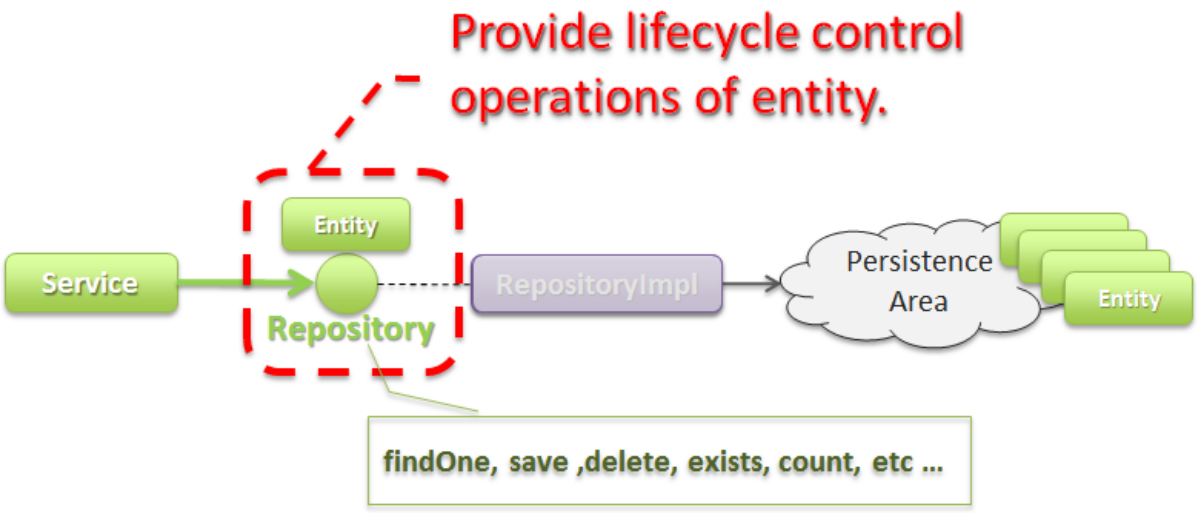
3.2.4 Implementation of Repository

Roles of Repository

Repository has following 2 roles.

1. **To provide to Service, the operations necessary to control Entity lifecycle (Repository interface).**

   The operations for controlling Entity lifecycle are CRUD operations.



2. **To provide persistence logic for Entity (implementation class of Repository interface).**

   Entity object should persist irrespective of the lifecycle (start and stop of server) of application.

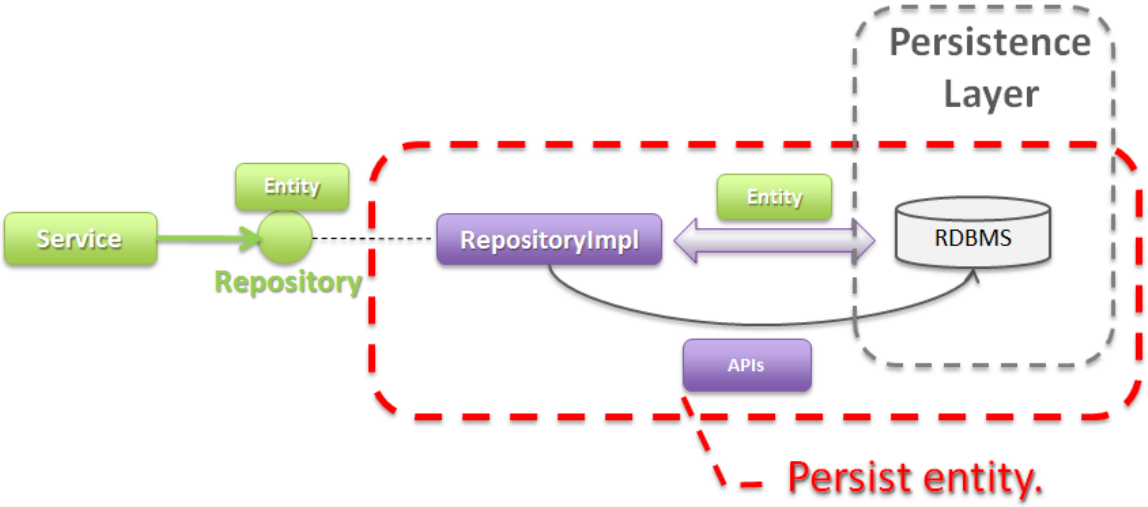
   Mostly relational database is the permanent destination of Entity. However, NoSQL database, cache server, external system and file (shared disk) can also be the permanent destination.

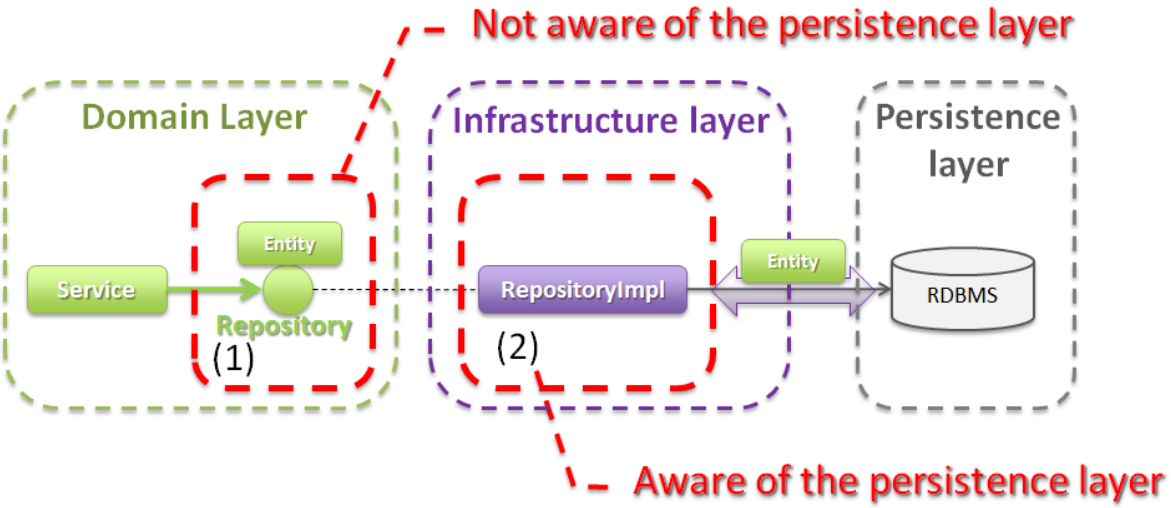
   The actual persistence processing is done using O/R Mapper API.

   This role is implemented in the RepositoryImpl of the infrastructure layer. Refer to *Implementation of Infrastructure Layer* for details.

Structure of Repository

Repository consists of Repository interface and RepositoryImpl and performs the following roles.





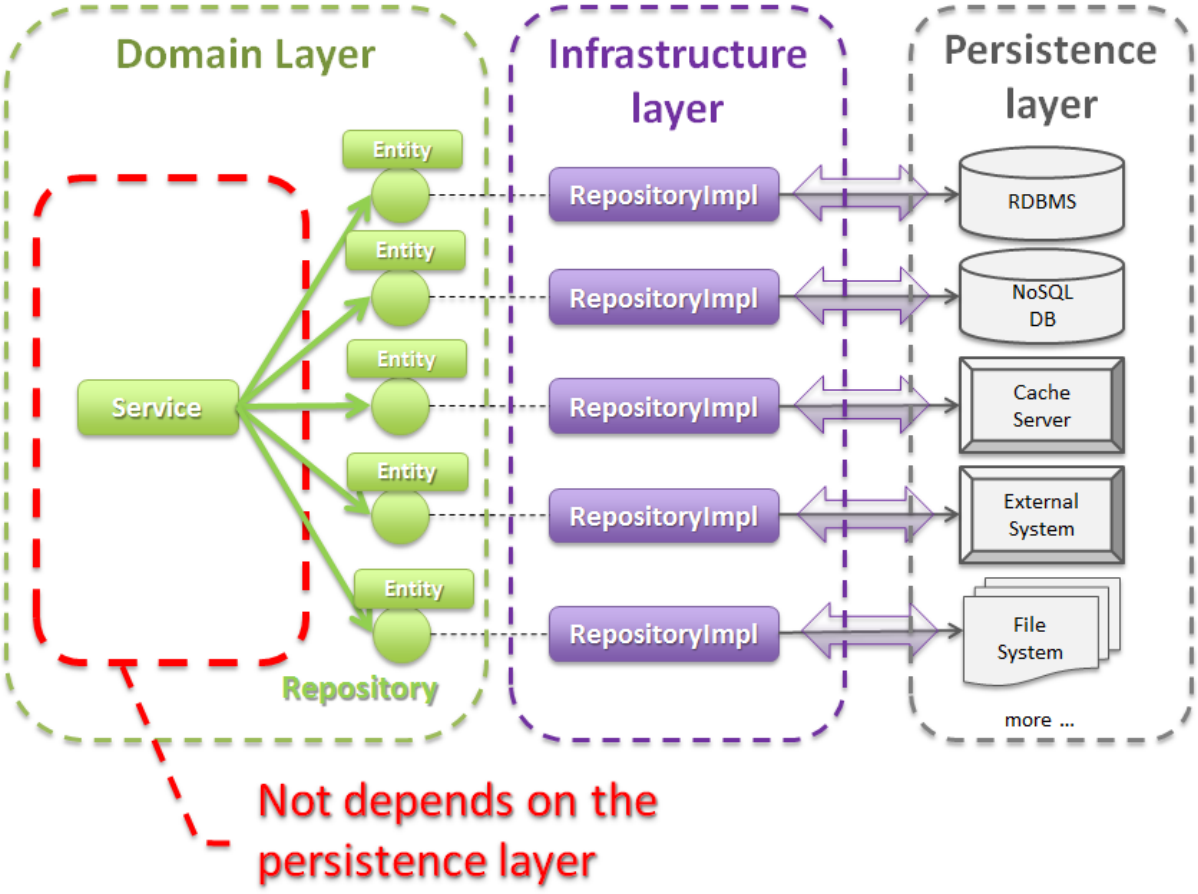
Sr. No.	Class(Interface)	Role	Description
(1)	Repository interface	Defines methods to control Entity lifecycle required for implementing business logic (Service).	Defines methods for CRUD operations of the Entity and is not dependent on persistence layer. Repository interface belongs to the domain layer since it plays the roles of defining the operations on Entity required for implementing business logic (Service).

次のページに続く

Table. 3.6 – 前のページからの続き

Sr. No.	Class(Interface)	Role	Description
(2)	RepositoryImpl	Implements the methods defined in Repository interface.	Implements CRUD operations of the Entity and is dependent on persistence layer. Performs actual CRUD processes using API that performs persistence provided by Spring Framework, O/R Mapper and middleware. RepositoryImpl belongs to infrastructure layer since it plays the role of implementing the operations defined in Repository interface. Refer to *Implementation of Infrastructure Layer* for the implementation of RepositoryImpl.

In case of multiple destinations in persistence layer, the resulting configuration as follows.
due to this, the logic depending on persistence platform of Entity is hidden from business logic (Service).



Note: Is it possible to hide 100% of persistence platform dependent logic from the Service class ?

In some cases it cannot be hidden completely due to constraints of persistence platform and the libraries used to access the platform. As much as possible, platform dependent logic should be implemented in RepositoryImpl instead of Service class. When it is difficult to exclude the platform dependent logic and merits of doing so are less, persistence platform dependent logic can be implemented as a part of business logic (Service) process.

A specific example of this is given here. There are cases when unique constraints violation error is needed to be handled when save method of `org.springframework.data.jpa.repository.JpaRepository` interface provided by Spring Data JPA is called. In case of JPA, there is a mechanism of cache entity operations and SQL is executed when transactions are committed. Therefore, since SQL is not executed even if save method of JpaRepository is called, unique constraints violation error cannot be handled in logic. There is a method (flush method) to reflect cached operations as means to explicitly issue SQLs in JPA. saveAndFlush

and flush methods are also provided in JpaRepository for the same purpose. Therefore, when unique constraints violation error needs to be handled using JpaRepository of Spring Data JPA, JPA dependent method (saveAndFlush or flush) must be called.

Warning: The most important purpose of creating Repository is not to exclude the persistence platform dependent logic from business logic. The most important purpose is to limit the implementation scope of business logic (Service) to the implementation of business rules. This is done by separating the operations to access business data in Repository. As an outcome of this, persistence platform dependent logic gets implemented in Repository instead of business logic (Service).

Creation of Repository

Repository must be created using the following policy only.

Sr. No.	Method	Supplementary
1.	Create Repository for the main Entity only.	This means separate Repository for operations of related Entity is not required. However, there are case when it is better to provide Repository for the related Entity in specific applications (for example, application having high performance requirements etc).
2.	Place Repository interface and RepositoryImpl in the same package of domain layer.	Repository interface belongs to domain layer and RepositoryImpl belongs to infrastructure layer. However, Java package of RepositoryImpl can be same as the Repository interface of domain layer.
3.	Place DTO used in Repository in the same package as Repository interface.	For example, DTO to store search criteria or summary DTO for that defines only a few items of Entity.

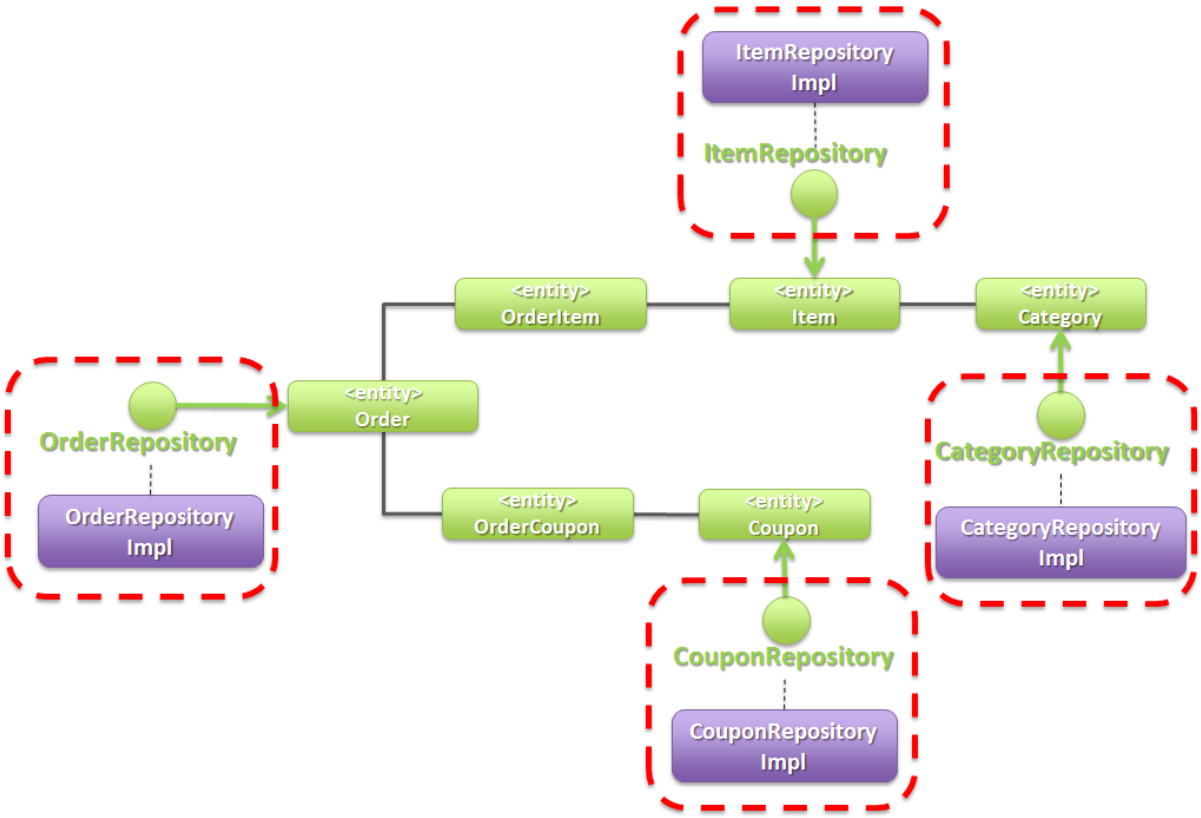
Example of creating Repository

An example of creating Repository is explained here.

An example of creating Repository of Entity class used in the explanation of *Example of creating Entity class* is as follows.

Structure of Repository

Entity class used in the explanation of *Example of creating Entity class* is used as an example, the resulting configuration is as follows:



Repository is created for the main Entity class.

Refer to *Project structure* for the recommended package structure.

Definition of Repository interface

Creation of Repository interface

An example of creating Repository interface is introduced below.

- `SimpleCrudRepository.java`

This interface provides only simple CRUD operations.

Method signature is created by referring to `CrudRepository` interface and `PagingAndSortingRepository` provided by Spring Data.

```
public interface SimpleCrudRepository<T, ID extends Serializable> {
    // (1)
    T findOne(ID id);
    // (2)
    boolean exists(ID id);
    // (3)
    List<T> findAll();
    // (4)
    Page<T> findAll(Pageable pageable);
    // (5)
    long count();
    // (6)
    T save(T entity);
    // (7)
    void delete(T entity);
}
```

Sr. No.	Description
(1)	Method to fetch the Entity object of specified ID.
(2)	Method to determine if the Entity of specified ID exists or not.
(3)	Method to retrieve the list of all Entities. In Spring Data, it was `java.util.Iterable`. Here as a sample, it is set to `java.util.List`.
(4)	Method to fetch collection of Entity objects corresponding to the specified pagination information (start position, record count, sort information). `Pageable` and `Page` are the interfaces provided by Spring Data.
(5)	Method to fetch total number of Entity objects.
(6)	Method to save (create, update) the specified Entity collection.
(7)	Method to delete the specified Entity.

- `TodoRepository.java`

  An example of creating Repository of Todo Entity, which was created in tutorial, on the basis of `SimpleCrudRepository` interface created above is shown below.

```
// (1)
public interface TodoRepository extends SimpleCrudRepository<Todo, String> {
    // (2)
    long countByFinished(boolean finished);
}
```

Sr. No.	Description
(1)	TodoRepository interface is created by specifying Todo entity in the generic type parameter “T” and String class in the generic type parameter “ID”.
(2)	Methods not provided by `SimpleCrudRepository` interface are added in this interface. In this case, “Method for acquiring count of Todo entity objects for which specified tasks have been finished” is added.

Method definition of Repository interface

It is recommended to have the same signature as `CrudRepository` and `PagingAndSortingRepository` provided by Spring Data for the methods performing general CRUD operations.

However, in case of returning collection, (`java.util.Collection` or `java.util.List`) interfaces which can be handled in a better way in logic are better than `java.lang.Iterable`.

In real development environment, it is difficult to develop an application using only general CRUD operations. Hence additional methods are required.

It is recommended to add the methods as per the following rules.

Sr. No.	Types of methods	Rules
1.	Method for searching a single record	1. Method name beginning with **findOneBy**to indicate that this method fetches a single record that matches with the condition. 2. In the method name after “findOneBy”, physical or logical name of the field used as search condition must be specified. Hence, the method name must be such that it becomes possible to estimate “the kind of entity that can be fetched using this method”. 3. There must be an argument for each search condition. However, when there are many conditions, DTO containing all search conditions can be provided. 4. Return value must be Entity class.

次のページに続く

Table. 3.7 – 前のページからの続き

Sr. No.	Types of methods	Rules
2.	Method for searching multiple records	1. Method name beginning with **findAllBy**to indicate that this method fetches all the records that matches with the condition. 2. In the method name after "findAllBy", physical or logical name of the field used as search condition must be specified. Hence, the method name must be such that it becomes possible to estimate "the kind of entity that can be fetched using this method". 3. There must be an argument for each search condition. However, when there are many conditions, DTO containing all search conditions can be provided. 4. Return value must be collection of Entity class.
3.	Method for searching multiple records with pagination	1. Method name beginning with **findPageBy**to indicate that this method fetches pages that matches with the condition. 2. In the method name after "findPageBy", physical or logical name of the field used as search condition must be specified. Hence, the method name must be such that it becomes possible to estimate "the kind of entity that can be fetched using this method". 3. There must be an argument for each search condition. However, when there are many conditions, DTO containing all search conditions can be provided. `Pageable` provided by Spring Data should be the interface for pagination information (start position, record count, sort information). 4. Return value should be `Page` interface provided by Spring Data.
4.	Count related method	1. Method name beginning with **countBy** to indicate that this method fetches count of Entities which matches with the condition. 2. Return value must be long type. 3. In the method name after "countBy", physical or logical name of the field used as search condition must be specified. Hence, the method name must be such that it becomes possible to estimate "the kind of entity that can be fetched using this method". 4. There must be an argument for each search condition. However, when there are many conditions, DTO containing all search conditions can be provided.

次のページに続く

Table. 3.7 – 前のページからの続き

Sr. No.	Types of methods	Rules
5.	Method for existence check	1. Method name beginning with **existsBy** to indicate that this method checks the existence of Entity which matches with the condition. 2. In the method name after “existsBy”physical or logical name of the field used as search condition must be specified. Hence, the method name must be such that it becomes possible to estimate “the kind of entity that can be fetched using this method”. 3. There must be an argument for each search condition. However, when there are many conditions, DTO containing all search conditions can be provided. 4. Return value must be boolean type.

Note: In case of methods related to update processing, it is recommended to construct methods in the same way as shown above. “find” in the method name above can be replaced by “update” or “delete”.

- `Todo.java` (Entity)

```
public class Todo implements Serializable {
    private String todoId;
    private String todoTitle;
    private boolean finished;
    private Date createdAt;
    // ...
 }
```

- `TodoRepository.java`

```
public interface TodoRepository extends SimpleCrudRepository<Todo, String> {
    // (1)
    Todo findOneByTodoTitle(String todoTitle);
    // (2)
    List<Todo> findAllByUnfinished();
    // (3)
    Page<Todo> findPageByUnfinished();
    // (4)
    long countByExpired(int validDays);
    // (5)
    boolean existsByCreateAt(Date date);
}
```

Sr. No.	Description
(1)	Example of method that fetches TODO objects whose title matches with specified value (TODO in which todoTitle=[argument value]). Physical name(todoTitle) of condition field is specified after findOneBy.
(2)	Example of method that fetches unfinished TODO objects (TODO objects where finished=false). Logical condition name is specified after findAllBy.
(3)	Example of method that fetches pages of unfinished TODOs (TODO objects where finished=false). Logical condition name is specified after findPageBy.
(4)	Example of method that fetches count of TODO objects for which the finish deadline has already passed (TODO for which createdAt < sysdate - [finish deadline in days] && finished=false). Logical condition name is specified after countBy.
(5)	Example of method that checks whether a TODO is created on a specific date (createdAt=specified date). Physical name (createdAt) is specified after existsBy.

Creation of RepositoryImpl

Refer to *Implementation of Infrastructure Layer* for the implementation of RepositoryImpl.

3.2.5 Implementation of Service

Roles of Service

Service plays the following 2 roles.

1. **Provides business logic to Controller.**

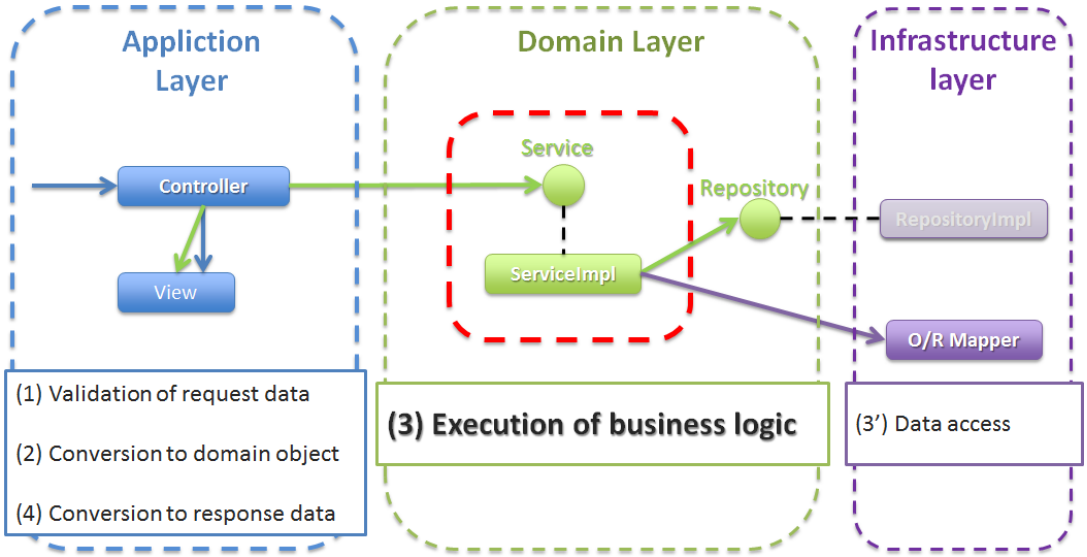
Business logic consists of create, update, consistency check etc of business data as well as all the processes related to business logic.

Create and update process of business data should be delegated to Repository(or O/R Mapper) and **service should be limited to implementation of business rules.**

Note: Regarding distribution of logic between Controller and Service

In this guideline, the logic to be implemented by Controller and Service should be as per the rules given below.

1. For the data requested from the client, single item check and correlated item check is to be performed in Controller (Bean Validation or Spring Validator).
2. Conversion processes (Bean conversion, Type conversion and Format conversion) for the data to be passed to Service, must be performed in Controller instead of Service.
3. **Business rules should be implemented in Service.**Access to business data is to be delegated to Repository or O/R Mapper.
4. Conversion processes (Type conversion and Format conversion) for the data received from Service (data to respond to the client), must be performed in Controller (View class etc).



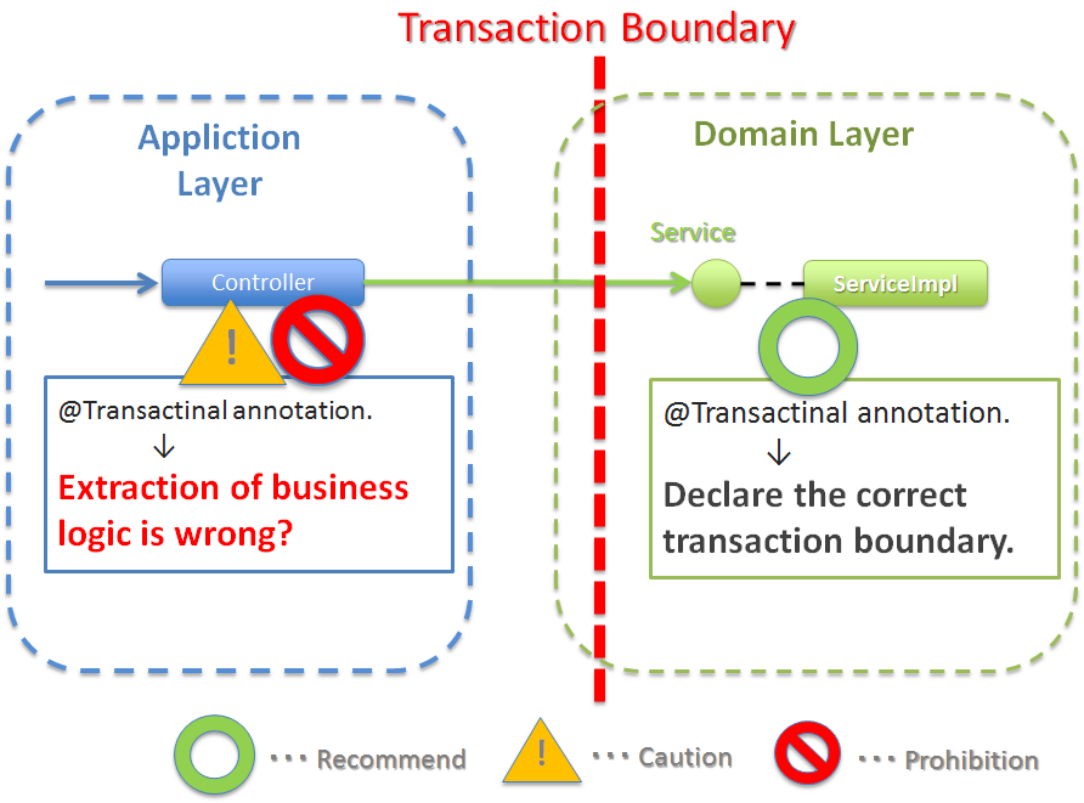
2. **Declare transaction boundary.**

   Declare transaction boundary when business logic is performing any operation which requires ensuring data consistency (mainly data update process).

   Even in case of logic that just read the data, often there are cases where transaction management is required due to the nature of business requirements. In such cases, declare transaction boundary.

   Transaction boundary must be set in Service layer as a principle rule. If it is found to be set in application layer (Web layer), there is a possibility that the extraction of business logic has

not been performed correctly.



Refer to *Regarding transaction management* for details.

Structure of Service class

Service consists of Service classes and SharedService classes and plays the following role.

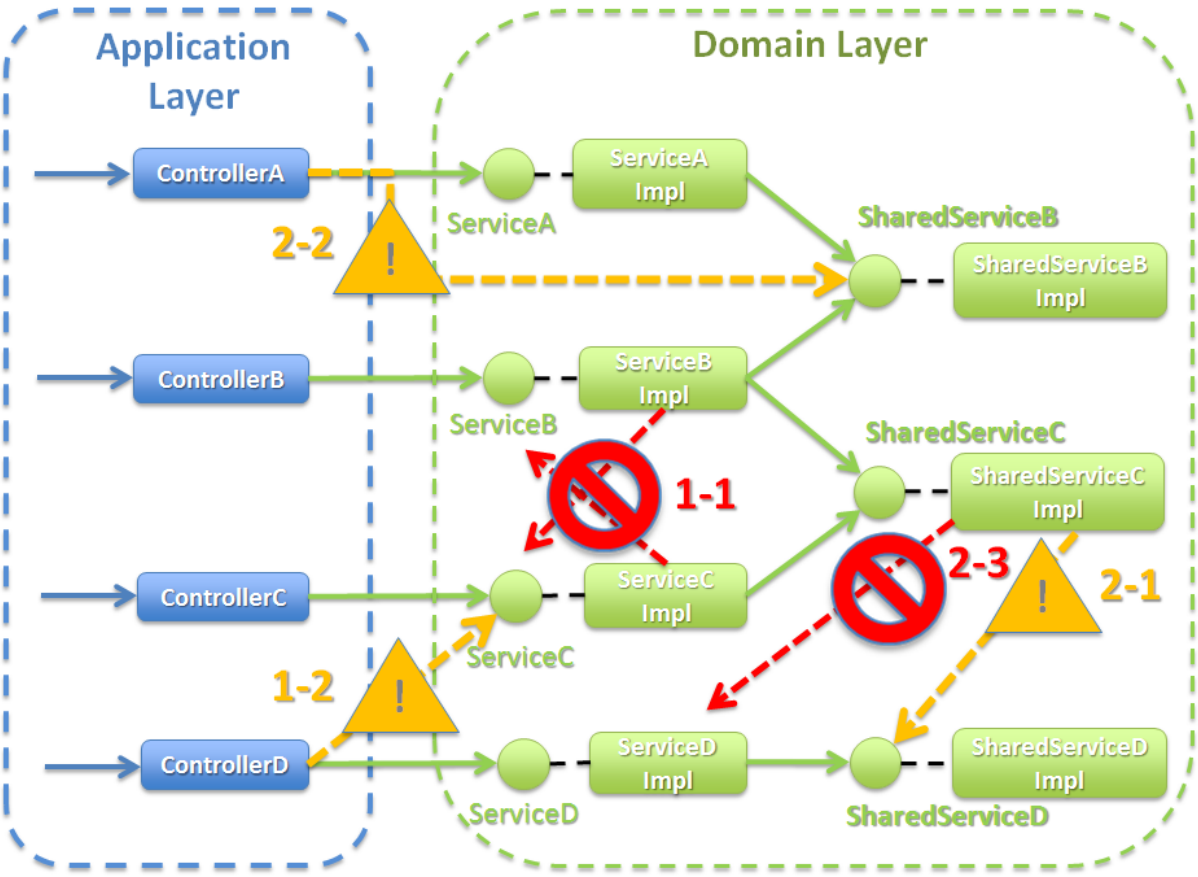
In this guideline, POJO (Plain Old Java Object) having `@Service` annotation is defined as Service or SharedService class.

We are not preventing the creation of interface and base classes that limit the signature of methods.

Sr. No.	Class	Role	Notes related to dependency relationship
1.	Service class	**Provides business logic to the specific Controller.** Service class methods **must not implement logic that need to be reused.**	1. **It is prohibited to call a method of Service class from another Service class method (Figure 1-1).**For shared logic, create SharedService class. 2. Method of Service class can be called from multiple Controllers (Figure 1-2). However, **it must be created for each controller when processing is to be branched based on the calling controller.**In such a scenario, create a method in SharedService class and call that method from the individual Service class methods.
2	SharedService class	**Provides shared (reusable) logic**for multiple Controllers and Service classes.	1. Methods of other SharedService classes can be called from a SharedService (Figure 2-1). However, **Calling hierarchy should not become complicated.** If calling hierarchy becomes complicated, there is a risk of reduction in maintainability. 2. Methods of SharedService classes can be called from Controller (Figure 2-2). However, **it can be only be done if there is no problem from transaction management perspective.**If there is a problem from transaction management perspective, first create a method in Service class and implement transaction management in this method. 3. **It is prohibited to call methods of Service class from SharedService (Figure 2-3).**

Dependency relationship of Service class and SharedService class is shown below.

The numbers inside the diagram are related to the numbering in “Notes related to dependency relationship” column of the above table.



Reason for separating Service and SharedService

Logic that cannot be (should not be) reused and logic that can be (should be) reused exist in the business logic.

To implement these 2 logics in the same class, it is difficult to decide whether a method can be re-used or not.

To avoid this problem, **it is strongly recommended to implement the method to be re-used in the SharedService class**in this guideline.

Reason for prohibiting the calling of other Service classes from Service class

In this guideline, calling methods of other Service classes from a Service class is prohibited.

Service provides business logic to a specific controller and is not created with the assumption of using it from other services.

If it is called directly from other Service classes, the following situations can easily occur **and there is a risk of reduced maintainability.**

Sr. No.	Situations that can occur
1.	The logic that must be implemented in the calling service class, gets implemented in the called service class for reasons like “having the logic at a single location” etc. As a result, **arguments for identifying the caller, get added to the method easily; Ultimately, the logic is incorrectly abstracted out as shared logic (like utilities). It results into a modular structure without much insight.**
2.	If the stack patterns or stack of services calling each other is large in number, **understanding the impact of modifications in source-code due to change in specifications or bug fixes, becomes difficult.**

Regarding interface and base classes to limit signature of method

In order to bring consistency in development of business logic, interfaces and base classes are created which limit the signature of the methods.

The purpose is also to prevent the injection of differences due to development style of each developer by limiting the signature through interfaces and base classes.

Note: In large scale development, there are situations where not every single developer is highly skilled or situations like having consistency in development of business logic considering maintainability after servicing. In such situations, limiting the signature through interfaces can be an appropriate decision.

In this guideline, we do not specifically recommended to create interface to limit signature; however, type of architecture must be selected on the basis of characteristics of the project. decide the type of architecture taking into account the project properties.

Note: Sample of implementation of interface and base classes to limit signature - Interface to limit signature

```
// (1)
public interface BLogic<I, O> {
  O execute(I input);
}
```

Sr. No.	Description
(1)	Interface to limit signature of implementation method of business logic. In the above example, it is defined as generic type of input (I) and output (O) information having one method (execute) for executing business logic. In this guideline, the above interface is called BLogic interface.

To standardize process flow of business logic when a fixed common process is included in Service, base classes are created to limit signature of method.

- Base classes to limit signature

```
// (2)
@Service
@Transactional
public abstract class AbstractBLogic<I, O> implements BLogic<I, O> {

    public O execute(I input){
      try{

          // omitted

          // (3)
          preExecute(input);

          // (4)
          O output = doExecute(input);

          // omitted

          return output;
      } finally {
          // omitted
      }

    }

    protected abstract void preExecute(I input);

    protected abstract O doExecute(I input);

}
```

Sr. No.	Description
(2)	While creating a base class, since the method executed externally and the class which implements the method are targets of AOP in the specification of *@Transactional*, they are assigned to base class when the transaction control is necessary. Similar to *@Servicve*, it must be assigned to the base class when Service is considered as an target using AOP, as for *ResultMessagesLoggingInterceptor*.
(3)	Call the method to perform pre-processing before executing business logic from base classes. In the preExecute method, business rules are checked.
(4)	Call the method executing business logic from the base classes.

Sample of extending base classes to limit signature is shown below.

- BLogic class (Service)

```
// (5)
public interface XxxBLogic extends BLogic<XxxInput, XxxOutput> {

}
```

Sr. No.	Description
(5)	Create an interface that inherits BLogic interface in order to enable type-safe injection. Implement a sub-interface which inherits BLogic in order to enable calling via a method of new interface.

```
@Service
public class XxxBLogicImpl extends AbstractBLogic<XxxInput, XxxOutput>␣
↪implements XxxBLogic {

    // (6)
    @Override
    protected void preExecute(XxxInput input) {

        // omitted
        Tour tour = tourRepository.findOne(input.getTourId());
        Date reservationLimitDate = tour.reservationLimitDate();
```

```
        if(input.getReservationDate().after(reservationLimitDate)){
            throw new BusinessException(ResultMessages.error().add("e.xx.
↪xx.0001"));
        }

    }

    // (7)
    @Override
    protected XxxOutput doExecute(XxxInput input) {
        TourReservation tourReservation = new TourReservation();

        // omitted

        tourReservationRepository.save(tourReservation);
        XxxOutput output = new XxxOutput();
        output.setTourReservation(tourReservation);

        // omitted
        return output;
    }

}
```

Sr. No.	Description
(6)	Implement pre-process before executing business logic. Business rules are checked.
(7)	Implement business logic. Logic is implemented to satisfy business rules.

- Controller

```
// (8)
@Inject
XxxBLogic xxxBLogic;

public String reserve(XxxForm form, RedirectAttributes redirectAttributes)
↪{

    XxxInput input = new XxxInput();
    // omitted

    // (9)
```

```
    XxxOutput output = xxxBlogic.execute(input);

    // omitted

    redirectAttributes.addFlashAttribute(output.getTourReservation());
    return "redirect:/xxx?complete";
}
```

Sr. No.	Description
(8)	Controller injects BLogic interface to be called.
(9)	Controller calls execute method of BLogic interface and executes business logic.

Patterns of creating service class

There are mainly 3 patterns for creating Service.

Sr. No.	Unit	Creation method	Description
1.	For each Entity	Create Service paired with the main Entity.	Main Entity is in other words, business data. **If the application is to be designed and implemented with focus on business data, then Service classes should be created in this way.** If service is created in this way, business logic will also be created for each entity and it will become to extract shared logic. However, if Service is created using this pattern, its affinity is not so good with the type of application which has to be developed by introducing a large number of developers at the same time. It can be said that the pattern is suitable when for the developing small or medium sized application.

次のページに続く

Table. 3.8 – 前のページからの続き

Sr. No.	Unit	Creation method	Description
2.	For each use-case	Create Service paired with the use-case.	**If the application is to be designed and implemented with focus on events on the screen, Service should be created in this way.** If the Service is created using this pattern, it is possible to assign a person to each use case; hence, its affinity is good with the type of application which has to be developed by introducing a large number of developer at the same time. On the other hand, if Service is created using this pattern, shared logic within use case can be extracted to a single location; however, shared logic which spans across multiple use-case might not get extracted to a single location. When it is very important to have extract shared logic out to a single location, it becomes necessary devise measures like having a separate team to look after designing shared components of business logic that span across multiple use cases.

次のページに続く

Table. 3.8 – 前のページからの続き

Sr. No.	Unit	Creation method	Description
3.	For each event	Create Service paired with the events generated from screen.	**If the application is to be designed and implemented with focus on events on the screen and BLogic class is auto-generated using TERASOLUNA ViSC, Service should be created in this way.** In this guideline, the Service class created using this pattern is called `BLogic`. The characteristics of the application if creating Service using this pattern are basically same as those when creating Service for each use case. However, in this case extracting shared logic out of business logic might get more difficult compared to creation of Service for each use-case (Pattern No. 2). In this guideline, pattern of creating Service for each event is not specifically recommended. However, in large scale development, creating Service using this pattern can be considered as one of the options with the view of having consistency in development style of business logic from maintainability point of view.

Warning: The pattern of Service creation must be decided by taking into account the features of application to be developed and the structure of development team.

It is not necessary to narrow down to any one pattern out of the 3 indicated patterns. Creating Services using different patterns randomly should be avoided for sure; however, **patterns can be used in combinations, if policy of usage of patterns in certain specific conditions has been well-thought decision and has been directed by the architect.** For example, the following combinations are possible.

[Example of usage of patterns in combination]

- For the business logic very important to the whole application, create as SharedService class for each Entity.
- For the business logic to be processed for the events from the screen, create as Service class for each Controller.
- In the Service class for each controller, implement business logic by calling the sharedService as and when required.

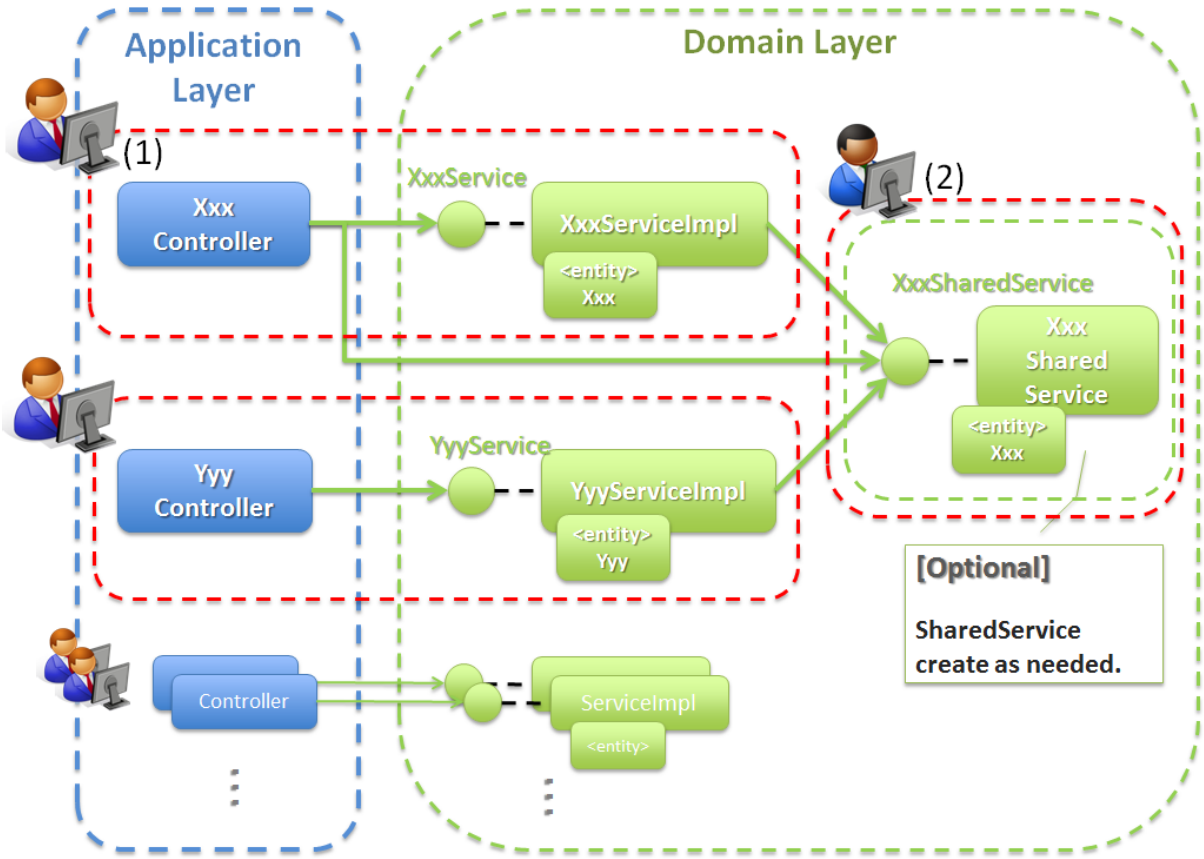
Tip: BLogic is generated directly from design documents when using “TERASOLUNA ViSC”.

Image of Application development - Creating Service for each Entity

Following is the image of application development when creating a Service for each Entity.

Note: An example of a typical application in which a Service is created for each Entity is a REST application. REST application provides CRUD operations (POST, GET, PUT, DELETE of HTTP) for published resources on HTTP. Most of the times, the resources published on HTTP are business data (Entity) or part of business data (Entity), they have good compatibility with the pattern of creating Service for each Entity.

In case of REST application, most of the times, use-cases are also extracted on a “per Entity” basis. Hence, the structure is similar to the case when Service is created for on a “per use-case” basis.



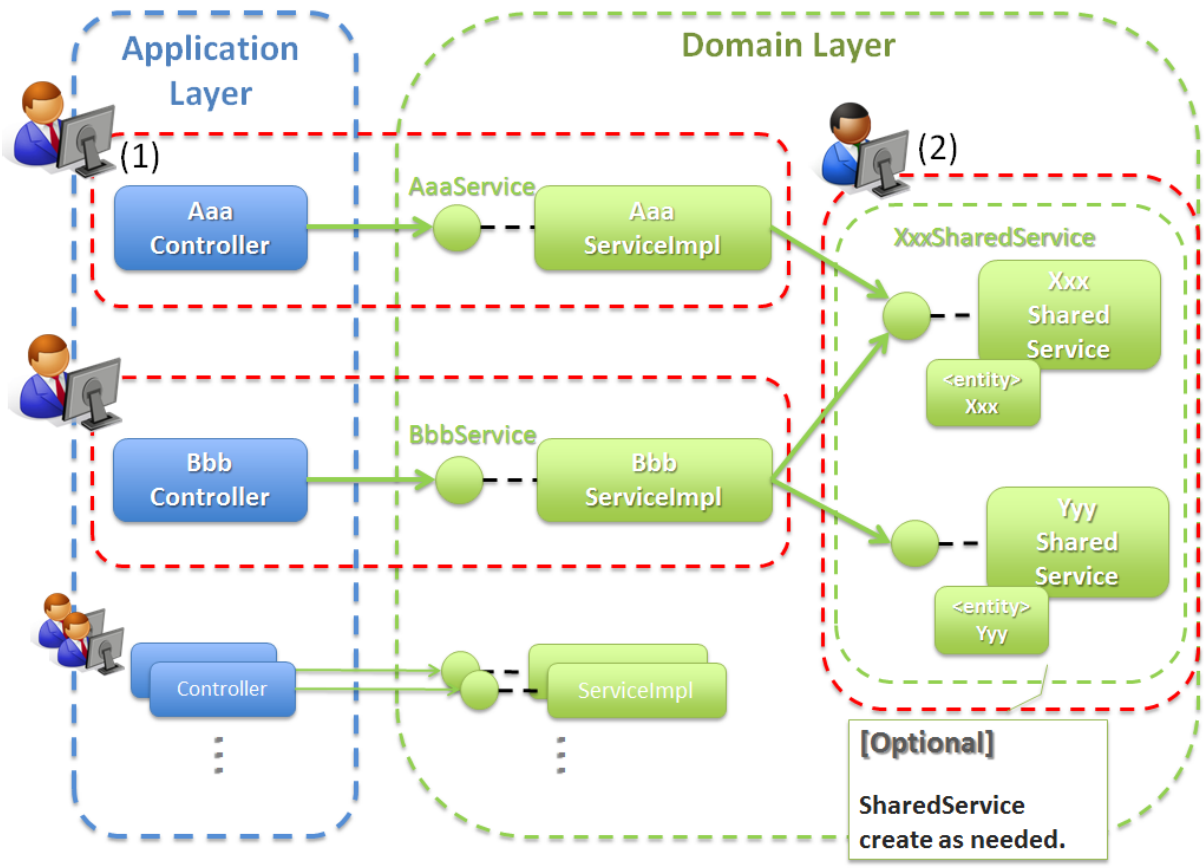
Sr. No.	Description
(1)	Implement Service by assigning a person for each Entity. If there is no specific reason, it is desirable that Controller must also be created for each Entity and must be developed by the same developer who created the Service class.
(2)	Implement SharedService if there is shared logic between multiple business logics. In the above figure, different person is assigned as the in-charge. However, he may be the same person as (1) depending per the project structure.

Image of Application development - Creating Service for each use case

Following is the image of application development when creating a Service for each use-case.

In case of use-case which performs CRUD operations on the Entity, structure is same as in case of creating

Service for each Entity.



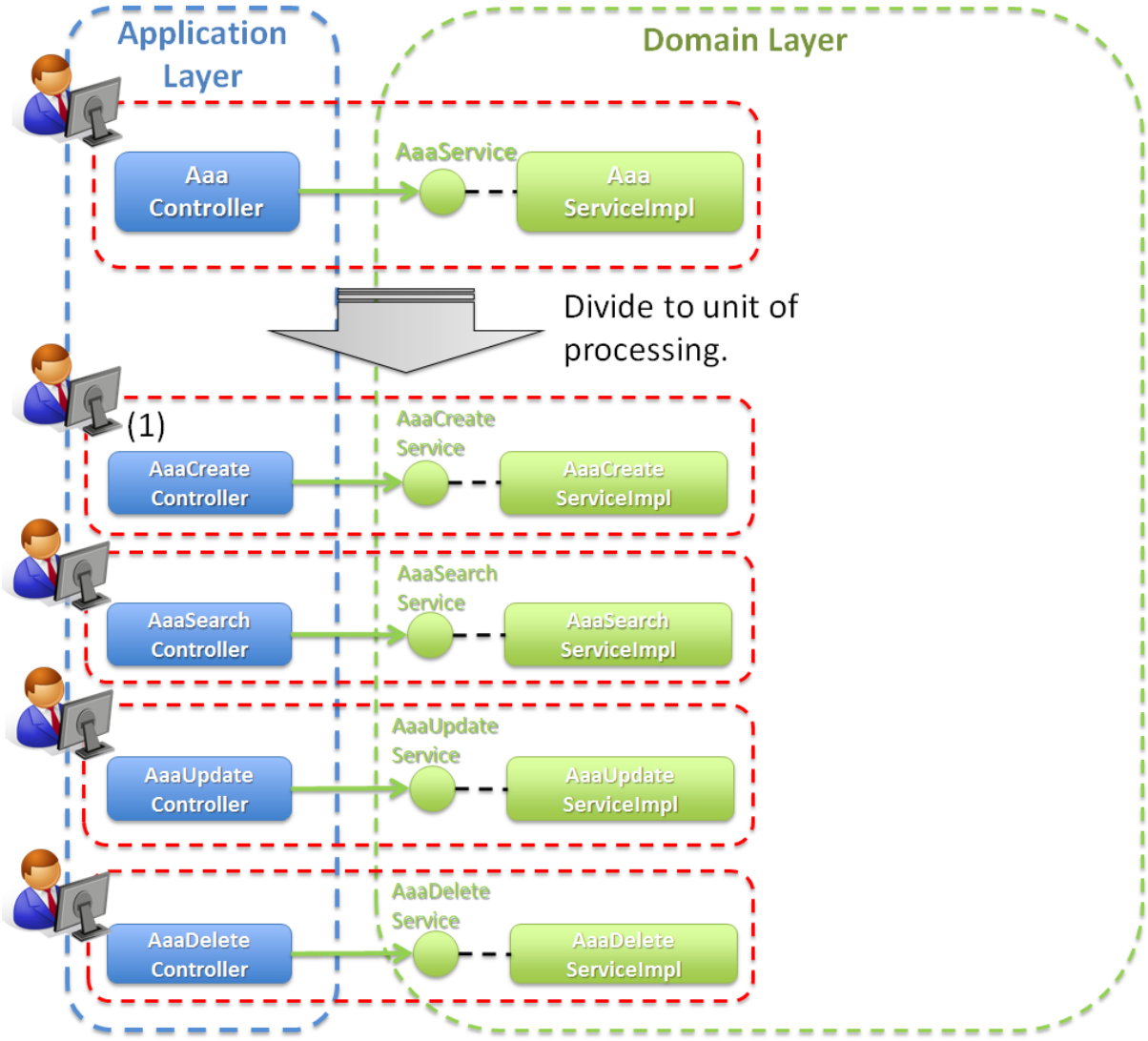
Sr. No.	Description
(1)	Implement Service by assigning a person for each use-case. If there is no specific reason, it is desirable that Controller must also be created for each use-case and must be developed by the same developer who created the Service class.
(2)	Implement SharedService if there is shared logic between multiple business logics. In the above figure, different person is assigned as the in-charge. However, he may be the same person as (1) depending per the project structure.

Note: With an increase in the size of the use-cases, the development scope of a person increases. At such a point of time, it becomes difficult to divide the work of this use-case with other developers. In case of application which has to be developed by introducing a large number of developer at the same time, the use-case can be further split into finer use-cases and which can then be allocated to more number of

developers.

Below is the image of application development when the use-case is further split.

Splitting a use-case has no impact on SharedService. Hence, the explanation is omitted here.

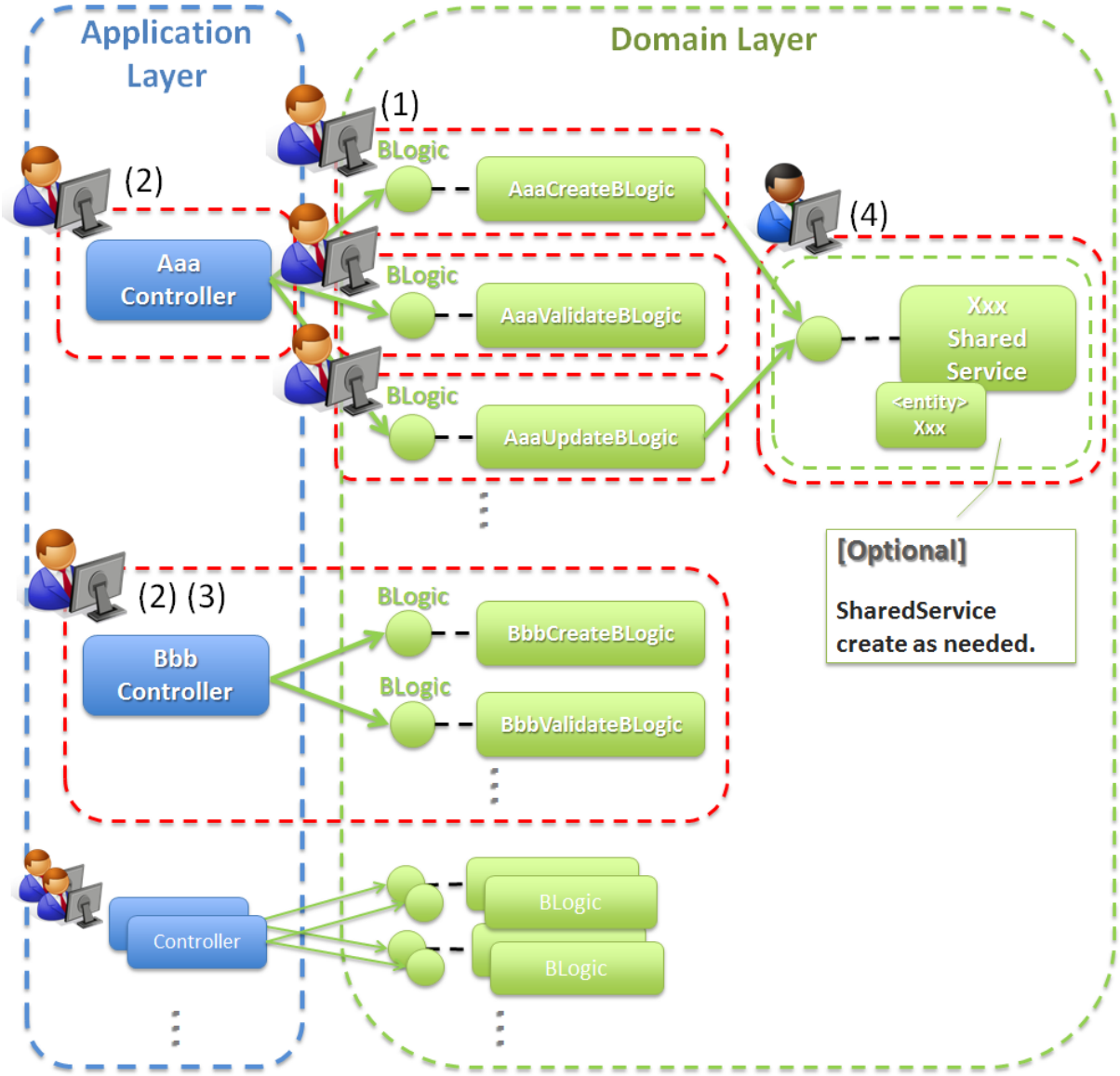


Sr. No.	Description
(1)	Divide the use-case into finer processes which make-up the complete use-case. Assign each fine process to a developer. Each developer creates the Service for assigned process. Note that the processes here are operations like search, create, update, delete etc. and these processes do not have a direct mapping to the processing required to be done for each event generated on screen. For example, if it the event generated on screen is “Update”, it includes multiple finer processes such as “Fetching the data to be updated”, “Compatibility check of update contents” etc. If there is no specific reason, it is desirable that Controller must also be created for each of these finer processes and must be developed by the same developer who creates the Service class.

Tip: In some projects, “group of use-cases” and “use-cases” are used in place of “use-case” and “processes” used in this guideline.

Image of Application development - Creating Service for each use event

Following is the image of application development when creating a Service(BLogic) for each event.



Sr. No.	Description
(1)	Implement Service(BLogic) by assigning a person for each event. Above example is an extreme case where separate developer is assigned for each service(BLogic). In reality, a single person must be assigned for a use-case.
(2)	If there is no specific reason, controller also should be created on “per use-case” basis.

次のページに続く

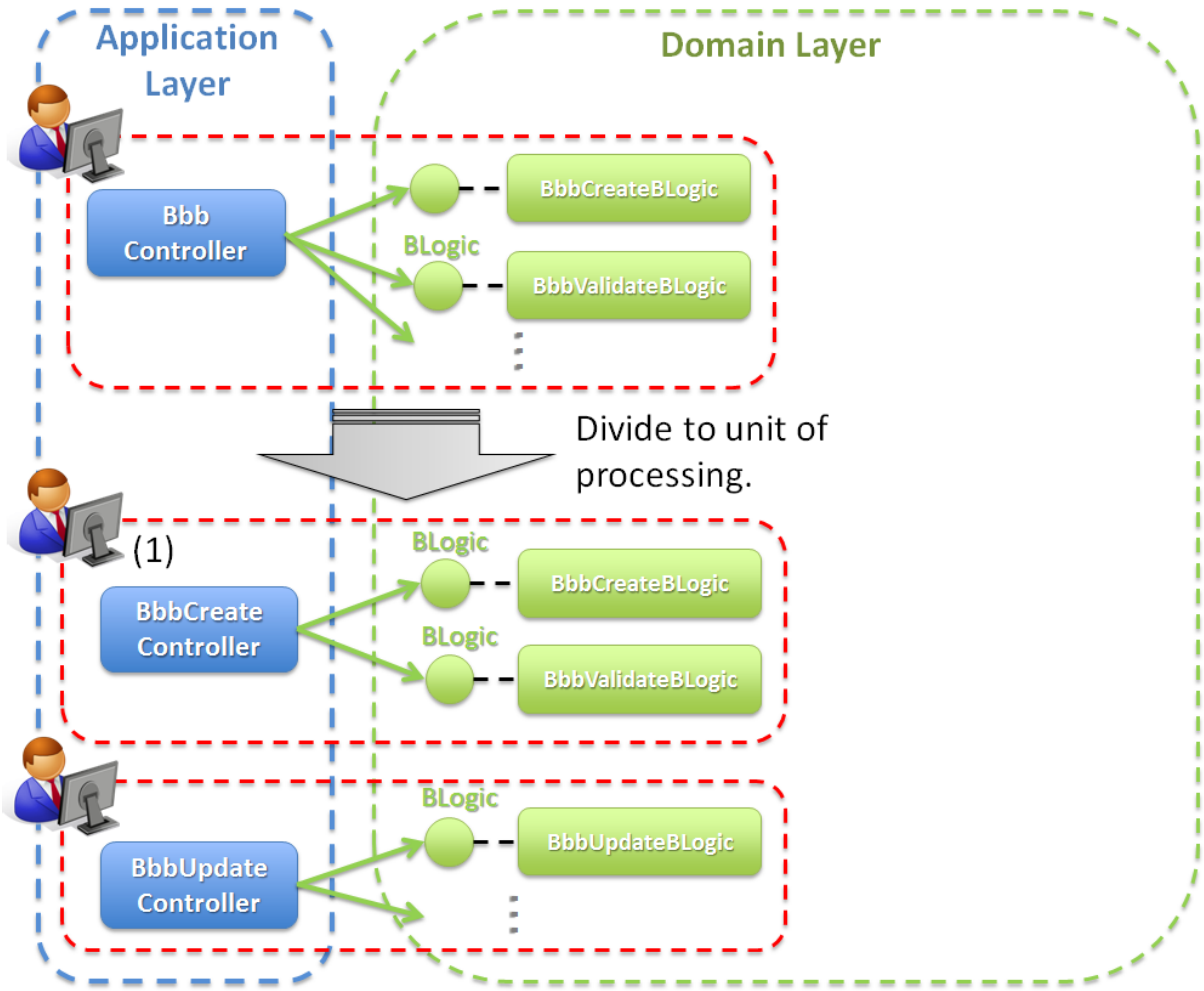
Table. 3.9 – 前のページからの続き

Sr. No.	Description
(3)	Even if the separate Service(BLogic) is created for each event, it is recommended that same person is the in-charge of the complete use-case.
(4)	Implement in SharedService to share the logic with multiple business logics. In the above figure, different person is assigned as the in-charge. However, he may be the same person as (1) depending per the project structure.

Note: With an increase in the size of the use-cases, the development scope of a person increases. At such a point of time, it becomes difficult to divide the work of this use-case with other developers. In case of application which has to be developed by introducing a large number of developer at the same time, the use-case can be further split into finer use-cases and which can then be allocated to more number of developers.

Below is the image of application development when the use-case is further split.

Splitting a use-case has no impact on SharedService. Hence, the explanation is omitted here.



Sr. No.	Description
(1)	Divide the use-case into finer processes which make-up the complete use-case. Assign each fine process to a developer. Each developer creates the Service for assigned process. Note that the processes here are operations like search, create, update, delete etc. and these processes do not have a direct mapping to the processing required to be done for each event generated on screen. For example, if it the event generated on screen is “Update”, it includes multiple finer processes such as “Fetching the data to be updated”, “Compatibility check of update contents” etc. If there is no specific reason, it is desirable that Controller must also be created for each of these finer processes and must be developed by the same developer who creates the Service class.

Creation of Service class

Methods of creating Service class

Below are the points to be taken care of while creating Service class.

- Creation of Service interface

```
public interface CartService { // (1)
    // omitted
}
```

Sr. No.	Description
(1)	**It is recommended to create Service interface.** By providing an interface, it is possible to execute the method published as Service explicitly.

Note: Merits from architecture perspective

1. If interface is there, When using AOP, Dynamic proxies functionality of standard JDK is used. In case of no interface, CGLIB included in Spring Framework is used. In case of CGLIB there are certain restrictions like “Advice cannot be applied on final methods” etc. Refer to Spring Reference Document -Aspect Oriented Programming with Spring(Proxying mechanisms)-for details.
2. It becomes easier to create a stub of business logic. When application layer and domain layer are developed in parallel using different development teams, stubs of Service are required. When there is a need to create stubs, it is recommended to have interface .

- Creation of Service class

```
@Service // (1)
@Transactional // (2)
public class CartServiceImpl implements CartService { // (3) (4)
    // omitted
}
```

```
<context:component-scan base-package="xxx.yyy.zzz.domain" /> <!-- (1) -->
```

Sr. No.	Description
(1)	**Add @Service annotation to class.** By adding the above annotation, bean definition in configuration file is not required. Specify package for component scanning in `base-package` attribute of <context:component-scan> element. In case of this example, all the classes in "xxx.yyy.zzz.domain" is registered in container.
(2)	**Add @Transactional annotation to class.** By adding the above annotation, transaction boundary is set for to the all the methods of the Service class. `value` attribute should be specified as required. Refer to *Information required for "Declarative transaction management"* for details. Moreover, to understand the points to be noted when using `@Transactional` annotation, it is advisable to confirm with "*Settings for using transaction management*".
(3)	**Consider interface name as XxxService and class name as XxxServiceImpl.** Any naming conventions can be used. However, it is recommended to use distinguishable naming conventions for Service class and SharedService class.
(4)	**Service class must not maintain state. Register it in container as bean of singleton scope .** Objects (POJO such as Entity/DTO/VO) and values (primitive type, primitive wrapper class) where state changes in each thread should not be maintained in class level fields. Setting scope to any value other than singleton (prototype, request, session) using `@Scope` annotation is also prohibited.

Note: Reason for adding @Transactional annotation to class

Transaction boundary is required only for the business logic that updates the database. However, it is recommended to apply the annotation at class level to prevent bugs due to skipped annotation. However, defining `@Transactional` annotation only at required places (methods which update the database) is also fine.

Note: Reason to prohibit non-singleton scopes

1. prototype, request, session are the scopes for registering bean that maintains state. Hence they must not be used in Service class.
2. When scope is set to request or prototype, performance is affected as the bean generation frequency is high in DI container.
3. When scope is set to request or session, it cannot be used in non Web applications (for example, Batch application).

Creation of methods of Service class

Below are the points to be taken care of while writing methods of Service class.

- Creation of method of Service interface

```
public interface CartService {
    Cart createCart(); // (1) (2)
    Cart findCart(String cartId); // (1) (2)
}
```

- Creation of methods of Service class

```
@Service
@Transactional
public class CartServiceImpl implements CartService {

    @Inject
    CartRepository cartRepository;

    public Cart createCart() { // (1) (2)
        Cart cart = new Cart();
        // ...
        cartRepository.save(cart);
        return cart;
    }
```

```
    @Transactional(readOnly = true) // (3)
    public Cart findCart(String cartId) { // (1) (2)
        Cart cart = cartRepository.findByCartId(cartId);
        // ...
        return cart;
    }

}
```

Sr. No.	Description
(1)	**Create a method of Service class for each business logic.**
(2)	**Define methods in Service interface and implement business logic in its implementation class.**
(3)	**Add @Transactional annotation for changes to default transaction definition (class level annotation).** Attributes should be specified as per the requirement. Refer to *Information required for "Declarative transaction management"* for details. Moreover, to understand the points to be noted when using `@Transactional` annotation, it is advisable to confirm with "*Settings for using transaction management*".

Tip: Transaction definition of business logic for reference

When business logic for reference is to be implemented, by specifying `@Transactional(readOnly = true)`, instruction can be given to run the SQL under "Read-only transactions" for JDBC driver.

The way of handling read-only transactions depends on the implementation of JDBC driver; hence, confirm the specifications of JDBC driver to be used.

Note: Points to be noted when using "Read-only transactions"

If it is set to "perform health check" when retrieving a connection from connection pool, "Read-only transactions" may not be enabled. For details on this event and to avoid the same, refer to *About cases*

where "Read-only transactions" are not enabled.

Note: Transaction definition when a new transaction is required to be started

Set `@Transactional(propagation = Propagation.REQUIRES_NEW)` to start a new transaction without participating in the transaction of the caller method.

Regarding arguments and return values of methods of Service class

The below points must be considered for arguments and return values of methods of Service class.

Serializable classes (class implementing `java.io.Serializable`) must be used for arguments and return values of Service class.
Since there is possibility of Service class getting deployed as distributed application, it is recommended to allow only Serializable class.

Typical arguments and return values of the methods are as follows.

- Primitive types (`int`, `long`)
- Primitive wrapper classes (`java.lang.Integer`, `java.lang.Long`)
- java standard classes (`java.lang.String`, `java.util.Date`)
- Domain objects (Entity, DTO)
- Input/output objects (DTO)
- Collection (implementation class of `java.util.Collection`) of above types
- void
- etc ...

Note: Input/Output objects

1. Input object indicates the object that has all the input values required for executing Service method.
2. Output object indicates the object that has all the execution results (output values) of Service method.

   If business logic(BLogic class) is generated using "TERASOLUNA ViSC" then, input and output objects are used as argument and return value of the of BLogic class.

Values that are forbidden as arguments and return values are as follows.

- Objects (`javax.servlet.http.HttpServletRequest`, `javax.servlet.http.HttpServletResponse`, `javax.servlet.http.HttpSession`, `org.springframework.http.server.ServletServerHttpRequest`) which are dependent on implementation architecture of application layer (Servlet API or web layer API of Spring).
- Model(Form, DTO) classes of application layer
- Implementation classes of `java.util.Map`

Note: Reason for prohibition

1. If objects depending on implementation architecture of application layer are allowed, then application layer and domain layer get tightly coupled.
2. `java.util.Map` is too generalized. Using it for method arguments and return values makes it difficult to understand what type of object is stored inside it. Further, since the values are managed using keys, the following problems may occur.

- Values are mapped to a unique key and hence cannot be retrieved by specifying a key name which is different from the one specified at the time of inserting the value.
- When key name has to be changed, it becomes difficult to determine the impacted area.

How to sharing the same DTO between the application layer and domain layer is shown below.

- DTO belonging to the package of domain layer can be used in application layer.

Warning: Form and DTO of application layer should not be used in domain layer.

Implementation of SharedService class

Creation of SharedService class

Below are the points to be taken care of while creating SharedService class.
Only the points which are different from Service class are explained here.

1. **Add @Transactional annotation to class as and when required.**

   `@Transactional` annotation is not required when data access is not involved.

2. **Interface name should be XxxSharedService and class name should be XxxSharedServiceImpl.**
   Any other naming conventions can also be used. However, it is recommended to use distinguishable naming conventions for Service class and SharedService class.

Creation of SharedService class method

Below are the points to be taken care of while writing methods of SharedService class.
Only the points which are different from Service class are explained here.

1. **Methods in SharedService class must be created for each logic which is shared between multiple business logics.**
2. **Add @Transactional annotation to class as and when required.**
   Annotation is not required when data access is not involved.

Regarding arguments and return values of SharedService class method

Points are same as *Regarding arguments and return values of methods of Service class*.

Implementation of logic

Implementation in Service and SharedService is explained here.

Service and SharedService has implementation of logic related to operations such as data fetch, update, consistency check of business data and implementation related to business rules.

Example of a typical logic is explained below.

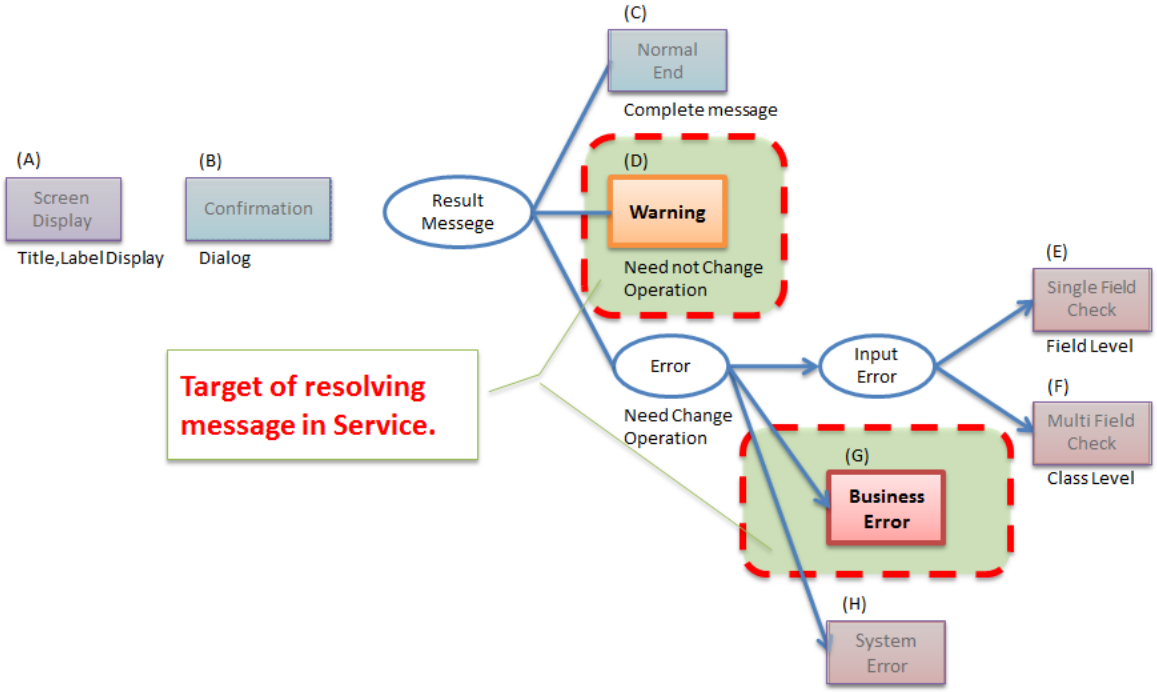
Operate on business data

Refer to the following for the examples of data (Entity) fetch and update.

- When using MyBatis3, *Database Access (MyBatis3)*
- When using JPA, *Database Access (JPA)*

Returning messages

Warning message and business error message are the two type of messages which must be resolved in Service (refer to the figure in red broken line below).
Other messages should be resolved in application layer.
Refer to *Message Management* for message types and message pattern.



Note: Regarding resolving message

In service, instead of the actual message **the information required for building the message (message code, message insert value) is resolved**.

Refer to the following for detailed implementation method.

- *Returning warning message*
- *Notifying business error*

Returning warning message

Message object must be returned for warning message. If domain object such as Entity needs to be returned with it,
message object and domain object should inserted into output object (DTO) and this output object must be returned.

Message object (`org.terasoluna.gfw.common.message.ResultMessages`) is provided as common library. When the class provided in common library
does not fulfill the requirements, message object should be created for each project.

- Creation of DTO

```
public class OrderResult implements Serializable {
    private ResultMessages warnMessages;
    private Order order;

    // omitted

}
```

- Implementation of method of Service class

  Following is an example of implementation of displaying a warning message. The message is "Products may not be delivered together since the order includes products which are not available right now".

```
public OrderResult submitOrder(Order order) {

    // omitted

    boolean hasOrderProduct = orderRepository.existsByOrderProduct(order); //␣
↪(1)

    // omitted

    Order order = orderRepository.save(order);

    // omitted

    ResultMessages warnMessages = null;
    // (2)
    if(hasOrderProduct) {
        warnMessages = ResultMessages.warn().add("w.xx.xx.0001");
    }
    // (3)
    OrderResult orderResult = new OrderResult();
    orderResult.setOrder(order);
    orderResult.setWarnMessages(warnMessages);
    return orderResult;
}
```

Sr. No.	Description
(1)	When the order includes products which are not available right now, set `hasOrderProduct` to `true`.
(2)	In the above example, when the order includes products which are not available right now, a warning message occurs.
(3)	In the above example, the registered `Order` object and warning message are returned by storing objects in a DTO called `OrderResult`.

Notifying business error

Business exception is thrown when business rules are violated while executing business logic.
The following can be the cases.

- When reservation date exceeds deadline while making tour reservation
- When the product is out of stock at the time of placing an order
- etc ...

Business exception (`org.terasoluna.gfw.common.exception.BusinessException`) is provided as common library.
When business exception class provided in common library does not fulfill the requirements, business exception class should be created in the project.
It is recommended to create business exception class as subclass of java.lang.RuntimeException.

> **Note: Reason for considering business exception as an unchecked exception**
>
> Since business exceptions need to be handled in controller class, they can be configured as checked exception. However in this guideline, it is recommended that business exception be subclass of unchecked exception (`java.lang.RuntimeException`). By default, if there is a RuntimeException, transaction will be rolled back. Hence, doing this will prevent leaving a bug in the source-code due to inadequate settings of @Transactional annotation. Obviously, if settings are changed such that transaction rollbacks

even in case checked exceptions, business exception can be configured as subclass of checked exceptions.

Example of throwing business exception.

Below example notifies that reservation is past the deadline and a business error.

```
// omitted

if(currentDate.after(reservationLimitDate)) { // (1)
    throw new BusinessException(ResultMessages.error().add("e.xx.xx.0001"));
}

// omitted
```

Sr. No.	Description
(1)	Business exception is thrown since reservation date is past the deadline at the time of making reservation.

Refer to *Exception Handling* for details of entire exception handling.

Notifying system error

System exception is thrown when error occurs in system while executing business logic.

The following can be the cases.

- When master data, directories and files that should already exist, do not exist
- When a checked exception generated by a library method is caught and this exception indicates abnormal system state.
- etc ...

System exception (`org.terasoluna.gfw.common.exception.SystemException`) is provided as common library.

When system exception class provided in common library does not fulfill the requirements, system exception class should be created in the project.

It is recommended to create system exception class as subclass of java.lang.RuntimeException.

The reason is system exception should not be handled by application code and rollback target of `@Transactinal` annotation is set to `java.lang.RuntimeException` by default.

Example of throwing system exception.

Example notifying the non-existence of the specific product in product master as system error is shown below.

```
ItemMaster itemMaster = itemMasterRepository.findOne(itemCode);
if(itemMaster == null) { // (1)
    throw new SystemException("e.xx.fw.0001",
        "Item master data is not found. item code is " + itemCode + ".");
}
```

Sr. No.	Description
(1)	System exception is thrown since master data that should already exist does not exist. Example of case when system error is detected in logic）

Example that throws system exception while catching IO exception while copying the file is shown below.

```
// ...

try {
    FileUtils.copy(srcFile, destFile);
} catch(IOException e) { // (1)
    throw new SystemException("e.xx.fw.0002",
        "Failed file copy. src file '" + srcFile + "' dest file '" + destFile␣
↪+ "'.", e);
}
```

Sr. No.	Description
(1)	System exception that is classified into invalid system state is thrown by the library method. **The exception generated by library must be passed to system exception class as cause exception.** If cause exception is lost, error occurrence location and basic error cause cannot be traced from the stack trace.

Note: Regarding handling of data access error

When data access error occurs in Repository and O/R Mapper while executing business logic, it is converted to subclass of `org.springframework.dao.DataAccessException` and thrown. Error can be handled in application layer instead of catching in business logic. However, some errors like unique constraints violation error should be handled in business logic as per business requirements. Refer to *Database Access (Common)* for details.

3.2.6 Regarding transaction management

Transaction management is required in the logic where data consistency must be ensured.

Method of transaction management

There are various transaction management methods. However, in this guideline, **it is recommended to use "Declarative Transaction Management" provided by Spring Framework.**

Declarative transaction management

In "Declarative transaction management", the information required for transaction management can be declared by the following 2 methods.

- Declaration in XML(bean definition file).
- **Declaration using annotation (@Transactional) (Recommended).**

Refer to Spring Reference Document -Transaction Management(Declarative transaction management)- for the details on "Declarative type transaction management" provided by Spring Framework.

Note: Reason for recommending annotation method

1. The transaction management to be performed can be understood by just looking at the source code.
2. AOP settings for transaction management is not required if annotations are used and so XML becomes simple.

Information required for "Declarative transaction management"

Specify `@Transactional` annotation for at class level or method level which are considered as target of transaction management and specify the information required for
transaction control in attributes of `@Transactional` annotation.

Note: In this guideline, it is a prerequisite to use `@org.springframework.transaction.annotation.Transac` annotation provided by Spring Framework.

Tip: From Spring 4, it is possible to use `@javax.transaction.Transactional` annotation added from JTA 1.2.

However, in this guideline, it is recommended to use an annotation of Spring Framework that can specify the information required for "Declarative transaction management" in a much more detailed way.

Following attributes can be specified if Spring Framework annotation is used.

- `NESTED`(JDBC savepoint) as an attribute value of the propagation method of transaction (`propagation` attribute).
- Isolation level of transaction (`isolation` attribute)
- Timeout period of transaction (`timeout` attribute)
- Read-only flag of transaction (`readOnly` attribute)

Sr. No.	Attribute name	Description
1	propagation	Specify transaction propagation method. **[REQUIRED]** Starts transaction if not started. (default when omitted) **[REQUIRES_NEW]** Always starts a new transaction. **[SUPPORTS]** Uses transaction if started. Does not use if not started. **[NOT_SUPPORTED]** Does not use transaction. **[MANDATORY]** Transaction should start. An exception occurs if not started. **[NEVER]** Does not use transaction (never start). An exception occurs if started. **[NESTED]** save points are set. They are valid only in JDBC.

次のページに続く

Table. 3.10 – 前のページからの続き

Sr. No.	Attribute name	Description
2	isolation	Specify isolation level of transaction. Since this setting depends on DB specifications, settings should be decided by checking DB specifications. **[DEFAULT]** Isolation level provided by DB by default.(default when omitted) **[READ_UNCOMMITTED]** Reads (uncommitted) data modified in other transactions. **[READ_COMMITTED]** Does not read (uncommitted) data modified in other transactions. **[REPEATABLE_READ]** Data read by other transactions cannot be updated. **[SERIALIZABLE]** Isolates transactions completely. Isolation level of transaction is considered as the parameter related to exclusion control. Refer to *Exclusive Control* for exclusion control.
3	timeout	Specify timeout of transaction (seconds). -1 by default (Depends on specifications and settings of DB to be used)
4	readOnly	Specify Read-only flag of transaction. false by default (Not read-only)
5	rollbackFor	Specify list of exception classes to rollback transactions. Blank by default (Not specified)

次のページに続く

Table. 3.10 – 前のページからの続き

Sr. No.	Attribute name	Description
6	rollbackForClassName	Specify list of exception class names to rollback transactions. Blank by default (Not specified)
7	noRollbackFor	Specify list of exception classes to commit transactions. Blank by default (Not specified)
8	noRollbackForClassName	Specify list of exception classes to commit transactions. Blank by default (Not specified)

Note: Location to specify the @Transactional annotation

It is recommended to specify the annotation at the class level or method level of the class. Must be noted that it should not interface or method of interface. For the reason, refer to 2nd Tips of Spring Reference Document -Transaction Management(Using @Transactional)-.

Warning: Default operations of rollback and commit when exception occurs

When rollbackFor and noRollbackFor is not specified, Spring Framework performs the following operations.

- Rollback when unchecked exception of (java.lang.RuntimeException and java.lang.Error) class or its subclass occurs.
- Commit when checked exception of (java.lang.Exception) class or its subclass occurs. **(Necessary to note)**

Note: Regarding value attributes of @Transactional annotation

There is a value attribute in `@Transactional` annotation. However, this attribute specifies which Transaction Manager to be used when multiple Transaction Managers are declared. It is not required to specify when there is only one Transaction Manager. When multiple Transaction Managers need to be used, refer to Spring Reference Document -Transaction Management(Multiple Transaction Managers with @Transactional)-.

Note: Default isolation levels of main DB are given below.

Default isolation levels of main DB are given below.

- Oracle : READ_COMMITTED
- DB2 : READ_COMMITTED
- PostgreSQL : READ_COMMITTED
- SQL Server : READ_COMMITTED
- MySQL : REPEATABLE_READ

Note: Cases where "Read-only transactions" are not enabled

A mechanism is provided to run SQL under "Read-only transactions" by specifying `readOnly = true`; however, when all of the following conditions are satisfied, there will be a JDBC driver where "Read-only transactions" are not enabled.

[Conditions to generate this event]

- Perform health check when retrieving a connection from connection pool.
- Disable 'Auto commit' of connection retrieved from connection pool.
- Use `DataSourceTransactionManager` or `JpaTransactionManager` as `PlatformTransactionManager`. (This event does not occur when using `JtaTransactionManager`).

[JDBC driver where occurrence of this event is confirmed]

- `org.postgresql:postgresql:9.3-1102-jdbc41` (JDBC 4.1 compatible JDBC driver for PostgreSQL 9.3)

[Method to avoid this event]

In a case where "Read-only transactions" are not enabled, if `readOnly = true` is specified, it ends up carrying out the unnecessary processes. Therefore, it is recommended to execute the SQL under "Updatable transactions" even for the reference processes.

Other methods to avoid this event are as follows:

- Do not perform health check when retrieving a connection from connection pool.
- Enable 'Auto commit' of the connection retrieved from connection pool. (Disable 'Auto commit' only when transaction management is required)

However, However, do not change the design for 'health check' and 'auto commit' to avoid this event.

[Remarks]

- Reproduction of this event is confirmed on PostgreSQL 9.3 and Oracle 12c. It is not performed on any other database and versions.
- In PostgreSQL 9.3, `SQLException` occurs when `java.sql.Connection#setReadOnly(boolean)` method is called.
- When SQL or API call of JDBC is logged in using *log4jdbc*, `SQLException` occurred from JDBC driver is output to log with ERROR level.
- **SQL Exception occurred from JDBC driver is ignored by exception handling of Spring Framework. Hence, even though it is not an error as application behavior, the "Read-only transactions" is not enabled.**
- Occurrence of this event is not confirmed in Oracle 12c.

[Reference]

When following log is output using *log4jdbc*, it will be treated as a case corresponding to this event.

```
date:2015-02-20 16:11:56        thread:main     user:   X-Track:
↪level:ERROR     logger:jdbc.audit
↪message:3. Connection.setReadOnly(true)
org.postgresql.util.PSQLException: Cannot change transaction read-only
↪property in the middle of a transaction.
    at org.postgresql.jdbc2.AbstractJdbc2Connection.
↪setReadOnly(AbstractJdbc2Connection.java:741) ~[postgresql-9.3-1102-
↪jdbc41.jar:na]
    ...
```

Note: timeout attribute of @Transactional annotation

Although `@Transactional`annotation consists of `timeout`attribute, value specified in `timeout`attribute is ignored in MyBatis3.3 and MyBatis-Spring 1.2 combination and is not used.

Propagation of transaction

In most of the cases, Propagation method of transaction is "REQUIRED".

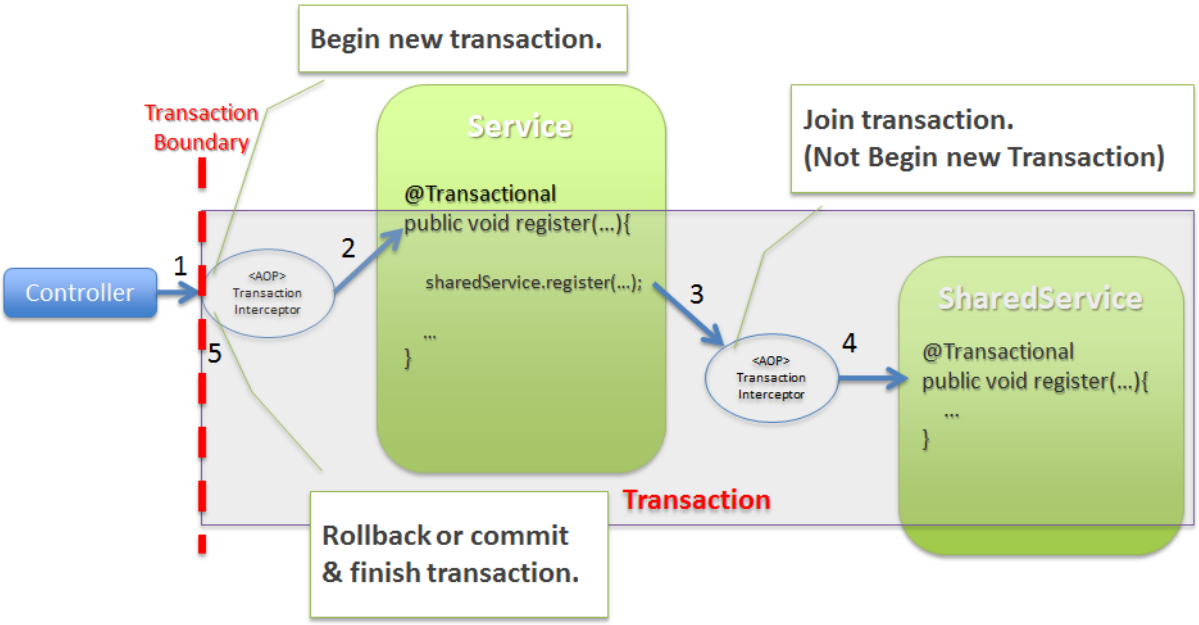
However, since **"REQUIRES_NEW" is also used according to the requirements of Application**, transaction control flow in case of "REQUIRED" and "REQUIRES_NEW" is explained below.

The explanation of other propagation methods is omitted in this guideline since their usage frequency is very low.

Transaction control flow when propagation method of transaction is set to "REQUIRED"

When propagation method of transaction is set to "REQUIRED", all sequential processes called from controller are processed in the same transaction.

1. Controller calls a method of Service class. At this time, since started transaction does not exist, transaction is started using `TransactionInterceptor`.
2. `TransactionInterceptor` calls the method of service class after starting the transaction.
3. Service class calls a method of `SharedService`. This method is also under transaction control. At this time, though started transaction exists, `TransactionInterceptor` participates in the started transaction without starting a new transaction.



4. `TransactionInterceptor` calls the method under transaction control after participating in the started transaction.

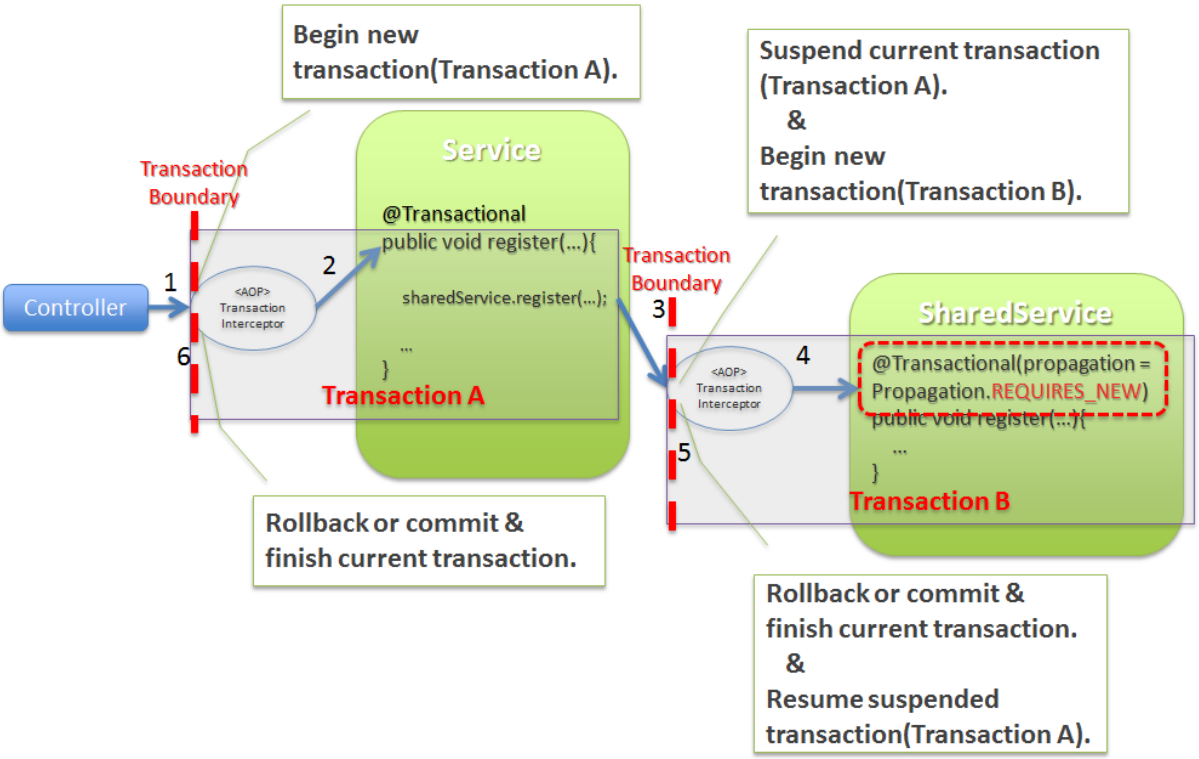
5. `TransactionInterceptor` performs commit or rollback according to result of processing and ends the transaction.

Note: Reason for occurrence of org.springframework.transaction.UnexpectedRollbackException

When propagation method of transaction is set to "REQUIRED", though there is only one physical transaction, internally Spring Framework creates transaction boundaries. In case of above example, when a method of SharedService is called, a TransactionInterceptor is started which internally provides transaction control boundary at SharedService level. Therefore, when an exception (which is set as target of rollback) occurs in `SharedService` method, status of transaction is set to rollback (rollback-only) by `TransactionInterceptor`. This transaction now cannot be committed. Going further, if the Service method tries to commit this transaction due to conflicting settings of rollback target exception between Service method and SharedShared method, `UnexpectedRollbackException` is generated by Spring Framework notifying that there is inconsistency in transaction control settings. When UnexpectedRollbackException is generated, it should be checked that there is no inconsistency in rollbackFor and noRollbackFor settings.

Transaction management flow when propagation method of transaction is set to "REQUIRES_NEW"

When propagation method of transaction is set to "REQUIRES_NEW", a part of the sequence of processing (Processing done in SharedService) are processed in another transaction when called from Controller.



1. Controller calls a method of Service class. This method is under transaction control. At this time, since started transaction does not exist, transaction is started by `TransactionInterceptor` (Hereafter, the started transaction is referred as "Transaction A").

2. `TransactionInterceptor` calls the method of service class after transaction (Transaction A) is started.

3. Service class calls a method of `SharedService` class. At this time, though started transaction (Transaction A) exists, since propagation method of transaction is "REQUIRES_NEW", new transaction is started by `TransactionInterceptor`. (Hereafter, started transaction is referred as "Transaction B"). At this time, "Transaction A" is interrupted and the status changes to 'Awaiting to resume'.

4. `TransactionInterceptor` calls the method of SharedService class after Transaction B is started.

5. `TransactionInterceptor` performs commit or rollback according to the process result and ends the Transaction B. At this time, "Transaction A" is resumed and status is changed to Active.

6. `TransactionInterceptor` performs commit or rollback according to the process result and ends the Transaction A.

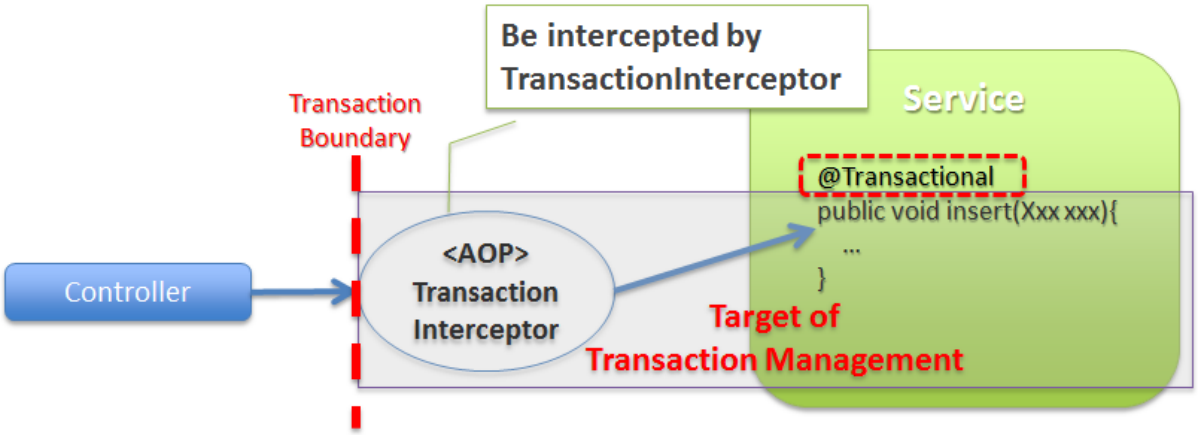
Way of calling the method which is under transaction control

Since “Declarative transaction management” provided by Spring Framework is implemented using AOP, transaction management is applied only for method calls for which AOP is enabled.

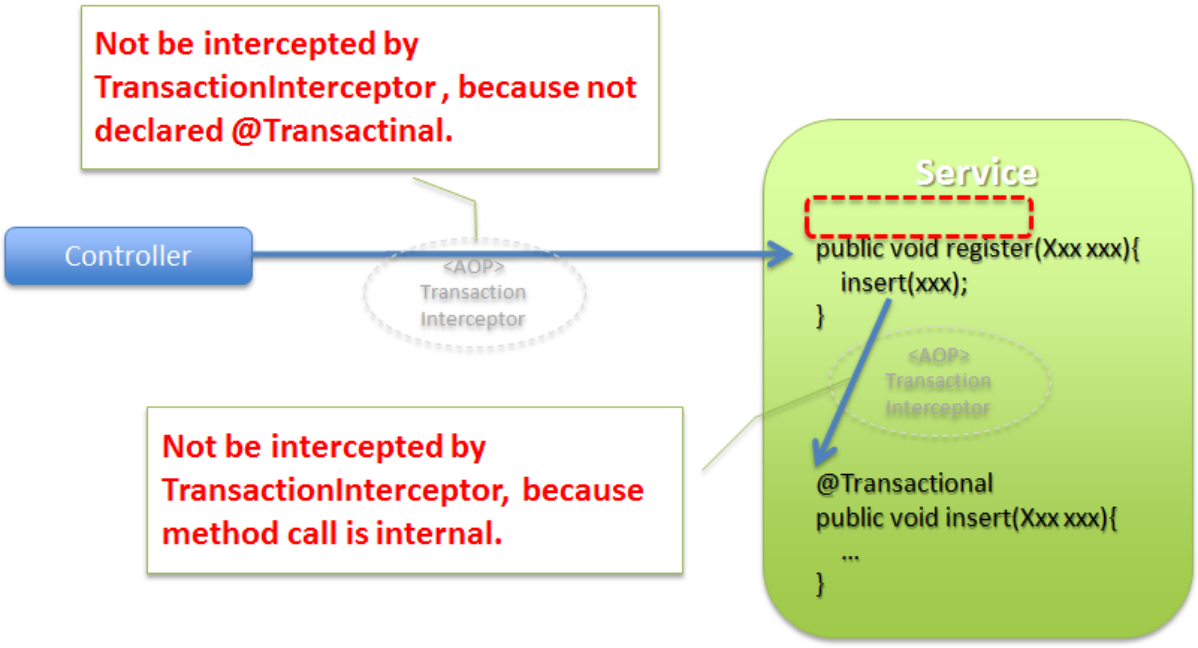
Since the default mode of AOP is **“proxy” mode, transaction control will be applied only when public method is called from another class.**

Note that **transaction control is not applied if the target method is called from an internal method even if the target method is a public method**.

- **Way of calling the method which is under transaction control**



- **Way of calling the method which is not under transaction control**



Note: In order to bring internal method calls under transaction control

It is possible to enable transaction control for internal method calls as well by setting the AOP mode to `"aspectj"`. However, if internal method call of transaction management is enabled, the route of transaction management may become complicated; hence it is recommended to use the default "proxy" for AOP mode.

Settings for using transaction management

The settings required for using transaction management are explained.

PlatformTransactionManager settings

In order to have transaction management done, it is necessary to define the bean of `PlatformTransactionManager`.

Spring Framework has provided couple of classes based on the purpose; any class can be specified that meets the requirement of the application.

- `xxx-env.xml`

  Example of settings for managing the transaction using JDBC connection which is fetched from DataSource is given below.

```
<!-- (1) -->
<bean id="transactionManager"
      class="org.springframework.jdbc.datasource.DataSourceTransactionManager
 ↪">
    <property name="dataSource" ref="dataSource" />
    <property name="rollbackOnCommitFailure" value="true" />
</bean>
```

Sr. No.	Description
(1)	Specify the implementation class of PlatformTransactionManager. It is recommended to set id as "transactionManager".

Note: When transaction management (Global transaction management) is required for multiple DBs (Multiple resources)

- It is necessary to use `org.springframework.transaction.jta.JtaTransactionManager` and manage transactions by using JTA functionality provided by application server.

- When JTA is to be used in WebSphere and Oracle WebLogic Server, a `JtaTransactionManager` which is extended for the application server is automatically set by specifying <tx:jta-transaction-manager/>.

Table.3.11 **Implementation class of PlatformTransactionManager provided by Spring Framework**

Sr. No.	Class name	Description
1.	org.springframework.jdbc.datasource. DataSourceTransactionManager	Implementation class for managing the transaction by calling API of JDBC(`java.sql.Connection`). Use this class when MyBatis and `JdbcTemplate` is to be used.
2.	org.springframework.orm.jpa. JpaTransactionManager	Implementation class for managing the transaction by calling API of JPA(`javax.persistence.EntityTransaction`). Use this class when JPA is to be used.
3.	org.springframework.transaction.jta. JtaTransactionManager	Implementation class for managing the transaction by calling API of JTA(`javax.transaction.UserTransaction`). Use this class to manage transaction with resources (Database/Messaging service/General-purpose EIS(Enterprise Information System) etc.) using JTS (Java Transaction Service) provided by application server. When it is necessary to execute the operations with multiple resources in a single transaction, it is necessary to use JTA for managing transactions.

Settings for enabling @Transactional

In this guideline, it is recommended to manage transaction by using "Declarative transaction management" where `@Transactional` annotation is used.

Here, the settings required for using `@Transactional` annotation are explained.

- xxx-domain.xml

```
<tx:annotation-driven /> <!-- (1) -->
```

Sr. No.	Description
(1)	With use of <tx:annotation-driven> element in XML (bean definition file), transaction control gets enabled at the locations where `@Transactional` annotation is used.

Note: Regarding drawbacks of transaction management

There is a description regarding "Understanding drawbacks of transaction" in IBM DeveloperWorks. Read this article about pitfalls of transaction management and pitfalls while using @Transactional of Spring Framework. Refer to Article on IBM DeveloperWorksfor details.

Since the article of IBM DeveloperWorks is an old article (year 2009), some of the content is different than the behavior when using Spring Framework 4.1.

Specifically, the contents of "Listing 7. Using read-only with REQUIRED propagation mode - JPA".

From Spring Framework 4.1, when Hibernate ORM 4.2 or higher version is used as JPA provider, it has been improved so that instruction can be given to run the SQL under "Read-only transactions" for JDBC driver (SPR-8959).

The way of handling read-only transactions depends on the implementation of JDBC driver; hence, confirm the specifications of JDBC driver to be used.

Note: Programmatic transaction management

In this guideline, "Declarative transaction management" is recommended. However, programmatic transaction management is also possible. Refer to Spring Reference Document -Transaction Management(Programmatic transaction management)- for details.

Regarding attributes of <tx:annotation-driven> element

Various attributes can be specified in <tx:annotation-driven> and default behavior can be customized.

- xxx-domain.xml

```
<tx:annotation-driven
    transaction-manager="txManager"
    mode="aspectj"
    proxy-target-class="true"
    order="0" />
```

Sr. No.	Attribute	Description
1	transaction-manager	Specify `PlatformTransactionManager` bean. When omitted, bean registered with the name “transactionManager” is used.
2	mode	Specify AOP mode. When omitted, `"proxy"` is the default value. `"aspectj"` can be also be specified. It is recommended to use `"proxy"`.
3	proxy-target-class	Flag to specify whether proxy target is limited to class (this is valid only in case of mode=”proxy”). When omitted, it will be “false”. • In case of false, if the target class has an interface, proxy is done using dynamic proxies functionality of standard JDK. When there is no interface, proxy is done using GCLIB functionality. • In case of true, proxy is done using GCLIB function irrespective of whether interface is available or not.
4	order	Order of Advice of AOP (Priority). When omitted, it will be “Last (Lowest priority)”.

3.2.7 Tips

Method of dealing with violation of business rules as field error

When it is necessary to output the error of business rules for each field, the mechanism of (Bean Validation or Spring Validator) on the Controller side should be used.

In this case, It is recommended to implement check logic as Service class and then to call the method of Service class from Bean Validation or Spring Validator.

Refer to *Input Validation* business logic approach for details.

3.3 Implementation of Infrastructure Layer

Implementing RepositoryImpl is carried out in infrastructure layer.

RepositoryImpl implements the method defined in Repository interface.

3.3.1 Implementing RepositoryImpl

Methods to create a Repository for relational database using MyBatis3 and JPA are introduced below.

- *Implementing Repository using MyBatis3*
- *Implementing Repository using JPA*

Implementing Repository using MyBatis3

When MyBatis3 is to be used as persistence API with relational database, RepositoryImpl need not be implemented, if Repository interface is created using "*Mapper interface mechanism*" provided by MyBatis3.

This is because it is a mechanism where MyBatis3 automatically maps the method of Mapper interface and the statement (SQL) to be called.

When using MyBatis3, an application developer creates:

- Repository interface (method definition)
- Mapping file (SQL and O/R mapping definition)

An example of creating Repository interface and mapping file is given below.

For details on how to use MyBatis3, refer to: *Database Access (MyBatis3)*.

- An example of creating Repository interface (Mapper interface)

```
package com.example.domain.repository.todo;

import com.example.domain.model.Todo;

// (1)
public interface TodoRepository {
    // (2)
    Todo findOne(String todoId);
}
```

Sr. No.	Description
(1)	Create as an interface of POJO. It is not necessary to specify MyBatis3 interface, annotation, etc.
(2)	Define a method of Repository. Basically it is not necessary to assign MyBatis3 annotation; however, annotation may also be specified in some cases.

- An example of creating mapping file

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org/DTD Mapper 3.0//EN"
    "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<!-- (3) -->
<mapper namespace="com.example.domain.repository.todo.TodoRepository">

    <!-- (4) -->
    <select id="findOne" parameterType="string" resultMap="todoResultMap">
      SELECT
          todo_id,
          title,
          finished
      FROM
          t_todo
      WHERE
          todo_id = #{todoId}
    </select>

    <!-- (5) -->
    <resultMap id="todoResultMap" type="Todo">
        <result column="todo_id" property="todoId" />
        <result column="title" property="title" />
        <result column="finished" property="finished" />
    </resultMap>

</mapper>
```

Sr. No.	Description
(3)	Create a mapping file for each Repository interface. Specify FQCN (Fully Qualified Class Name) of Repository interface in namespace of mapping file (`namespace` attribute of `mapper` element).
(4)	Define statement (SQL) to be run for each method defined in Repository interface. Specify a method name of Repository interface in statement ID of each statement element (`id` attribute of `select`/`insert`/`update`/`delete` element).
(5)	When a query is to be raised, define O/R mapping as required. Auto mapping can be used for simple O/R mapping; however, individual mapping definition is needed for complex O/R mapping. In the above example, auto mapping can also be used for mapping definition as it is simple O/R mapping.

Implementing Repository using JPA

When JPA is to be used as persistence API with relational database, Repository can be very easily created if `org.springframework.data.jpa.repository.JpaRepository` of Spring Data JPA is used.

For details on how to use Spring Data JPA, refer to *Database Access (JPA)*.

When Spring Data JPA is used, only an interface with inherited JpaRepository is required to be created for basic CRUD operations. In other words, RepositoryImpl is not required.

However, RepositoryImpl is needed for using dynamic query (JPQL).

Refer to *Database Access (JPA)* for implementing RepositoryImpl when using Spring Data JPA.

- TodoRepository.java

```
public interface TodoRepository extends JpaRepository<Todo, String> { // (1)
    // ...
}
```

Sr. No.	Description
(1)	Only by defining the interface that inherits JpaRepository, basic CRUD operations for Todo entity can be performed without being implemented.

Describe the case to add operations which are not provided by JpaRepository.

When Spring Data JPA is used, if it is a static query, it is advisable to add a method to the interface and to specify the query (JPQL) to be executed when that method is called, using the annotation.

- TodoRepository.java

```
public interface TodoRepository extends JpaRepository<Todo, String> {
    @Query("SELECT COUNT(t) FROM Todo t WHERE finished = :finished") // (1)
    long countByFinished(@Param("finished") boolean finished);
    // ...
}
```

Sr. no.	Description
(1)	Specify a query (JPQL) using `@Query` annotation.

Implementing Repository to link with external system using RestTemplate

---

Todo

TBD

Details will be provided in the next version.

3.4 Implementation of Application Layer

This chapter explains implementation of application layer of a web application that uses HTML forms.

Note: Refer to the following page for the implementation required for Ajax development and REST API.

- *Ajax*

Implementation of application layer is given in the following 3 steps.

1. *Implementing Controller*

   Controller receives the request, calls business logic, updates model, decides View. Thereby, controls one complete round of operations after receiving the request.

   It is the most important part in the implementation of application layer.

2. *Implementing form object*

   Form object transfers the values between HTML form and application.

3. *Implementing View*

   View (JSP) acquires the data from model (form object, domain object etc.) and generates screen (HTML).

3.4.1 Implementing Controller

Implementation of Controller is explained below.

The Controller performs the following roles.

1. **Provides a method to receive the request.**

   Receives requests by implementing methods to which `@RequestMapping` annotation is assigned.

2. **Performs input validation of request parameter.**

   For request which requires input validation of request parameters, it can be performed by specifying `@Validated` annotation to form object argument of method which receives the request.

   Perform single item check using Bean Validation and correlation check using Spring Validator or Bean Validation.

3. **Calls business logic.**

Controller does not implement business logic but delegates by calling Service method.

4. **Reflects the output of business logic to Model.**

   The output can be referred in View by reflecting domain object returned from Service method to `Model`.

5. **Returns View name corresponding to business logic output.**

   Controller does not implement the logic of rendering the view. Rendering is done in View technologies like JSP etc.

   Controller only returns the name of the view in which the rendering logic is implemented.

   The View corresponding to view name is resolved by `ViewResolver` provided by Spring Framework.

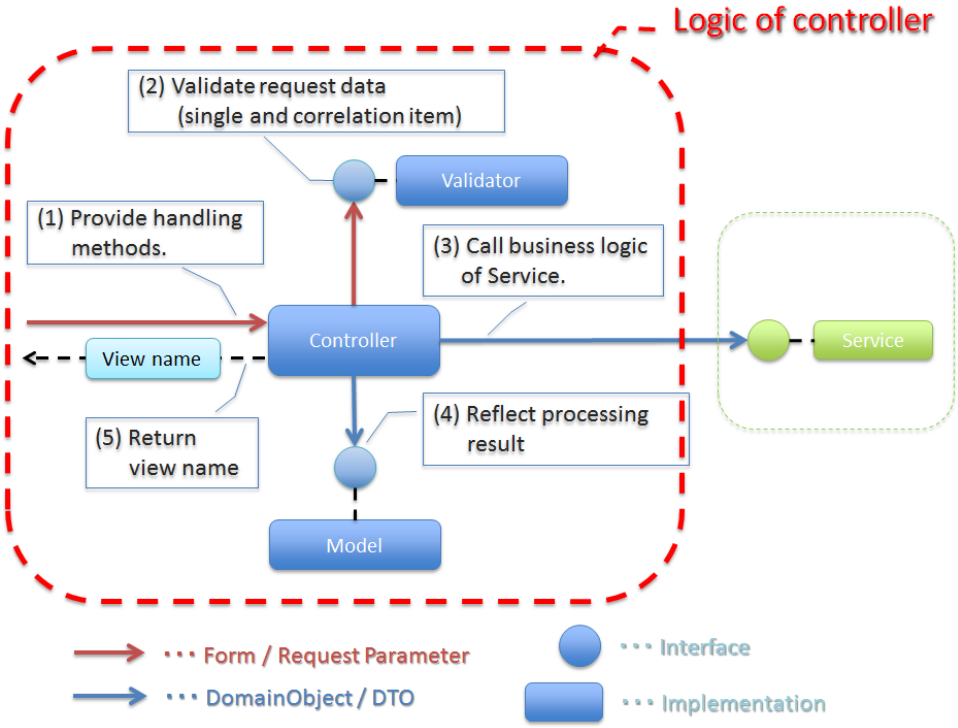


Figure.3.1 **Picture - Logic of controller**

Note: **It is recommended that controller implements only the routing logic** such as calling business logic, reflecting output of the business logic to `Model`, deciding the View name is implemented in the Controller.

The implementation of Controller is explained by focusing on the following points.

- *Creating Controller class*

- *Mapping request and handler method*
- *Regarding arguments of handler method*
- *Regarding return value of handler method*

Creating Controller class

Controller class is created with @Controller annotation added to POJO class (Annotation-based Controller).

Controller in Spring MVC can also be created by implementing `org.springframework.web.servlet.mvc.Controller` interface (Interface-based Controller). However, it is preferred to avoid using it as it is Deprecated from Spring 3 onwards.

```
@Controller
public class SampleController {
    // ...
}
```

Mapping request and handler method

`@RequestMapping` annotation is assigned to the method that receives request.

In this document, the method to which `@RequestMapping` is added, is called as “handler method”.

```
@RequestMapping(value = "hello")
public String hello() {
    // ...
}
```

The rules for mapping the incoming request with a handler method can be specifying as attributes of `@RequestMapping` annotation.

Sr. No.	Attribute name	Description
1.	value	Specify request path which needs to be mapped (multiple values allowed).
2.	method	Specify HTTP method (`RequestMethod` type) which needs to be mapped (multiple methods allowed). GET/POST are mainly used for mapping requests from HTML form, while other HTTP methods (such as PUT/DELETE) are used for mapping requests from REST APIs as well.
3.	params	Specify request parameters which need to be mapped (multiple parameters allowed). Request parameters are mainly used for mapping request from HTML form. If this mapping method is used, the case of mapping multiple buttons on HTML page can be implemented easily.
4.	headers	Specify request headers which need to be mapped (multiple headers allowed). Mainly used while mapping REST API and Ajax requests.
5.	consumes	Mapping can be performed using Content-Type header of request. Specify media type which needs to be mapped (multiple types allowed). Mainly used while mapping REST API and Ajax requests.
6.	produces	Mapping can be performed using Accept header of request. Specify media type which needs to be mapped (multiple types allowed). Mainly used while mapping REST API and Ajax requests.

Note: Combination of mapping

Complex mapping can be performed by combining multiple attributes, but considering maintainability,

mapping should be defined and designed in the simplest way possible . It is recommended to consider combining 2 attributes (value attribute and any other 1 attribute).

6 examples of mapping are shown below.

- *Mapping with request path*
- *Mapping by HTTP method*
- *Mapping by request parameter*
- *Mapping using request header*
- *Mapping using Content-Type header*
- *Mapping using Accept header*

In the following explanation, it is prerequisite to define the handler method in the Controller class.

```
@Controller // (1)
@RequestMapping("sample") // (2)
public class SampleController {
    // ...
}
```

Sr. No.	Description
(1)	With `@Controller`, it is recognized as Annotation-based controller class and becomes the target of component scan.
(2)	All the handler methods in this class are mapped to URLs with "sample" by adding `@RequestMapping("sample")` annotation at class level.

Mapping with request path

In case of the following definition, if the URL "sample/hello" is accessed, then hello() method is executed.

```
@RequestMapping(value = "hello")
public String hello() {
```

When multiple values are specified, it is handled by 'OR' condition.

In case of following definition, if "sample/hello" or "sample/bonjour" is accessed, then hello() method is executed.

```
@RequestMapping(value = {"hello", "bonjour"})
public String hello() {
```

Pattern can be specified instead of a specific value for request path. For details of specifying patterns, refer to reference documentation of Spring Framework.

- URI Template Patterns
- URI Template Patterns with Regular Expressions
- Path Patterns
- Patterns with Placeholders

Mapping by HTTP method

In case of the following definition, if the URL "sample/hello" is accessed with POST method, then hello() method is executed. For the list of supported HTTP methods, refer to Javadoc of RequestMethod. When not specified, all supported HTTP methods are mapped.

```
@RequestMapping(value = "hello", method = RequestMethod.POST)
public String hello() {
```

When multiple values are specified, it is handled by 'OR' condition.

In case of following definition, if "sample/hello" is accessed with GET or HEAD method, then hello() method is executed.

```
@RequestMapping(value = "hello", method = {RequestMethod.GET, RequestMethod.
↪HEAD})
public String hello() {
```

Mapping by request parameter

In case of following definition, if the URL "sample/hello?form" is accessed, then hello() method is executed.

When request is sent as a POST request, request parameters may exist in request body even if they do not exist in URL.

```
@RequestMapping(value = "hello", params = "form")
public String hello() {
```

When multiple values are specified, it is handled by 'AND' condition.

In case of following definition, if the URL "sample/hello?form&formType=foo" is accessed, then hello() method is executed.

```
@RequestMapping(value = "hello", params = {"form", "formType=foo"})
public String hello(@RequestParam("formType") String formType) {
```

Supported formats are as follows.

Sr. No.	Format	Explanation
1.	paramName	Mapping is performed when request parameter of the specified parameName exists.
2.	!paramName	Mapping is performed when request parameter of the specified parameName does not exist.
3.	paramName=paramValue	Mapping is performed when value of the specified parameName is paramValue.
4.	paramName!=paramValue	Mapping is performed when value of the specified parameName is not paramValue.

Mapping using request header

Refer to the details on the following page to mainly use the controller to map REST API and Ajax requests.

- *Ajax*

Mapping using Content-Type header

Refer to the details on the following page to mainly use the controller to map REST API and Ajax requests.

- *Ajax*

Mapping using Accept header

Refer to the details on the following page to mainly use the controller to map REST API and Ajax requests.

- *Ajax*

Mapping request and handler method

Mapping by the following method is recommended.

- **Grouping of URL of request is done for each unit of business flow or functional flow.**

URL grouping means defining `@RequestMapping(value = "xxx")` as class level annotation.

- **Use the same URL for requests for screen transitions within same functional flow**

  The same URL means the value of 'value' attribute of `@RequestMapping(value = "xxx")` must be same.

  Determining which handler method is used for a particular request with same functional flow is performed using HTTP method and HTTP parameters.

The following is an example of mapping between incoming request and handler method by a sample application with basic screen flow.

- *Overview of sample application*
- *Request URL*
- *Mapping request and handler method*
- *Implementing form display*
- *Implementing the display of user input confirmation screen*
- *Implementing 'redisplay of form'*
- *Implementing 'create new user' business logic*

Overview of sample application

Functional overview of sample application is as follows.

- Provides functionality of performing CRUD operations of Entity.
- Following 5 operations are provided.

Sr. No.	Operation name	Overview
1.	Fetching list of Entities	Fetch list of all the created Entities to be displayed on the list screen.
2.	Create Entity	Create a new Entity with the specified contents. Screen flow (form screen, confirmation screen, completion screen) exists for this process.
3.	Fetching details of Entity	Fetch Entity of specified ID to be displayed on the details screen.
4.	Entity update	Update Entity of specified ID. Screen flow (form screen, confirmation screen, completion screen) exists for this process.
5.	Entity delete	Delete Entity of specified ID.

- Screen flow of all functions is as follows.

  It is not mentioned in screen flow diagram however, when input validation error occurs, form screen is displayed again.

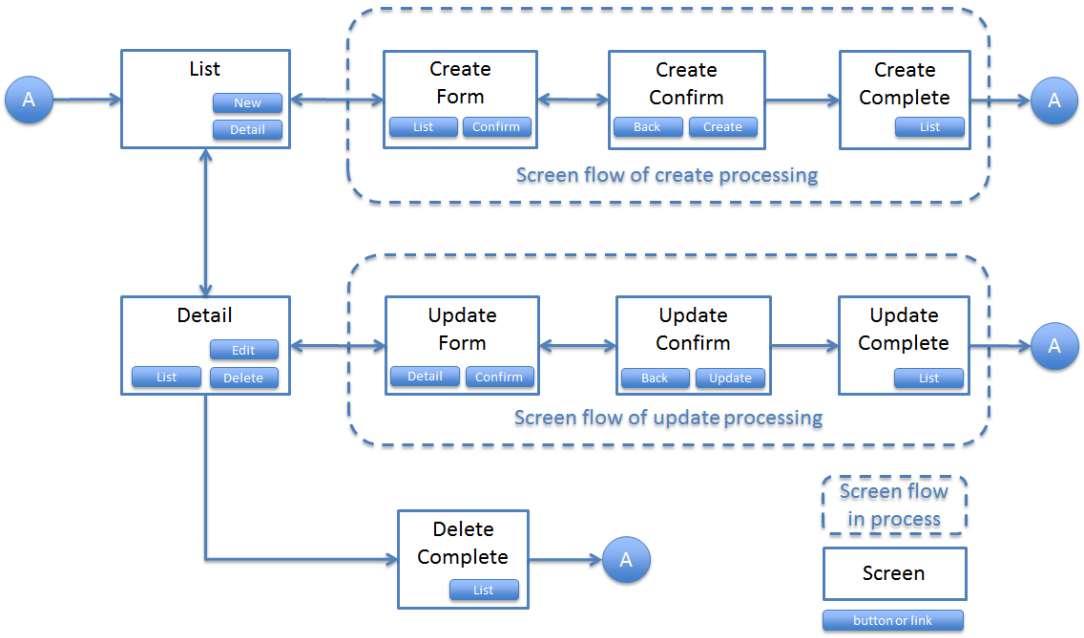


Figure.3.2 **Picture - Screen flow of entity management function**

Request URL

Design the URL of the required requests.

- Request URLs of all the requests required by the process flow are grouped.

  This functionality performs CRUD operations of Entity called ABC, therefore URL that starts with `"/abc/"` is considered.

- Design request URL for each operation of the functionality.

Sr. No.	Operation name	URL for each operation (pattern)
1.	Fetching list of Entities	/abc/list
2.	Create Entity	/abc/create
3.	Fetching details of Entity	/abc/{id}
4.	Entity update	/abc/{id}/update
5.	Entity delete	/abc/{id}/delete

Note: `"{id}"` specified in URL of 'Fetching details of Entity', 'Entity update', 'Entity delete' operations is called as, URI Template Pattern and any value can be specified. In this sample application, Entity ID is specified.

Assigned URL of each operation of screen flow diagram is mapped as shown below:

Mapping request and handler method

Design the mapping between incoming request and handler method.

The following is the mapping design which is designed according to mapping policy.

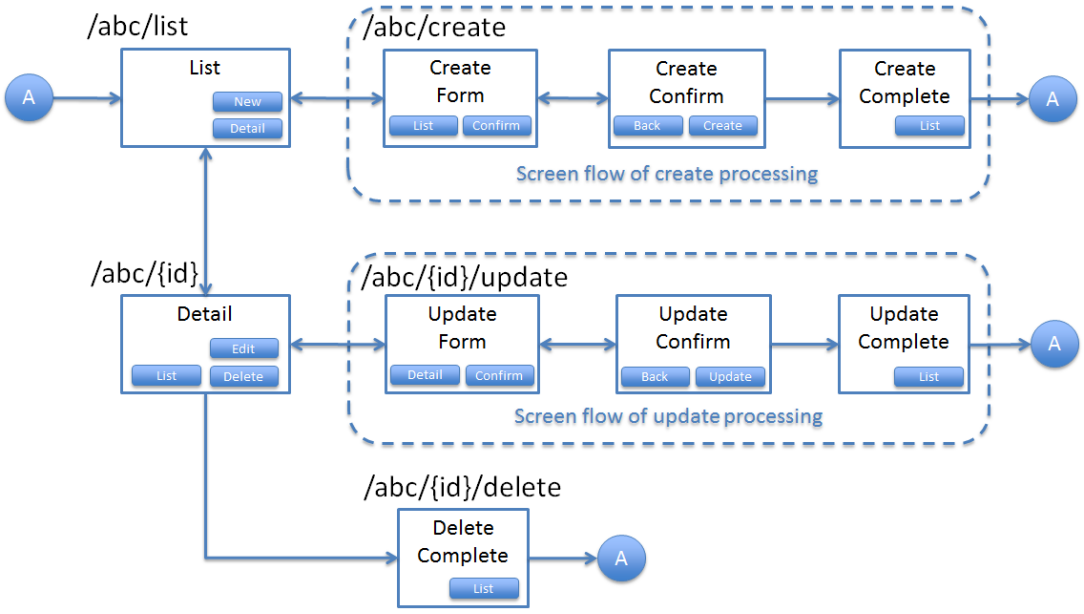


Figure.3.3 **Picture - Screen flow of entity management function and corresponding assigned URL**

Sr. No.	Operation name	URL	Request name	HTTP Method	HTTP Parameter	Handler method
1.	Fetching list of Entities	/abc/list	List display	GET	-	list
2.	Create New Entity	/abc/create	Form display	-	form	createForm
3.			Displaying input confirmation	POST	confirm	createConfirm
4.			Form re-display	POST	redo	createRedo
5.			Entity Creation	POST	-	create
6.			Displaying completion of Entity Creation	GET	complete	createComplete

次のページに続く

Table. 3.12 – 前のページからの続き

Sr. No.	Operation name	URL	Request name	HTTP Method	HTTP Parameter	Handler method
7.	Fetching details of Entity	/abc/{id}	Display details of Entity	GET	-	read
8.	Entity update	/abc/{id}/update	Displaying Form	-	form	updateForm
9.			Displaying confirmation of user input	POST	confirm	updateConfirm
10.			Form re-display	POST	redo	updateRedo
11.			Update	POST	-	update
12.			Displaying completion of update process	GET	complete	updateComplete
13.	Entity delete	/abc/{id}/delete	Delete	POST	-	delete
14.			Displaying completion of delete process	GET	complete	deleteComplete

Multiple requests exist for each of Create Entity, Entity Update and Entity Delete functions. Therefore switching of handler methods is done using HTTP method and HTTP parameters.

The following is the flow of requests in case of multiple requests in a function like “Create New Entity”.

All URLs are `"/abc/create"` and determining the handler method is done based on combination of HTTP method and HTTP parameters.

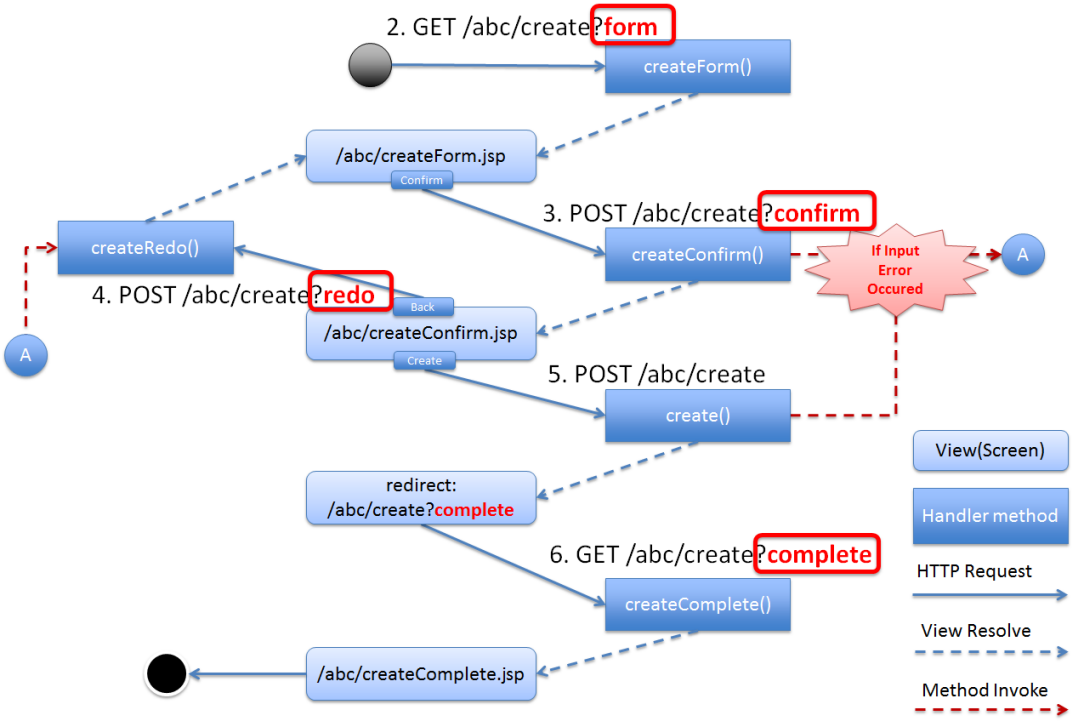


Figure.3.4 **Picture - Request flow of entity create processing**

Implementation of handler method for “Create New Entity” is shown below.

Here, the purpose is to understand mapping between request and handler method and therefore focus must on `@RequestMapping`.

The details of argument and return value (view name and view) of handler method are explained in the next chapter.

- *Implementing form display*
- *Implementing the display of user input confirmation screen*
- *Implementing ‘redisplay of form’*

- *Implementing 'create new user' business logic*
- *Implementing notification of create new user process completion*
- *Placing multiple buttons on HTML form*

Implementing form display

In order to display the form, `form` is specified as HTTP parameter.

```
@RequestMapping(value = "create", params = "form") // (1)
public String createForm(AbcForm form, Model model) {
    // omitted
    return "abc/createForm"; // (2)
}
```

Sr. No.	Description
(1)	Specify "`form`" as value of `params` attribute.
(2)	Return view name of JSP to render form screen.

Note: In this handler method, `method` attribute is not specified since it is not required for HTTP GET method.

Example of implementation of sections other than handler method is explained below.

Besides implementing the handler method for form display, points mentioned below are required:

- Implement generation process of form object. Refer to *Implementing form object* for the details of form object.
- Implement View of form screen. Refer to *Implementing View* for the details of View.

Use the following form object.

```
public class AbcForm implements Serializable {
    private static final long serialVersionUID = 1L;

    @NotEmpty
    private String input1;

    @NotNull
    @Min(1)
    @Max(10)
    private Integer input2;

    // omitted setter&getter
}
```

Creating an object of AbcForm.

```
@ModelAttribute
public AbcForm setUpAbcForm() {
    return new AbcForm();
}
```

Create view(JSP) of form screen.

```
<h1>Abc Create Form</h1>
<form:form modelAttribute="abcForm"
  action="${pageContext.request.contextPath}/abc/create">
  <form:label path="input1">Input1</form:label>
  <form:input path="input1" />
  <form:errors path="input1" />
  <br>
  <form:label path="input2">Input2</form:label>
  <form:input path="input2" />
  <form:errors path="input2" />
  <br>
  <input type="submit" name="confirm" value="Confirm" /> <!-- (1) -->
</form:form>
```

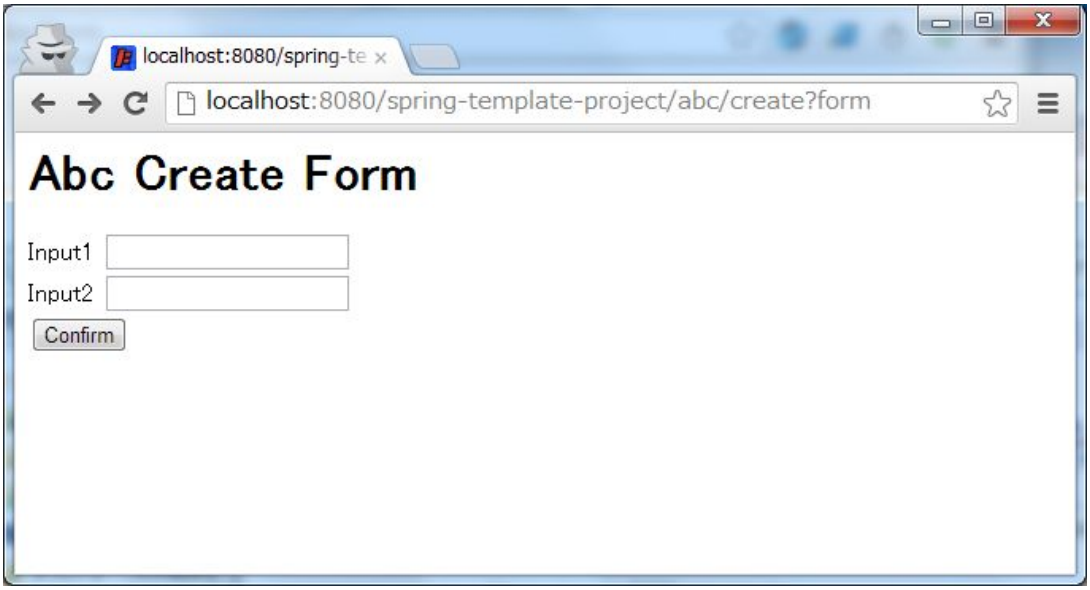
Sr. No.	Description
(1)	Specify `name="confirm"` parameter for submit button to transit to confirmation screen.

The operations are explained below.

Sending the request for form display.

Access "`abc/create?form`" URL.

Since `form` is specified in the URL as an HTTP parameter, `createForm` method of controller is called and form screen is displayed.



Implementing the display of user input confirmation screen

To check user input in the form, data is sent by POST method and `confirm` is specified as HTTP parameter.

```
@RequestMapping(value = "create", method = RequestMethod.POST, params =
↪"confirm") // (1)
public String createConfirm(@Validated AbcForm form, BindingResult result,
        Model model) {
    if (result.hasErrors()) {
        return createRedo(form, model); // return "abc/createForm"; (2)
    }
    // omitted
    return "abc/createConfirm"; // (3)
}
```

Sr. No.	Description
(1)	Specify "RequestMethod.POST" in `method` attribute and "confirm" in `params` attribute.
(2)	In case of input validation errors, it is recommended to call the handler method of form re-display.
(3)	Return view-name of JSP to render the screen for user input confirmation.

Note: POST method is specified to prevent displaying confidential information such as password and other personal information etc. in the address bar. (Needless to say that these security measures not sufficient and needs more secure measures such as SSL etc.)

Example of implementation of sections other than handler method is explained below.

Besides implementing handler method for user input confirmation screen, points mentioned below are required.

- Implement view of user-input confirmation screen. Refer to *Implementing View* for the details of view.

Create the view (JSP) for user input confirmation screen.

```
<h1>Abc Create Form</h1>
<form:form modelAttribute="abcForm"
  action="${pageContext.request.contextPath}/abc/create">
  <form:label path="input1">Input1</form:label>
  ${f:h(abcForm.input1)}
  <form:hidden path="input1" /> <!-- (1) -->
  <br>
  <form:label path="input2">Input2</form:label>
  ${f:h(abcForm.input2)}
  <form:hidden path="input2" /> <!-- (1) -->
  <br>
  <input type="submit" name="redo" value="Back" /> <!-- (2) -->
  <input type="submit" value="Create" /> <!-- (3) -->
</form:form>
```

Sr. No.	Description
(1)	The values entered on form screen is set as the hidden fields of HTML form since they must be sent back to the server when Create or Back buttons are clicked.
(2)	Specify `name="redo"` parameter for submit button to return to form screen.
(3)	Parameter name need not be specified for submit button. Submit button will do the actual create operation.

Note: In the above example, HTML escaping is performed as an XSS countermeasure using `f:h()` function while displaying the user input values. For details, refer to *Cross Site Scripting*.

The operations are explained below.

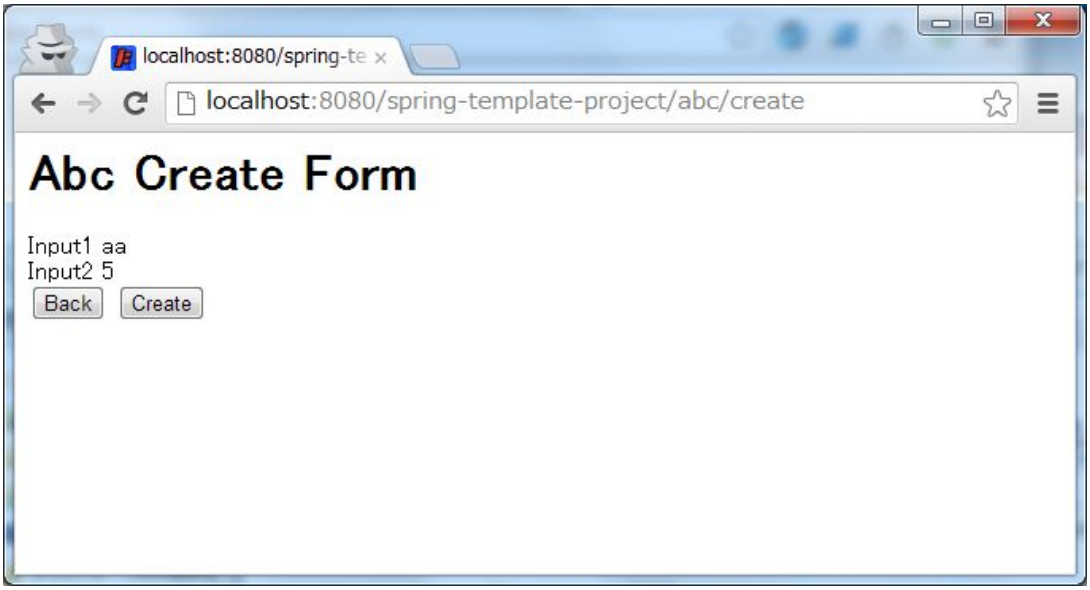
Send the request for displaying user input confirmation.

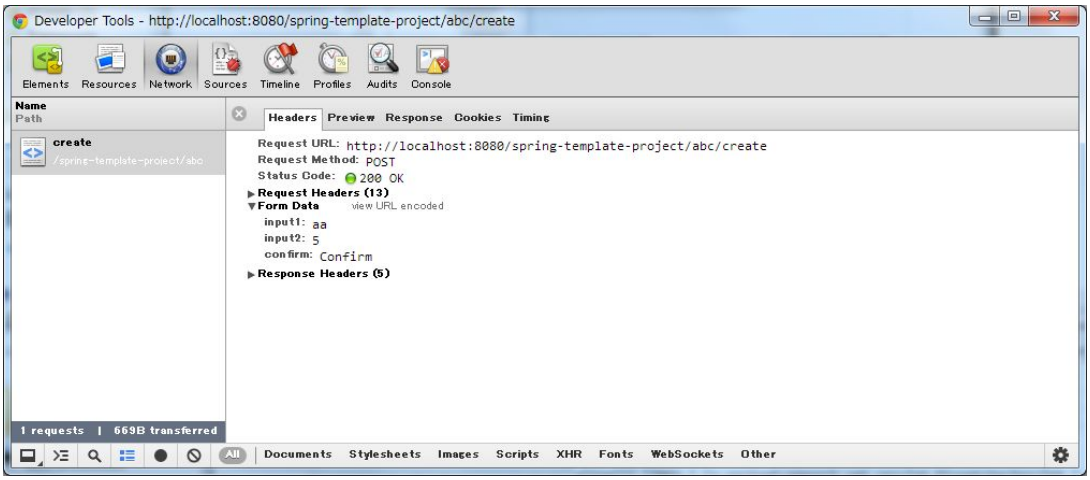
Enter `"aa"` in Input1 and `"5"` in Input2 and click Confirm button on form screen.

After clicking Confirm button, `"abc/create?confirm"` URI gets accessed using POST method.

Since HTTP parameter `confirm` is present in the URI, `createConfirm` method of controller is called and user input confirmation screen is displayed.

Since HTTP parameters are sent across through HTTP POST method after clicking the Confirm button, it does not appear in URI. However, “confirm” is included as HTTP parameter.





Implementing 'redisplay of form'

"redo" is specified as HTTP parameter to indicate that form needs to be redisplayed.

```
@RequestMapping(value = "create", method = RequestMethod.POST, params = "redo
↪") // (1)
public String createRedo(AbcForm form, Model model) {
    // omitted
    return "abc/createForm"; // (2)
}
```

Sr. No.	Description
(1)	Specify “RequestMethod.POST” in `method` attribute and “redo” in `params` attribute.
(2)	Return view name of JSP to render the form screen.

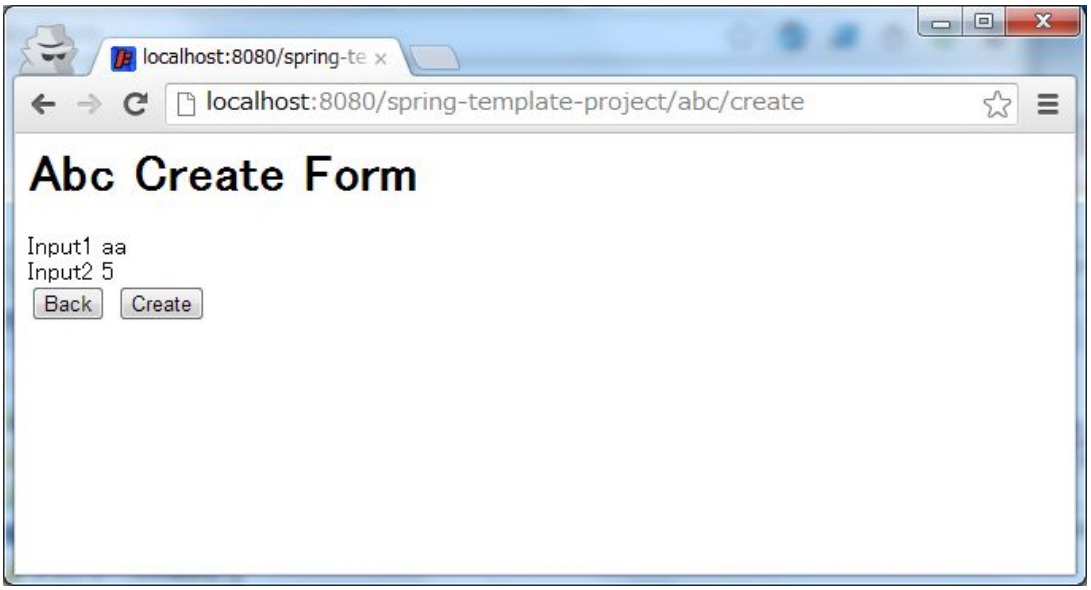
Operation is described below.

Send the request to redisplay the form screen.

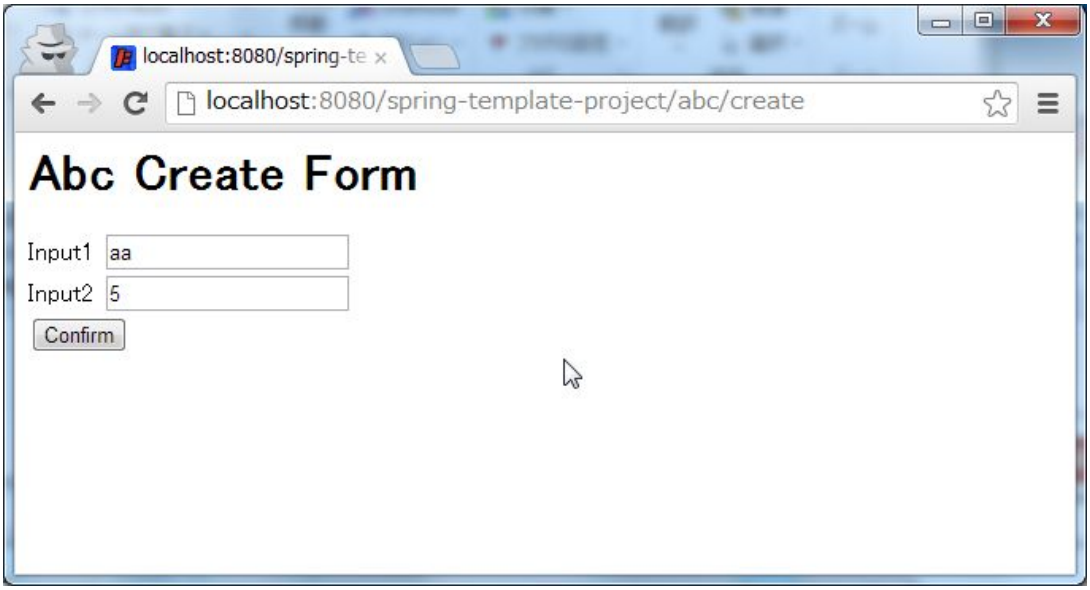
Click Back button on user input confirmation screen.

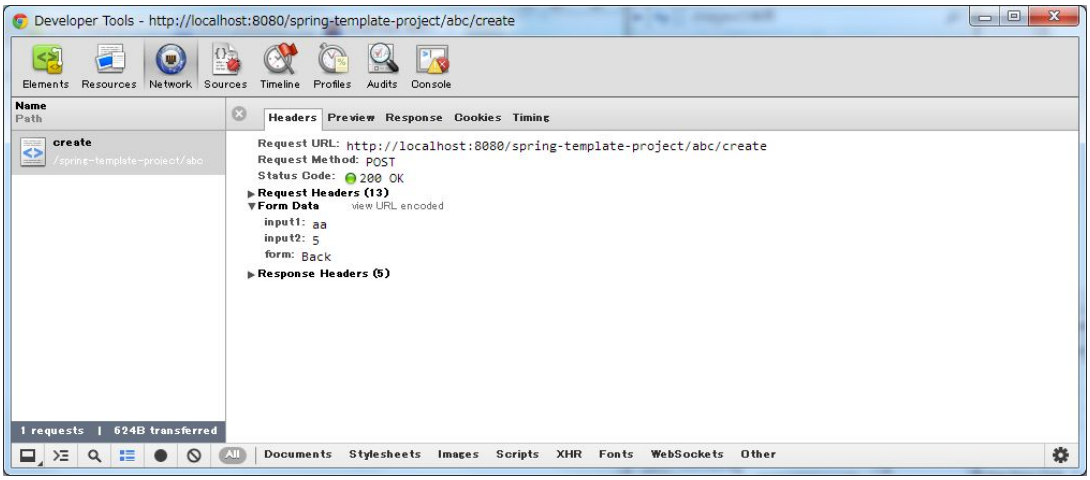
When Back button is clicked, “abc/create?redo” URI gets accessed through HTTP POST method.

Since “redo” HTTP parameter is present in the URI, `createRedo` method of controller is invoked and form screen is redisplayed.



Since HTTP parameters are sent across through HTTP POST method after clicking the Back button, it does not appear in URI. However, “redo” is included as HTTP parameter. Moreover, since input values of form had been sent as hidden fields, input values can be restored on redisplayed form screen.





Note: In order to implement back button functionality, setting `onclick="javascript:history.back()"` is also one of the ways. Both the methods differ in the following ways. Appropriate method must be selected as per requirement.

- Behavior when Back button on browser is clicked.
- Behavior when page having Back button is accessed and Back button is clicked.
- History of browser

Implementing 'create new user' business logic

To register input contents of form, the data (hidden parameters) to be registered is sent with HTTP POST method.

Sorting is not carried out using HTTP parameters since new request will be the main request of this operation.

Since the state of database changes in this process, it should not be executed multiple times due to double submission.

Therefore, it is 'redirected' to the next screen (create complete screen) instead of directly displaying View (screen) after

completing this process. This pattern is called as POST-Redirect-GET(PRG) pattern. For the details of PRG (Post-Redirect-Get) pattern

refer to *Double Submit Protection* .

```
@RequestMapping(value = "create", method = RequestMethod.POST) // (1)
public String create(@Validated AbcForm form, BindingResult result, Model␣
↪model) {
    if (result.hasErrors()) {
        return createRedo(form, model); // return "abc/createForm";
    }
    // omitted
    return "redirect:/abc/create?complete"; // (2)
}
```

Sr. No.	Description
(1)	Specify `RequestMethod.POST` in `method` attribute. Do not specify `params` attribute.
(2)	Return URL to the request needs to be redirected as view name in order to use PRG pattern.

Note: It can be redirected to "/xxx" by returning "redirect:/xxx" as view name.

Warning: PRG pattern is used to avoid double submission when the browser gets reloaded by clicking F5 button. However, as a countermeasure for double submission, it is necessary to use TransactionTokenCheck functionality. For details of TransactionTokenCheck, refer to *Double Submit Protection* .

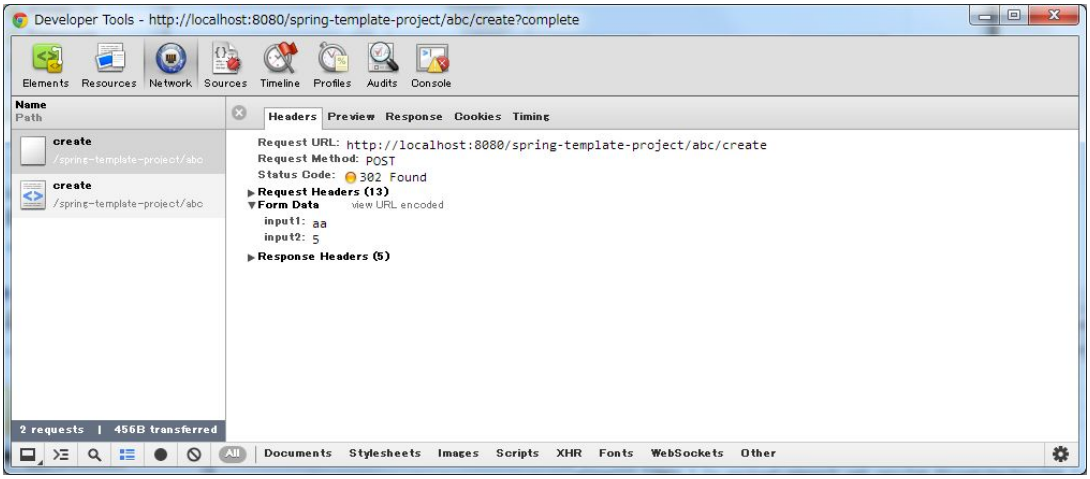
Operation is described below.

Click 'Create' button on input confirmation screen.

After clicking 'Create' button, "abc/create" URL is accessed through POST method.

Since HTTP parameters are not sent for identifying a button, it is considered as main request of Entity create process and 'create' method of Controller is invoked.

'Create' request does not return to the screen directly, but it is redirected to create complete display ("/abc/create?complete"). Hence HTTP status is changed to 302.



Implementing notification of create new user process completion

In order to notify the completion of create process, complete must be present in the request as HTTP parameter.

```
@RequestMapping(value = "create", params = "complete") // (1)
public String createComplete() {
    // omitted
    return "abc/createComplete"; // (2)
}
```

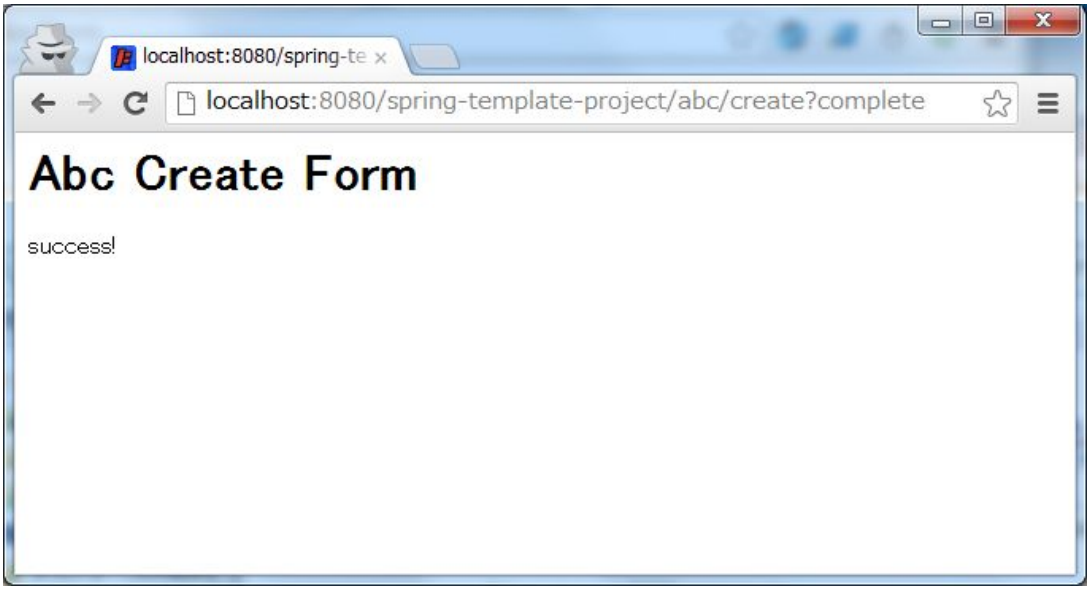
Sr. No.	Description
(1)	Specify "complete" in params attribute.
(2)	Return View name of JSP to render the create completion screen.

Note: In this handler method, method attribute is not specified since it is not required for HTTP GET method.

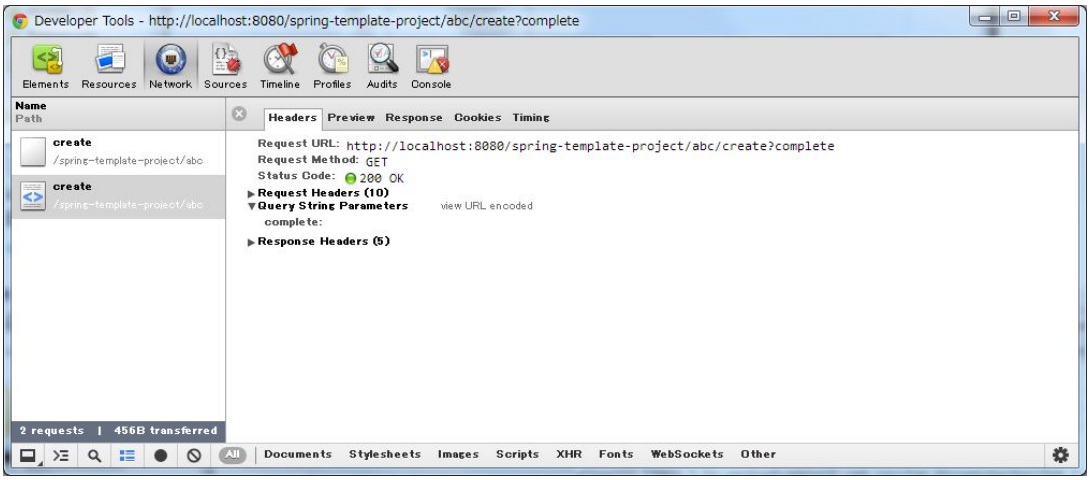
Operation is described below.

After completing creation of user, access URI ("/abc/create?complete") is specified as redirect destination.

Since HTTP parameter is complete, createComplete() method of controller is called and create completion screen is displayed.



Note: Since PRG pattern is used, even if browser is reloaded, create completion screen is only re-displayed



without re-executing create process.

Placing multiple buttons on HTML form

To place multiple buttons on a single form, send HTTP parameter to identify the corresponding button and so that the handler method of controller can be switched. An example of Create button and Back button on input confirmation screen of sample application is explained here.

'Create' button to perform 'user creation' and 'Back' button to redisplay 'create form' exists on the form of input confirmation screen as shown below.

To redisplay 'create form' using request (`"/abc/create?redo"`) when Back button is clicked, the following code is required in HTML form.

```
<input type="submit" name="redo" value="Back" /> <!-- (1) -->
<input type="submit" value="Create" />
```

Sr. No.	Description
(1)	In input confirmation screen (`"abc/createConfirm.jsp"`), specify `name="redo"` parameter for Back button.

For the operations when Back button is clicked, refer to *Implementing 'redisplay of form'*.

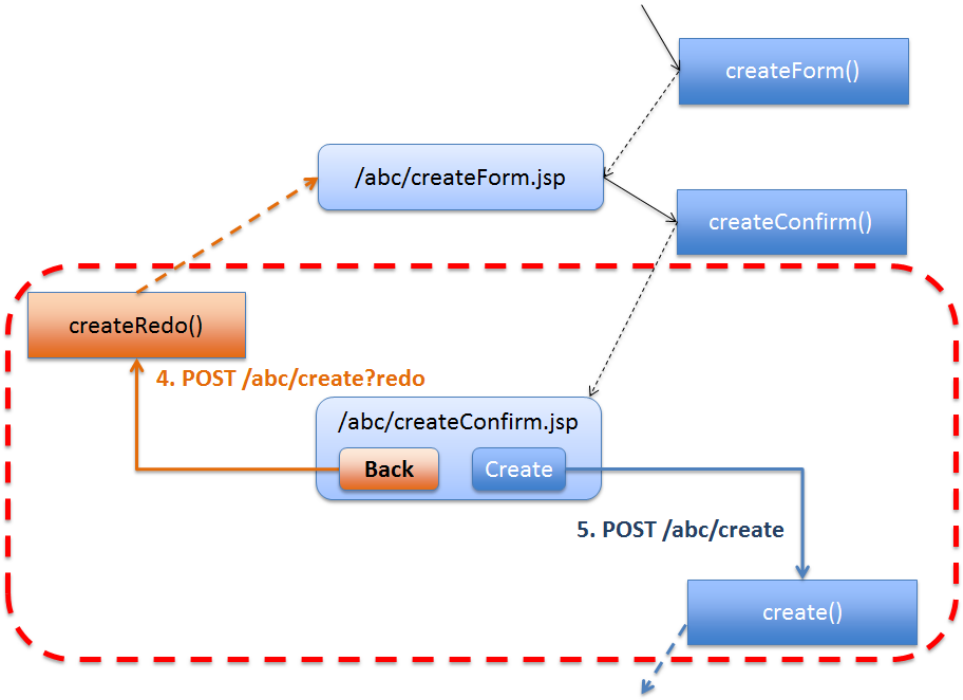


Figure.3.5 **Picture - Multiple button in the HTML form**

Source code of controller of sample application

Source-code of controller after implementing create process of sample application are shown below.

Fetching list of Entities, Fetching detail of Entity, Entity update, Entity delete are implemented using the same guidelines.

```
@Controller
@RequestMapping("abc")
public class AbcController {

    @ModelAttribute
    public AbcForm setUpAbcForm() {
        return new AbcForm();
    }

    // Handling request of "/abc/create?form"
    @RequestMapping(value = "create", params = "form")
    public String createForm(AbcForm form, Model model) {
        // omitted
        return "abc/createForm";
    }

    // Handling request of "POST /abc/create?confirm"
    @RequestMapping(value = "create", method = RequestMethod.POST, params =
↪"confirm")
```

```
    public String createConfirm(@Validated AbcForm form, BindingResult result,
            Model model) {
        if (result.hasErrors()) {
            return createRedo(form, model);
        }
        // omitted
        return "abc/createConfirm";
    }

    // Handling request of "POST /abc/create?redo"
    @RequestMapping(value = "create", method = RequestMethod.POST, params =
↪"redo")
    public String createRedo(AbcForm form, Model model) {
        // omitted
        return "abc/createForm";
    }

    // Handling request of "POST /abc/create"
    @RequestMapping(value = "create", method = RequestMethod.POST)
    public String create(@Validated AbcForm form, BindingResult result, Model␣
↪model) {
        if (result.hasErrors()) {
            return createRedo(form, model);
        }
        // omitted
        return "redirect:/abc/create?complete";
    }

    // Handling request of "/abc/create?complete"
    @RequestMapping(value = "create", params = "complete")
    public String createComplete() {
        // omitted
        return "abc/createComplete";
    }

}
```

Regarding arguments of handler method

The arguments of handler method can be used to fetch various values; however, as a principle rule, the following should not be fetched using arguments of handler method of controller.

- ServletRequest

- HttpServletRequest
- org.springframework.web.context.request.WebRequest
- org.springframework.web.context.request.NativeWebRequest
- java.io.InputStream
- java.io.Reader
- java.io.OutputStream
- java.io.Writer
- java.util.Map
- org.springframework.ui.ModelMap

Note: When generalized values like getAttribute/setAttribute of `HttpServletRequest` and get/put of `Map` are allowed, liberal use of these can degrade the maintainability of the project with an increase in project size.

For the above reason, using `HttpSession` is not reccomended in the case when there are alternatives.

When common parameters (request parameters) need to be stored in JavaBean and passed as an argument to a method of controller, it can be implemented using *Implementing HandlerMethodArgumentResolver* as described later.

Arguments depending on the purpose of usage are described below.

- *Passing data to screen (View)*
- *Retrieving values from URL path*
- *Retrieving request parameters individually*
- *Retrieving request parameters collectively*
- *Performing input validation*
- *Passing data while redirecting request*
- *Passing request parameters to redirect destination*
- *Inserting values in redirect destination URL path*

- *Acquiring values from Cookie*
- *Writing values in Cookie*
- *Retrieving pagination information*
- *Retrieving uploaded file*
- *Displaying result message on the screen*

Passing data to screen (View)

To pass data to be displayed on screen (View), fetch `org.springframework.ui.Model` (Hereafter called as `Model`) as argument of handler method and add the data (Object) to `Model` object.

- SampleController.java

```
@RequestMapping("hello")
public String hello(Model model) { // (1)
    model.addAttribute("hello", "Hello World!"); // (2)
    model.addAttribute(new HelloBean("Bean Hello World!")); // (3)
    return "sample/hello"; // returns view name
}
```

- hello.jsp

```
Message : ${f:h(hello)}<br> <%-- (4) --%>
Message : ${f:h(helloBean.message)}<br> <%-- (5) --%>
```

- HTML of created by View(hello.jsp)

```
Message : Hello World!<br> <!-- (6) -->
Message : Bean Hello World!<br> <!-- (6) -->
```

Sr. No.	Description
(1)	Fetch `Model` object as argument.
(2)	Call `addAttribute` method of `Model` object received as argument, and add the data to `Model` object. For example, `"HelloWorld!"` string is added to the attribute name `"hello"`.
(3)	If first argument of `addAttribute` method is omitted, the class name beginning with lower case letter will become the attribute name. For example, the result of `model.addAttribute("helloBean",new HelloBean());` is same as the result of `model.addAttribute(new HelloBean());`
(4)	In View (JSP), it is possible to acquire the data added to `Model` object by specifying "${Attribute name}". For example, HTML escaping is performed using "${f:h(Attribute name)}" function of EL expression. For details of functions of EL expression that perform HTML escaping, refer to *Cross Site Scripting*.
(5)	The values of JavaBean stored in `Model` can be acquired by specifying "${Attribute name.property name}".
(6)	JSP is output in HTML format.

Note: Even though the `Model` is not used, it can be specified as an argument. Even if it is not required at the initial stage of implementation, it can be used later (so that the signature of methods need not be changed in future).

Note: The value can also be referred from the module which is not managed under Spring MVC

(for example, ServletFilter, etc.) since `addAttribute` in `Model` performs a `setAttribute` in `HttpServletRequest`.

Retrieving values from URL path

To retrieve values from URL path, add `@PathVariable` annotation to argument of handler method of controller.

In order to retrieve values from the path using `@PathVariable` annotation, value of `@RequestMapping` annotation must contain those values in the form of variables (for example, {id}).

```
@RequestMapping("hello/{id}/{version}") // (1)
public String hello(
        @PathVariable("id") String id, // (2)
        @PathVariable Integer version, // (3)
        Model model) {
    // do something
    return "sample/hello"; // returns view name
}
```

Sr. No.	Description
(1)	Specify the portion to be extracted as path variable in the value of `@RequestMapping` annotation. Specify path variable in "{variable name}" format. For example, 2 path variables such as `"id"` and `"version"` are specified.
(2)	Specify variable name of path variable in `@PathVariable` annotation. For example, when the URL `"sample/hello/aaaa/1"` is accessed, the string `"aaaa"` is passed to argument "id".
(3)	Value attribute of `@PathVariable` annotation can be omitted. When it is omitted, the argument name is considered as the request parameter name. In the above example, when the URL `"sample/hello/aaaa/1"` is accessed, value `"1"` is passed to argument "version". However, in this method compilation needs to be done by specifying either of: • `-g` option (mode to output debug information) • `-parameters` option added from Java8 (mode to generate metadata for reflection in the method parameters)

Note: Binding argument can be of any data type other than string. In case of different data type, `org.springframework.beans.TypeMismatchException` is thrown and default response is 400 (Bad Request). For example, when the URL `"sample/hello/aaaa/v1"` is accessed, an exception is thrown since `"v1"` cannot be converted into Integer type.

Warning: When omitting the value attribute of `@PathVariable` annotation, the application to be deployed needs to be compiled by specifying `-g` option or `-parameters` option which is added from Java8. When these options are specified, there is a likely impact on memory and processing performance since information or processing required for debugging gets appended to the class after compilation. Basically, it is recommended to explicitly specify the value attribute.

Retrieving request parameters individually

To retrieve request parameters individually, add `@RequestParam` annotation to argument.

```
@RequestMapping("bindRequestParams")
public String bindRequestParams(
        @RequestParam("id") String id, // (1)
        @RequestParam String name, // (2)
        @RequestParam(value = "age", required = false) Integer age, // (3)
        @RequestParam(value = "genderCode", required = false, defaultValue =
↪"unknown") String genderCode, // (4)
        Model model) {
    // do something
    return "sample/hello"; // returns view name
}
```

Sr. No.	Description
(1)	Specify request parameter name in the value attribute of `@RequestParam` annotation. For example, when the URL `"sample/hello?id=aaaa"` is accessed, the string `"aaaa"` is passed to argument “id”.
(2)	value attribute of `@RequestParam` annotation can be omitted. When it is omitted, the argument name becomes the request parameter name. For example, when the URL `"sample/hello?name=bbbb&...."` is accessed, string `"bbbb"` is passed to argument “name”. However, in this method compilation needs to be done by specifying either of: • `-g` option (mode to output debug information) • `-parameters` option added from Java8 (mode to generate metadata for reflection in the method parameters)
(3)	By default, an error occurs if the specified request parameter does not exist. When request parameter is not required, specify `false` in the `required` attribute. For example, when it is accessed where request parameter `age` does not exist, `null` is passed to argument “age”.
(4)	When default value is to be used if the specified request parameter does not exist, specify the default value in defaultValue attribute. For example, when it is accessed where request parameter `genderCode` does not exist, `"unknown"` is passed to argument “genderCode”.

Note: When it is accessed without specifying mandatory parameters, `org.springframework.web.bind.MissingServletRequestParameterException` is thrown and default operation is responded with 400 (Bad Request). However, when defaultValue attribute is specified, the value specified in defaultValue attribute is passed without throwing exception.

Note: Binding argument can be of any data type. In case the data type do not match, `org.springframework.beans.TypeMismatchException` is thrown and default response is 400 (Bad Request). For example, when `"sample/hello?age=aaaa&..."` URL is accessed, ex-

ception is thrown since `aaaa` cannot be converted into Integer.

Binding to form object must be done only when any of the following conditions are met.

- If request parameter is an item in the HTML form.
- If request parameter is not an item in HTML form, however, input validation other than mandatory check needs to be performed.
- If error details of input validation error needs to be output for each parameter.
- If there are 3 or more request parameters. (maintenance and readability point of view)

Retrieving request parameters collectively

Use form object to collectively fetch all the request parameters.

Form object is JavaBean representing HTML form. For the details of form object, refer to *Implementing form object*.

Following is an example that shows the difference between handler method that fetches each request parameter using `@RequestParam` and the same handler method when fetching request parameters in a form object

Handler method that receives request parameter separately using `@RequestParam` is as shown below.

```
@RequestMapping("bindRequestParams")
public String bindRequestParams(
        @RequestParam("id") String id,
        @RequestParam String name,
        @RequestParam(value = "age", required = false) Integer age,
        @RequestParam(value = "genderCode", required = false, defaultValue =
↪"unknown") String genderCode,
        Model model) {
    // do something
    return "sample/hello"; // returns view name
}
```

Create form object class

For jsp of HTML form corresponding to this form object, refer to *Binding to HTML form*.

```
public class SampleForm implements Serializable{
    private static final long serialVersionUID = 1477614498217715937L;

    private String id;
    private String name;
    private Integer age;
    private String genderCode;

    // omit setters and getters

}
```

Note: Request parameter name should match with form object property name.

When parameters `"id=aaa&name=bbbb&age=19&genderCode=men?tel=01234567"` are sent to the above f
the values of `id` , `name` , `age` , `genderCode` matching with the property name, are stored, however `tel` is not included in form object, as it does not have matching property name.

Make changes such that request parameters which were being fetched individually using `@RequestParam` now get fetched as form object.

```
@RequestMapping("bindRequestParams")
public String bindRequestParams(@Validated SampleForm form, // (1)
        BindingResult result,
        Model model) {
    // do something
    return "sample/hello"; // returns view name
}
```

Sr. No.	Description
(1)	Receive `SampleForm` object as argument.

Note: When form object is used as argument, unlike `@RequestParam`, mandatory check is not performed. **When using form object,** *Performing input validation* **should be performed as described below**.

Warning: Domain objects such as Entity, etc. can also be used as form object without any changes required. However, the parameters such as password for confirmation, agreement confirmation checkbox, etc. should exist only on WEB screen. Since the fields depending on such screen items should not be added to domain objects, it is recommended to create class for form object separate from domain object. When a domain object needs to be created from request parameters, values must first be received in form object and then copied to domain object from form object.

Performing input validation

When performing input validation for the form object, add `@Validated` annotation to form object argument, and specify `org.springframework.validation.BindingResult` (Hereafter called as `BindingResult`) to argument immediately after form object argument.

Refer to *Input Validation* for the details of input validation.

Add annotations required in input validation to the fields of form object class.

```
public class SampleForm implements Serializable {
    private static final long serialVersionUID = 1477614498217715937L;

    @NotNull
    @Size(min = 10, max = 10)
    private String id;

    @NotNull
    @Size(min = 1, max = 10)
    private String name;

    @Min(1)
    @Max(100)
    private Integer age;

    @Size(min = 1, max = 10)
    private Integer genderCode;

    // omit setters and getters
}
```

Add `@Validated` annotation to form object argument.

Input validation is performed for the argument with `@Validated` annotation before the handler method of controller is executed. The check result is stored in the argument `BindingResult` which immediately follows form object argument.

The type conversion error that occurs when a data-type other than String is specified in form object, is also stored in `BindingResult`.

```
@RequestMapping("bindRequestParams")
public String bindRequestParams(@Validated SampleForm form, // (1)
        BindingResult result, // (2)
        Model model) {
    if (result.hasErrors()) { // (3)
        return "sample/input"; // back to the input view
    }
    // do something
    return "sample/hello"; // returns view name
}
```

Sr. No.	Description
(1)	Add `@Validated` annotation to `SampleForm` argument, and mark it as target for input validation.
(2)	Specify `BindingResult` in the argument where input validation result is stored.
(3)	Check if input validation error exists. If there is an error, `true` is returned.

Passing data while redirecting request

To redirect after executing a handler method ofcontroller and to pass data along with it, fetch `org.springframework.web.servlet.mvc.support.RedirectAttributes` (Henceforth called as `RedirectAttributes`) as an argument of handler method, and add the data to `RedirectAttributes` object.

- SampleController.java

```
@RequestMapping("hello")
public String hello(RedirectAttributes redirectAttrs) { // (1)
```

```
    redirectAttrs.addFlashAttribute("hello", "Hello World!"); // (2)
    redirectAttrs.addFlashAttribute(new HelloBean("Bean Hello World!")); //␣
↪(3)
    return "redirect:/sample/hello?complete"; // (4)
}

@RequestMapping(value = "hello", params = "complete")
public String helloComplete() {
    return "sample/complete"; // (5)
}
```

- complete.jsp

```
Message : ${f:h(hello)}<br> <%-- (6) --%>
Message : ${f:h(helloBean.message)}<br> <%-- (7) --%>
```

- HTML of created by View(complete.jsp)

```
Message : Hello World!<br> <!-- (8) -->
Message : Bean Hello World!<br> <!-- (8) -->
```

Sr. No.	Description
(1)	Fetch `RedirectAttributes` object as argument of the handler method of controller.
(2)	Call `addFlashAttribute` method of `RedirectAttributes` and add the data to `RedirectAttributes` object. For example, the string data `"HelloWorld!"` is added to attribute name `"hello"`.
(3)	If first argument of `addFlashAttribute` method is omitted, the class name beginning with lower case letter becomes the attribute name. For example, the result of `model.addFlashAttribute("helloBean",new HelloBean());` is same as `model.addFlashAttribute(new HelloBean());`.
(4)	Send a redirect request to another URL which will display the next screen instead of displaying screen (View) directly.

次のページに続く

Table. 3.13 – 前のページからの続き

Sr. No.	Description
(5)	In the handler method after redirection, return view name of the screen that displays the data added in (2) and (3).
(6)	In the View (JSP) side, the data added to `RedirectAttributes` object can be obtained by specifying “${attribute name}”. For example, HTML escaping is performed using “${f:h(attribute name)}” function of EL expression. For the details of functions of EL expression that performs HTML escaping, refer to *Cross Site Scripting*.
(7)	The value stored in `RedirectAttributes` can be obtained from JavaBean by using “${Attribute name.Property name}”.
(8)	HTML output.

Warning: The data cannot be passed to redirect destination even though it is added to `Model`.

Note: It is similar to the `addAttribute` method of `Model`. However survival time of data differs. In `addFlashAttribute` of `RedirectAttributes`, the data is stored in a scope called flash scope. Data of only 1 request (G in PRG pattern) can be referred after redirect. The data from the second request onwards is deleted.

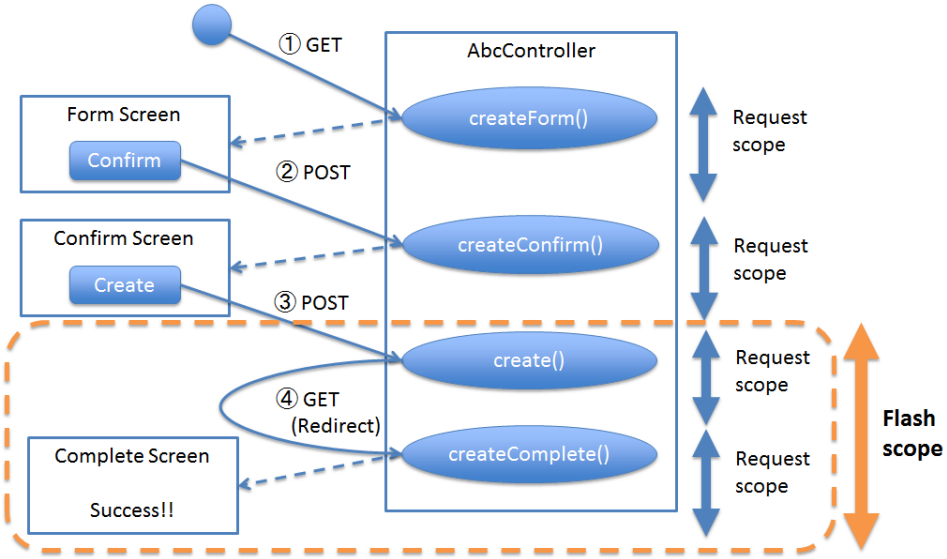


Figure.3.6 **Picture - Survival time of flush scope**

Passing request parameters to redirect destination

When request parameters are to be set dynamically to redirect destination, add the values to be passed to `RedirectAttributes` object of argument.

```
@RequestMapping("hello")
public String hello(RedirectAttributes redirectAttrs) {
    String id = "aaaa";
    redirectAttrs.addAttribute("id", id); // (1)
    // must not return "redirect:/sample/hello?complete&id=" + id;
    return "redirect:/sample/hello?complete";
}
```

Sr. No.	Description
(1)	Specify request parameter name in argument `name and request parameter value in argument ``value` and call `addAttribute` method of `RedirectAttributes` object. In the above example, it is redirected to `"/sample/hello?complete&id=aaaa"`.

Warning: In the above example, the result is the same as of `return "redirect:/sample/hello?complete&id=" + id;` (as shown in the commented out line in the above example). However, since URL encoding is also performed if `addAttribute` method of `RedirectAttributes` object is used, the request parameters that needs to be inserted dynamically **should be set to the request parameter using addAttribute method and should not be set to redirect URL specified as return value.** The request parameters which are not to be inserted dynamically ("complete" as in the above example), can be directly specified in the redirect URL specified as the return value.

Inserting values in redirect destination URL path

To insert values in redirect destination URL path dynamically, add the value to be inserted in `RedirectAttributes` object of argument as shown in the example to set request parameters.

```
@RequestMapping("hello")
public String hello(RedirectAttributes redirectAttrs) {
    String id = "aaaa";
    redirectAttrs.addAttribute("id", id); // (1)
    // must not return "redirect:/sample/hello/" + id + "?complete";
    return "redirect:/sample/hello/{id}?complete"; // (2)
}
```

Sr. No.	Description
(1)	Specify attribute name and the value using `addAttribute` method of `RedirectAttributes` object.
(2)	Specify the path of the variable "{Attribute name}" to be inserted in the redirect URL. In the above example, it is redirected to `"/sample/hello/aaaa?complete"`.

Warning: In the above example, the result is same as of `"redirect:/sample/hello/" + id + "?complete";` (as shown in the commented out line in the above example). However, since URL encoding is also performed when using `addAttribute` method of `RedirectAttributes` object, the path values to be inserted dynamically **should be inserted using addAttribute method and path variable and should not be set to redirect URL specified as return value.**

Acquiring values from Cookie

Add `@CookieValue` annotation to the argument of handler method to acquire the values from a cookie.

```
@RequestMapping("readCookie")
public String readCookie(@CookieValue("JSESSIONID") String sessionId, Model␣
↪model) { // (1)
    // do something
    return "sample/readCookie"; // returns view name
}
```

Sr. No.	Description
(1)	Specify name of the cookie in the `value` attribute of `@CookieValue` annotation. In the above example, "JSESSIONID" value is passed from cookie to sessionId argument.

Note:

As in the case of `@RequestParam` , it has `required` attribute and `defaultValue` attribute. Also, the data type of the a

Refer to *Retrieving request parameters individually* for details.

Writing values in Cookie

To write values in cookie, call `addCookie` method of `HttpServletResponse` object directly and add the value to cookie.

Since there is no way to write to cookie in Spring MVC (3.2.3 version), ** Only in this case, HttpServletResponse can fetched as an argument of handler method of controller.**

```
@RequestMapping("writeCookie")
public String writeCookie(Model model,
        HttpServletResponse response) { // (1)
    Cookie cookie = new Cookie("foo", "Helloworld!");
    response.addCookie(cookie); // (2)
    // do something
    return "sample/writeCookie";
}
```

Sr. No.	Description
(1)	Specify `HttpServletResponse` object as argument to write to cookie.
(2)	Generate `Cookie` object and add to `HttpServletResponse` object. For example, `"Helloworld!"` value is assigned to Cookie name `"foo"`.

Tip: No difference compared to use of `HttpServletResponse` which fetched as an argument of handler method, however, `org.springframework.web.util.CookieGenerator` class is provided by Spring as a class to write values in cookie. It should be used if required.

Note: Please note that some of the characters that cannot be used for Cookie name and value exist in In RFC 6265 which prescribes HTTP Cookie process. For example, space character cannot be used in Cookie value in Tomcat 8.5 of implementation which is in conformance with RFC 6265.

Refer to RFC 6265(HTTP State Management Mechanism) ☒ 4.1 SetCookie Syntax.

Retrieving pagination information

Pagination related information is required for the requests performing list search.

Fetching `org.springframework.data.domain.Pageable` (henceforth called as `Pageable`) object as an argument of handler method enables to handle pagination related information (page count, fetch record count) easily.

> Refer to *Pagination* for details.

Retrieving uploaded file

Uploaded file can be obtained in 2 ways.

- Provide `MultipartFile` property in form object.
- Use `org.springframework.web.multipart.MultipartFile` as an argument of handler method having `@RequestParam` annotation.

Refer to *File Upload* for details.

Displaying result message on the screen

`Model` object or `RedirectAttributes` object can be obtained as an argument of handler method and result message of business logic execution can be displayed by adding `ResultMessages` object to Model or RedirectAttributes.

Refer to *Message Management* for details.

Regarding return value of handler method

For return values of handler methods, various values can be fetched ; however, only the following values should be used.

- String (for logical name of view)

Return types depending on the purpose of usage are described below:

- *HTML response*
- *Responding to downloaded data*

HTML response

To get HTML response to display the output of handler method, it has to return view name of JSP.

`ViewResolver` when generating HTML using JSP will be an inherited class of `UrlBasedViewResolver` (`InternalViewResolver` and‘‘TilesViewResolver‘‘, etc).

An example using `InternalViewResolver` for JSP is given below; however, it is recommended to use `TilesViewResolver` when the screen layout is in a templated format.

Refer to *Screen Layout using Tiles* for the usage of `TilesViewResolver`.

- spring-mvc.xml

Example of definition when `<bean>` element is to be used

```
<!-- (1) -->
<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver
↪">
    <property name="prefix" value="/WEB-INF/views/" /> <!-- (2) -->
    <property name="suffix" value=".jsp" /> <!-- (3) -->
    <property name="order" value="1" /> <!-- (4) -->
</bean>
```

Example of definition when using <mvc:view-resolvers> element added from Spring Framework 4.1

```
<mvc:view-resolvers>
    <mvc:jsp prefix="/WEB-INF/views/" /> <!-- (5) -->
</mvc:view-resolvers>
```

- SampleController.java

```
@RequestMapping("hello")
public String hello() {
    // omitted
    return "sample/hello"; // (6)
}
```

Sr. No.	Description
(1)	Define InternalViewResolver for JSP.
(2)	Specify base directory (prefix of file path) where JSP files are stored. By specifying prefix, there is no need to specify physical storage location of JSP files at the time of returning View name in Controller.
(3)	Specify extension (suffix of file path) of JSP file. By specifying suffix, specifying extension of JSP files at the time of returning View name in Controller is no longer needed.
(4)	Specify execution order when multiple ViewResolver are specified. It can be specified in the range of Integer and executed sequentially from smallest value.
(5)	Define InternalViewResolver for JSP using <mvc:jsp> element added from Spring Framework 4.1. • In prefix attribute, specify base directory (prefix of file path) where JSP file is stored. • It need not be explicitly specified in suffix attribute as ".jsp" is used as default value. **Note:** When <mvc:view-resolvers> element is used, it is possible to define ViewResolver in simple way. Hence this guideline recommends to use <mvc:view-resolvers>.
(6)	When View name "sample/hello" is the return value of handler method, "/WEB-INF/views/sample/hello.jsp" is called and HTML is sent as response.

Note: HTML output is generated using JSP in the above example, however, even if HTML is generated using other template engine such as Velocity, FreeMarker, return value of handler method will be `"sample/hello"`. `ViewResolver` takes care of task to determine which template engine is to be used.

Note: When simply a method to return view name is to be implemented, implementation of Controller class can be substituted by using `<mvc:view-controller>` .

- Implementation example of Controller which uses `<mvc:view-controller>`.

```
<mvc:view-controller path="/hello" view-name="sample/hello" />
```

Warning: Considerations for using <mvc:view-controller>

Since HTTP methods authorized by `<mvc:view-controller>` are restricted only to GET and HEAD due to version-up of Spring Framework 4.3 (SPR-13130), `<mvc:view-controller>` cannot used for the pages which are to be accessed by HTTP methods other than GET and HEAD (POST etc). This also happens when pages are forwarded by methods other than GET and HEAD (POST etc). Hence, when HTTP methods for forwarding operations such as transition to error page etc cannot be restricted, adequate care must be taken to avoid the use of `<mvc:view-controller>`.

Responding to downloaded data

In order to return the data stored in db as download data (`"application/octet-stream"`), it is recommended to create a view

for generating response data (download process).The handler method adds the data to be downloaded to `Model` and returns

name of the view which performs the actual download process.

The solution to create a separate ViewResolver to resolve a view using its view name, however, `BeanNameViewResolver` provided by Spring Framework is recommended.

Refer to *File Download* for the details of download processing.

- spring-mvc.xml

  Example of definition when <bean> element is to be used

```
<!-- (1) -->
<bean class="org.springframework.web.servlet.view.BeanNameViewResolver">
    <property name="order" value="0"/> <!-- (2) -->
</bean>

<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver
↪">
    <property name="prefix" value="/WEB-INF/views/" />
    <property name="suffix" value=".jsp" />
    <property name="order" value="1" />
</bean>
```

  Example of definition when using <mvc:view-resolvers> element added from Spring Framework 4.1

```
<mvc:view-resolvers>
    <mvc:bean-name /> <!-- (3) -->
    <mvc:jsp prefix="/WEB-INF/views/" />
</mvc:view-resolvers>
```

- SampleController.java

```
@RequestMapping("report")
public String report() {
    // omitted
    return "sample/report"; // (4)
}
```

- XxxExcelView.java

```
@Component("sample/report") // (5)
public class XxxExcelView extends AbstractExcelView { // (6)
    @Override
    protected void buildExcelDocument(Map<String, Object> model,
            HSSFWorkbook workbook, HttpServletRequest request,
            HttpServletResponse response) throws Exception {
        HSSFSheet sheet;
        HSSFCell cell;

        sheet = workbook.createSheet("Spring");
        sheet.setDefaultColumnWidth(12);

        // write a text at A1
        cell = getCell(sheet, 0, 0);
        setText(cell, "Spring-Excel test");
```

```
        cell = getCell(sheet, 2, 0);
        setText(cell, (Date) model.get("serverTime")).toString());
    }
}
```

Sr. No.	Description
(1)	Define `BeanNameViewResolver`. `BeanNameViewResolver` is a class that resolves View by searching for the bean which matches with the returned View name, from application context.
(2)	When `InternalViewResolver` for JSP and `TilesViewResolver` are to be used together, it is recommended to give it a higher priority compared to these `ViewResolver`. In the above example, by specifying `"0"`, View is resolved by `BeanNameViewResolver` prior to `InternalViewResolver`.
(3)	Define `BeanNameViewResolver` using `<mvc:bean-name>` element added from Spring Framework 4.1. When defining `ViewResolver` using `<mvc:view-resolvers>` element, definition order of `ViewResolver` specified in child element will be the priority order. In the above example, by defining it above (`<mvc:jsp>`) element in order to define `InternalViewResolver` for JSP, View is resolved by `BeanNameViewResolver` prior to `InternalViewResolver` for JSP. --- **Note:** When `<mvc:view-resolvers>` element is used, it is possible to define `ViewResolver` in a simple way. Hence, this guideline recommends to use `<mvc:view-resolvers>`. ---
(4)	When View name `"sample/report"` is the return value of handler method, the data generated by View instance which is registered in step (5), is responded as download data.
(5)	Register View object as Bean by specifying View name to the name of component. In above example, `x.y.z.app.views.XxxExcelView` instance is registered as a bean with bean name (view name) as `"sample/report"` .
(6)	Example of View implementation. Implementation of View class that inherits `org.springframework.web.servlet.view.document` and generates Excel data.

Implementing the process

The point here is that **do not implement business logic in controller** .

Business logic must be implemented in Service. Controller must call the service methods in which the business logic is implemented.

Refer to *Domain Layer Implementation* for the details of implementation of business logic.

Note: Controller should be used only for routing purposes (mapping requests to corresponding business logic) and deciding the screen transition for each request as well as setting model data. Thereby, controller should be simple as much as possible. By consistently following this policy, the contents of controller become clear which ensures maintainability of controller even if the size of development is large.

Operations to be performed in controller are shown below:

- *Correlation check of input value*
- *Calling business logic*
- *Reflecting values to domain object*
- *Reflecting values to form object*

Correlation check of input value

Correlation check of input values should be done using `Validation` class which implements `org.springframework.validation.Validator` interface.

Bean Validation can also be used for correlation check of input values.

Refer to *Input Validation* for the details of implementation of correlation check.

The implementation of correlation check itself should not be written in the handler method of controller. However, it is necessary to add the `Validator` to

`org.springframework.web.bind.WebDataBinder`.

```
@Inject
PasswordEqualsValidator passwordEqualsValidator; // (1)

@InitBinder
protected void initBinder(WebDataBinder binder){
    binder.addValidators(passwordEqualsValidator); // (2)
}
```

Sr. No.	Description
(1)	Inject `Validator` that performs correlation check.
(2)	Add the injected `Validator` to `WebDataBinder`. Adding the above to `WebDataBinder` enables correlation check by executing `Validator` before the handler method gets called.

Calling business logic

Execute business logic by injecting the Service in which business logic is implemented and calling the injected Service method.

```
@Inject
SampleService sampleService; // (1)

@RequestMapping("hello")
public String hello(Model model){
    String message = sampleService.hello(); // (2)
    model.addAttribute("message", message);
    return "sample/hello";
}
```

Sr. No.	Description
(1)	Inject the `Service` in which business logic is implemented.
(2)	Call the injected `Service` method to execute business logic.

Reflecting values to domain object

In this guideline, it is recommended to bind the data sent by HTML form to form object instead of the domain object.

Therefore, the controller should perform the process of reflecting the values of form object to domain object which is then passed to the method of service class.

```
@RequestMapping("hello")
public String hello(@Validated SampleForm form, BindingResult result, Model␣
↪model){
    // omitted
    Sample sample = new Sample(); // (1)
    sample.setField1(form.getField1());
    sample.setField2(form.getField2());
    sample.setField3(form.getField3());
    // ...
    // and more ...
    // ...
    String message = sampleService.hello(sample); // (2)
    model.addAttribute("message", message); // (3)
    return "sample/hello";
}
```

Sr. No.	Description
(1)	Create domain object and reflect the values bound to form object in the domain object.
(2)	Call the method of service class to execute business logic.
(3)	Add the data returned from business logic to `Model`.

The process of reflecting values to domain object should be implemented by the handler method of controller. However considering the readability of processing

method in case of large amount of code, it is recommended to delegate the process to Helper class.

Example of delegating the process to Helper class is shown below:

- SampleController.java

```
@Inject
SampleHelper sampleHelper; // (1)

@RequestMapping("hello")
public String hello(@Validated SampleForm form, BindingResult result){
    // omitted
    String message = sampleHelper.hello(form); // (2)
    model.addAttribute("message", message);
    return "sample/hello";
}
```

- SampleHelper.java

```
public class SampleHelper {

    @Inject
    SampleService sampleService;

    public String hello(SampleForm form){ // (3)
        Sample sample = new Sample();
        sample.setField1(form.getField1());
        sample.setField2(form.getField2());
        sample.setField3(form.getField3());
        // ...
        // and more ...
```

```
        // ...
        String message = sampleService.hello(sample);
        return message;
    }
}
```

Sr. No.	Description
(1)	Inject object of Helper class in controller.
(2)	Value is reflected to the domain object by calling the method of the injected Helper class. Delegating the process to Helper class enables to keep the implementation of controller simple.
(3)	Call the Service class method to execute the business logic after creating domain object.

Note: Bean conversion functionality can be used as an alternative way to delegate the process of reflecting form object values, to Helper class. Refer to *Bean Mapping (Dozer)* for the details of Bean conversion functionality.

Reflecting values to form object

In this guideline, it is recommended that form object (and not domain object) must be used to for that data which is to be bound to HTML form.

For this, it is necessary to reflect the values of domain object (returned by service layer) to form object. This conversion should be performed in controller class.

```
@RequestMapping("hello")
public String hello(SampleForm form, BindingResult result, Model model){
    // omitted
    Sample sample = sampleService.getSample(form.getId()); // (1)
    form.setField1(sample.getField1()); // (2)
    form.setField2(sample.getField2());
    form.setField3(sample.getField3());
    // ...
```

```
    // and more ...
    // ...
    model.addAttribute(sample); // (3)
    return "sample/hello";
}
```

Sr. No.	Description
(1)	Call the method of service class in which business logic is implemented and fetch domain object.
(2)	Reflect values of acquired domain object to form object.
(3)	When there are fields only for display, add domain object to `Model` so that data can be referred.

Note: In JSP, it is recommended to refer the values from domain object instead of form object for the fields to be only displayed on the screen.

The process of reflecting value to form object should be implemented by the handler method of controller. However considering the readability of handler method in case of large amount of code, it is recommended to delegate the process to Helper class method.

- SampleController.java

```
@RequestMapping("hello")
public String hello(@Validated SampleForm form, BindingResult result){
    // omitted
    Sample sample = sampleService.getSample(form.getId());
    sampleHelper.applyToForm(sample, form); // (1)
    model.addAttribute(sample);
    return "sample/hello";
}
```

- SampleHelper.java

```
public void applyToForm(SampleForm destForm, Sample srcSample){
    destForm.setField1(srcSample.getField1()); // (2)
    destForm.setField2(srcSample.getField2());
    destForm.setField3(srcSample.getField3());
    // ...
    // and more ...
```

```
    // ...
}
```

Sr. No.	Description
(1)	Call the method to reflect the values of domain object to form object.
(2)	Reflect the values of domain object to form object.

Note: Bean conversion functionality can be used as an alternative method to delegate the process to Helper class. Refer to *Bean Mapping (Dozer)* for the details of Bean conversion functionality.

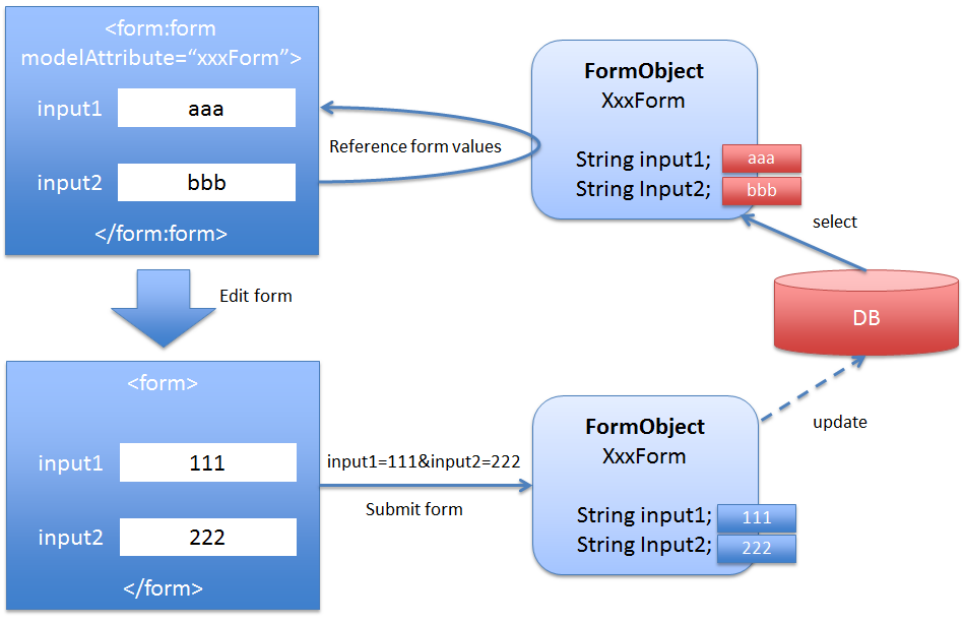
3.4.2 Implementing form object

Form object is the object (JavaBean) which represents HTML form and plays the following role.

1. **Holds business data stored in the database so that it can be referred by HTML form (JSP).**
2. **Holds request parameters sent by HTML form so that they can be referred by handler method of controller.**

Implementation of form object can be described by focusing on the following points.

- *Creating form object*
- *Initializing form object*
- *Binding to HTML form*
- *Binding request parameters*



Creating form object

Create form object as a JavaBean. Spring Framework provides the functionality to convert and bind the request parameters (string) sent by HTML form to the format defined in form object. Hence, the fields to be defined in form object need not only be in `java.lang.String` format.

```
public class SampleForm implements Serializable {
    private String id;
    private String name;
    private Integer age;
    private String genderCode;
    private Date birthDate;
    // omitted getter/setter
}
```

Tip: Regarding the mechanism provided by Spring Framework that performs format conversion

Spring Framework executes format conversion using the following 3 mechanisms and supports conversion to basic format as standard. Refer to linked page for the details of each conversion function.

- Spring Type Conversion
- Spring Field Formatting
- java.beans.PropertyEditor implementations

> **Warning:** In form object, it is recommended to maintain only the fields of HTML form and not the fields which are just displayed on the screen. If display only fields are also maintained in form object, more memory will get consumed at the time of storing form object in HTTP session object causing memory exhaustion. In order to display the values of display only fields on the screen, it is recommended to add objects of domain layer (such as Entity) to request scope by using (`Model.addAttribute`).

Number format conversion of fields

Number format can be specified for each field using `@NumberFormat` annotation.

```
public class SampleForm implements Serializable {
    @NumberFormat(pattern = "#,#") // (1)
    private Integer price;
    // omitted getter/setter
}
```

Sr. No.	Description
(1)	Specify the number format of request parameter sent by HTML form. For example, binding of value formatted by ”,” is possible since “”#, #”” format is specified as pattern. When value of request parameter is “”1,050””, Integer object of “”1050”” will bind to the property `price` of form object.

Attributes of `@NumberFormat` annotation are given below.

Sr. No.	Attribute name	Description
1.	style	Specify number format style. For details refer to Javadoc of NumberFormat.Style.
2.	pattern	Specify number format of Java. Refer to ‘Javadoc of DecimalFormat <http://docs.oracle.com/javase/8/docs/api/java/text/DecimalFormat.html>’_for details.

Date and time format conversion of fields

Date and time format for each field can be specified using `@DateTimeFormat` annotation.

```
public class SampleForm implements Serializable {
    @DateTimeFormat(pattern = "yyyyMMdd") // (1)
    private Date birthDate;
    // omitted getter/setter
}
```

Sr. No.	Description
(1)	Specify the date and time format of request parameter sent by HTML form. For example, `"yyyyMMdd"` format is specified as pattern. When the value of request parameter is `"20131001"`, Date object of 1st October, 2013 will bind to property `birthDate` of form object.

Attributes of `@DateTimeFormat` annotation are given below.

Sr. No.	Attribute name	Description
1.	iso	Specify ISO date and time format. For details refer to Javadoc of DateTimeFormat.ISO.
2.	pattern	Specify Java date and time format. Refer to 'Javadoc of SimpleDateFormat <http://docs.oracle.com/javase/8/docs/api/java/text/SimpleDateFormat.html>'_for details.
3.	style	Specify style of date and time as two-digit string. First digit will be style of date and second digit will be style of time. Values that can be specified as style are given below. S : Format same as `java.text.DateFormat.SHORT`. M : Format same as `java.text.DateFormat.MEDIUM`. L : Format same as `java.text.DateFormat.LONG`. F : Format same as `java.text.DateFormat.FULL`. - : A style meaning omissions. Example of specification and conversion) MM : Dec 9, 2013 3:37:47 AM M- : Dec 9, 2013 -M : 3:41:45 AM

DataType conversion in controller

`@InitBinder` annotation can be used to define datatype conversions at controller level.

```
@InitBinder // (1)
public void initWebDataBinder(WebDataBinder binder) {
    binder.registerCustomEditor(
            Long.class,
            new CustomNumberEditor(Long.class, new DecimalFormat("#,#"),␣
↪true)); // (2)
}
```

```
@InitBinder("sampleForm") // (3)
public void initSampleFormWebDataBinder(WebDataBinder binder) {
    // ...
}
```

Sr. No.	Description
(1)	If a method with `@InitBinder` annotation is provided, it is called before executing the binding process and thereby default operations can be customized.
(2)	For example, `"#.  #"` format is specified for a field of type Long. This enables binding of value formatted with ",".
(3)	Default operation for each form object can be customized by specifying it in the value attribute of `@InitBinder` annotation. In the above example, the method is called before binding form object `"sampleForm"`.

Specifying annotation for input validation

Since form object is validated using Bean Validation, it is necessary to specify the annotation which indicates constraints of the field. Refer to *Input Validation* for the details of input validation.

Initializing form object

Form object can also be called as form-backing bean and binding can be performed using `@ModelAttribute` annotation. Initialize form-backing bean by the method having `@ModelAttribute` annotation. In this guideline, such methods are called as ModelAttribute methods and defined with method names like `setUpXxxForm`.

```
@ModelAttribute // (1)
public SampleForm setUpSampleForm() {
    SampleForm form = new SampleForm();
    // populate form
    return form;
}
```

```
@ModelAttribute("xxx") // (2)
public SampleForm setUpSampleForm() {
    SampleForm form = new SampleForm();
    // populate form
    return form;
}
```

```
@ModelAttribute
public SampleForm setUpSampleForm(
        @CookieValue(value = "name", required = false) String name, // (3)
        @CookieValue(value = "age", required = false) Integer age,
        @CookieValue(value = "birthDate", required = false) Date birthDate) {
    SampleForm form = new SampleForm();
    form.setName(name);
    form.setAge(age);
    form.setBirthDate(birthDate);
    return form;
}
```

Sr. No.	Description
(1)	Class name beginning with lower case letter will become the attribute name to add to `Model`. In the above example, `"sampleForm"` is the attribute name. The returned object is added to `Model` and an appropriate process `model.addAttribute(form)` is executed.
(2)	When attribute name is to be specified to add to `Model`, specify it in the value attribute of `@ModelAttribute` annotation. In the above example, `"xxx"` is the attribute name. For returned object, appropriate process “model.addAttribute(“xxx”, form)”is executed and it is returned to `Model`. When attribute name other than default value is specified, it is necessary to specify `@ModelAttribute("xxx")` at the time of specifying form object as an argument of handler method.
(3)	ModelAttribute method can pass the parameters required for initialization as with the case of handler method. In the above example, value of cookie is specified using `@CookieValue` annotation.

Note: When form object is to be initialized with default values, it should be done using ModelAttribute method. In point (3) in above example , value is fetched from cookie, However, fixed value defined in constant class can be set directly.

Note: Multiple ModelAttribute methods can be defined in the controller. Each method is executed before calling handler method of controller.

Warning: If ModelAttribute method is executed for each request, initialization needs to be repeated for each request and unnecessary objects will get created. So, for form objects which are required only for specific requests, should be created inside handler method of controller and not through the use of ModelAttribute method.

Binding to HTML form

It is possible to bind form object added to the `Model` to HTML form(JSP) using `<form:xxx>` tag.

For the details of `<form:xxx>` tag, refer to Using Spring's form tag library.

```
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %> <!-
↪- (1) -->
```

```
<form:form modelAttribute="sampleForm"
           action="${pageContext.request.contextPath}/sample/hello"> <!-- (2)␣
↪-->
    Id          : <form:input path="id" /><form:errors path="id" /><br /> <!--␣
↪(3) -->
    Name        : <form:input path="name" /><form:errors path="name" /><br />
    Age         : <form:input path="age" /><form:errors path="age" /><br />
    Gender      : <form:input path="genderCode" /><form:errors path="genderCode
↪" /><br />
    Birth Date : <form:input path="birthDate" /><form:errors path="birthDate"␣
↪/><br />
</form:form>
```

Sr. No.	Description
(1)	Define taglib to use `<form:form>` tag.
(2)	Specify form object stored in `Model` in the `modelAttribute` attribute of `<form:form>` tag.
(3)	Specify property name of form object in path attribute of `<form:input>` tag.

Binding request parameters

It is possible to bind the request parameters sent by HTML form to form object and pass it as an argument to the handler method of controller.

```
@RequestMapping("hello")
public String hello(
        @Validated SampleForm form, // (1)
```

```
        BindingResult result,
        Model model) {
    if (result.hasErrors()) {
        return "sample/input";
    }
    // process form...
    return "sample/hello";
}
```

```
@ModelAttribute("xxx")
public SampleForm setUpSampleForm() {
    SampleForm form = new SampleForm();
    // populate form
    return form;
}

@RequestMapping("hello")
public String hello(
        @ModelAttribute("xxx") @Validated SampleForm form, // (2)
        BindingResult result,
        Model model) {
    // ...
}
```

Sr. No.	Description
(1)	Form object is passed as an argument to the handler method of controller after reflecting request parameters to the form object.
(2)	When the attribute name is specified in ModelAttribute method, it is necessary to explicitly specify attribute name of form object as `@ModelAttribute("xxx")`.

Warning: When attribute name specified by ModelAttribute method and attribute name specified in the `@ModelAttribute("xxx")` in the argument of handler method are different, it should be noted that a new instance is created other than the instance created by ModelAttribute method. When attribute name is not specified with `@ModelAttribute` in the argument to handler method, the attribute name is deduced as the class name with first letter in lower case.

Determining binding result

Error (including input validation error) that occurs while binding request parameter sent by HTML form to form object, is stored in `org.springframework.validation.BindingResult`.

```
@RequestMapping("hello")
public String hello(
        @Validated SampleForm form,
        BindingResult result, // (1)
        Model model) {
    if (result.hasErrors()) { // (2)
        return "sample/input";
    }
    // ...
}
```

Sr. No.	Description
(1)	When `BindingResult` is declared immediately after form object, it is possible to refer to the error inside the handler method of controller.
(2)	Calling `BindingResult.hasErrors()` can determine whether any error occurred in the input values of form object.

It is also possible to determine field errors, global errors (correlated check errors at class level) separately. These can be used separately if required.

Sr. No.	Method	Description
1.	`hasGlobalErrors()`	Method to determine the existence of global errors.
2.	`hasFieldErrors()`	Method to determine the existence of field errors.
3.	`hasFieldErrors(String field)`	Method to determine the existence of errors related to specified field.

3.4.3 Implementing View

View plays the following role.

1. **View generates response (HTML) as per the requirements of the client.**

View retrieves the required data from model (form object or domain object) and generates response in the format which is required by the client for rendering.

Implementing JSP

Implement View using JSP to generate response(HTML) as per the requirement of the client.

Use the class provided by Spring Framework as the `ViewResolver` for calling JSP. Refer to *HTML response* for settings of `ViewResolver`.

Basic implementation method of JSP is described below.

- *Creating common JSP for include*
- *Displaying value stored in model*
- *Displaying numbers stored in model*
- *Displaying date and time stored in model*
- *Generate request URL*
- *Binding form object to HTML form*
- *Displaying input validation errors*
- *Displaying message of processing result*
- *Displaying codelist*
- *Displaying fixed text content*
- *Switching display according to conditions*
- *Repeated display of collection elements*
- *Displaying link for pagination*
- *Switching display according to authority*

In this chapter, usage of main JSP tag libraries are described. However, refer to respective documents for the detailed usage since all JSP tag libraries are not described here.

Sr. No.	JSP tag library name	Document
1.	Spring's form tag library	• http://docs.spring.io/spring/docs/4.3.5.RELEASE/spring-framework-reference/html/view.html#view-jsp-formtaglib • http://docs.spring.io/spring/docs/4.3.5.RELEASE/spring-framework-reference/html/spring-form-tld.html
2.	Spring's tag library	• http://docs.spring.io/spring/docs/4.3.5.RELEASE/spring-framework-reference/html/spring-tld.html
3.	JSTL	• http://download.oracle.com/otndocs/jcp/jstl-1.2-mrel2-eval-oth-JSpec/
4.	Common library's tags & el functions	• [*JSP Tag Libraries and EL Functions offered by common library*] of this guideline

Warning: If terasoluna-gfw-web 1.0.0.RELEASE is being used, `action` tag must be always be specified while using `<form:form>` tag of Spring's form tag library.

terasoluna-gfw-web 1.0.0.RELEASE has a dependency on Spring MVC(3.2.4.RELEASE). In this version of Spring MVC, if `action` attribute of `<form:form>` tag is not specified, it will expose a vulnerability of XSS(Cross-site scripting). For further details regarding the vulnerability, refer to CVE-2014-1904 of National Vulnerability Database (NVD).

Also, terasoluna-gfw-web 1.0.1.RELEASE have been upgraded to Spring MVC(3.2.10.RELEASE and above); hence this vulnerability is not present.

Creating common JSP for include

Create a JSP that contains directive declaration which are required by all the JSP files of the project. By specifying this JSP in `<jsp-config>/<jsp-property-group>/<include-prelude>` element of `web.xml`, eliminates the need to declare these directives and each and every JSP file of the project. Further, this file is provided in blank project also.

- include.jsp

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%> <%-- (1) --%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt"%>

<%@ taglib uri="http://www.springframework.org/tags" prefix="spring"%> <%--␣
↪(2) --%>
<%@ taglib uri="http://www.springframework.org/tags/form" prefix="form"%>
<%@ taglib uri="http://www.springframework.org/security/tags" prefix="sec"%>

<%@ taglib uri="http://terasoluna.org/functions" prefix="f"%> <%-- (3) --%>
<%@ taglib uri="http://terasoluna.org/tags" prefix="t"%>
```

- web.xml

```
<jsp-config>
    <jsp-property-group>
        <url-pattern>*.jsp</url-pattern>
        <el-ignored>false</el-ignored>
        <page-encoding>UTF-8</page-encoding>
        <scripting-invalid>false</scripting-invalid>
        <include-prelude>/WEB-INF/views/common/include.jsp</include-prelude>
↪<!-- (4) -->
    </jsp-property-group>
</jsp-config>
```

Sr. No.	Description
(1)	JSP tag libraries of JSTL are declared. In this example, `core` and `fmt` are used.
(2)	JSP tag libraries of Spring Framework are declared. In this example, `spring`, `form` and `sec` are used.
(3)	JSP tag libraries provided by common library are declared.
(4)	The contents specified in JSP to be included (`/WEB-INF/views/common/include.jsp`) are included at the beginning of each JSP (file specified in `<url-pattern>`).

Note: Refer to `JSP.1.10 Directives` of JavaServer Pages Specification(Version2.2)for details of directives.

Note: Refer to `JSP.3.3 JSP Property Groups` of JavaServer Pages Specification(Version2.2)for details of <jsp-property-group> element.

Displaying value stored in model

To display the value stored in `Model` (form object or domain object) in HTML, use EL expressions or JSP tag libraries provided by JSTL.

Display using EL expressions.

- SampleController.java

```
@RequestMapping("hello")
public String hello(Model model) {
    model.addAttribute(new HelloBean("Bean Hello World!")); // (1)
    return "sample/hello"; // returns view name
}
```

- hello.jsp

```
Message : ${f:h(helloBean.message)} <%-- (2) --%>
```

Sr. No.	Description
(1)	Add `HelloBean` object to `Model` object.
(2)	In View(JSP), data added to the `Model` object can be retrieved by describing `${Attribute name.Property name of JavaBean}`. In this example, HTML escaping is performed using `${f:h(Attribute name.Property name of JavaBean)}` function of EL expression.

Note: Since HTML escaping function (`f:h`) is provided in the common components, always use it if EL expressions are used to output values in HTML. For details of function of EL expression that perform HTML escaping, refer to *Cross Site Scripting*.

Display using <c:out> tag provided by JSP tag library of JSTL.

```
Message : <c:out value="${helloBean.message}" /> <%-- (1) --%>
```

Sr. No.	Description
(1)	Specify the values fetched using EL expressions in value attribute of <c:out> tag. HTML escaping is also performed.

Note: Refer to CHAPTER 4 General-Purpose Actions of JavaServer Pages Standard Tag Library(Version 1.2)for the details of <c:out>.

Displaying numbers stored in model

Use JSP tag library provided by JSTL to output format number.

Display using <fmt:formatNumber> tag provided by JSP tag library of JSTL.

```
Number Item : <fmt:formatNumber value="${helloBean.numberItem}" pattern="0.00
↪" /> <%-- (1) --%>
```

Sr. No.	Description
(1)	Specify the value acquired by EL expressions in the value attribute of <fmt:formatNumber> tag. Specify the format to be displayed in pattern attribute. For example, "0.00" is specified . When the value acquired in ${helloBean.numberItem} is "1.2" temporarily, "1.20" is displayed on the screen.

Note: Refer to CHAPTER 9 Formatting Actions of JavaServer Pages Standard Tag Library(Version 1.2)for the details of <fmt:formatNumber>.

Displaying date and time stored in model

Use JSP tag library provided by JSTL to output format date and time value.

Display using <fmt:formatDate> tag provided by JSP tag library of JSTL.

```
Date Item : <fmt:formatDate value="${helloBean.dateItem}" pattern="yyyy-MM-dd
↪" /> <%-- (1) --%>
```

Sr. No.	Description
(1)	Specify the value fetched using EL expression in value attribute of <fmt:formatDate> tag. Specify the format to be displayed in pattern attribute. In this example, "yyyy-MM-dd" is specified. When value received for ${helloBean.dateItem} is 2013-3-2, "2013-03-02" is displayed on the screen.

Note: Refer to CHAPTER 9 Formatting Actions of JavaServer Pages Standard Tag Library(Version 1.2) for details of <fmt:formatDate>.

Note: JSP tag library provided by Joda Time should be used to use org.joda.time.DateTime as date and time object type. Refer to *Date Operations (Joda Time)* for the details of Joda Time.

Generate request URL

When a request URL (a URL for calling Controller method) is to be set for action attribute of <form> element(<form:form> element of JSP tag library) of HTML and href attribute of <a> element, a URL is generated using either of the methods described below.

- Build a request URL as a character string
- Build a request URL using EL function which has been added from Spring Framework 4.1

Note: Although either of these methods can be used, using both the methods together in a single application should be avoided since it may result in decrease in maintainability.

An implementation sample of the Controller method used further in the description is shown.

In the description hereafter, an implementation method is described wherein a request URL is generated for calling the method given below.

```
package com.example.app.hello;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;

@RequestMapping("hello")
@Controller
public class HelloController {

    // (1)
    @RequestMapping({"", "/"})
    public String hello() {
        return "hello/home";
    }

}
```

Sr. No.	Description
(1)	Request URL assigned to the method is "`{Context path}/hello`".

Building a request URL as a character string

First, the method to build a request URL using a character string is explained.

```
<form action="${pageContext.request.contextPath}/hello"> <!-- (2) -->
    <!-- ... -->
```

```
</form>
```

Sr. No.	Description
(2)	Fetch context path assigned to Web application from `pageContext` (implicit object of JSP)(`${pageContext.request.contextPath}`), add servlet path (`/hello` in the example above) assigned to the Controller method that has been called, after the context path.

Tip:

- `<c:url>` offered by JSTL
- `<spring:url>` offered by Spring Framework

are available as JSP tag libraries which build URL. A request URL can also be built using these JSP tag libraries.

When it is necessary to build a request URL dynamically, a URL is preferably built by using these JSP tag libraries.

Building a request URL by using EL function added from Spring Framework 4.1

Next, a method to build a request URL using EL function (`spring:mvcUrl`) added from Spring Framework 4.1 is described.

If `spring:mvcUrl` function is used, a request URL can be built by linking with meta information of Controller method (method signature and annotation etc).

```
<form action="${spring:mvcUrl('HC#hello').build()}"> <!-- (3) -->
    <!-- ... -->
</form>
```

Sr. No.	Description
(3)	Specify a request mapping name assigned to the Controller method which has been called, in the `spring:mvcUrl` function argument. An object of class (`MvcUriComponentsBuilder.MethodArgumentBuilder`) which builds a request URL is returned from `spring:mvcUrl` function. `MvcUriComponentsBuilder.MethodArgumentBuilder` class offers following methods • `arg` method • `build` method • `buildAndExpand` method which play roles described below respectively. • `arg` method specifies the value to be passed to argument of Controller method. • `build` method generates a request URL. • `buildAndExpand` method generates a request URL after specifying a value embedded in dynamic part (path variable etc) which has not been declared as an argument of Controller method. In the example above, since a request URL is static, a request URL is generated only by calling `build` method. When the request URL is dynamic (a URL wherein a path variable and a query string exists), `arg` method and `buildAndExpand` method must be called. Refer to "Spring Framework Reference Documentation(Building URIs to Controllers and methods from views)" for the basic usage of `arg` method and `buildAndExpand` method.

Note: Regarding request mapping name

In the default implementation (`org.springframework.web.servlet.mvc.method.RequestMappingInfoHa` implementation), the request mapping name is represented as "Class name in upper case (short name of class) + "#" + method name".

Request mapping name must not be duplicated. When the name is duplicated, a unique name must be specified in `name` attribute of `@RequestMapping` annotation.

When a request mapping name assigned to Controller method is to be verified, settings given below should be added to `logback.xml`.

```
<logger name="org.springframework.web.servlet.mvc.method.annotation.
↪RequestMappingHandlerMapping">
    <level value="trace" />
</logger>
```

If the program is rebooted after applying the settings given above, the log shown below is output.

```
date:2014-12-09 18:34:29        thread:RMI TCP Connection(2)-127.0.0.1  X-
↪Track:        level:TRACE     logger:o.s.w.s.m.m.a.
↪RequestMappingHandlerMapping       message:Mapping name=HC#hello
```

Binding form object to HTML form

Use JSP tag library provided by Spring Framework to bind form object to HTML form and to display the values stored in form object.

Bind using `<form:form>` tag provided by Spring Framework.

```
<form:form action="${pageContext.request.contextPath}/sample/hello"
           modelAttribute="sampleForm"> <%-- (1) --%>
    Id : <form:input path="id" /> <%-- (2) --%>
</form:form>
```

Sr. No.	Description
(1)	Specify attribute name of form object stored in `Model` in `modelAttribute` attribute of `<form:form>` tag.
(2)	Specify name of property to bind in the `path` attribute of `<form:xxx>` tag. `xxx` part changes along with each input element.

Note: For the details of `<form:form>` , `<form:xxx>` tag refer to Using Spring's form tag library.

Displaying input validation errors

To display the contents of input validation error, use JSP tag library provided by Spring Framework.

Display using `<form:errors>` tag provided by Spring Framework.

Refer to *Input Validation* for details.

```
<form:form action="${pageContext.request.contextPath}/sample/hello"
           modelAttribute="sampleForm">
    Id : <form:input path="id" /><form:errors path="id" /><%-- (1) --%>
</form:form>
```

Sr. No.	Description
(1)	Specify name of the property to display the error in `path` attribute of `<form:errors>` tag.

Displaying message of processing result

To display the message notifying the output of processing the request, use JSP tag library provided in common components.

Use `<t:messagesPanel>` tag provided in common components.

Refer to *Message Management* for details.

```
<div class="messages">
    <h2>Message pattern</h2>
    <t:messagesPanel /> <%-- (1) --%>
</div>
```

Sr. No.	Description
(1)	Messages stored with attribute name "`resultMessages`" are output.

Displaying codelist

To display the codelist (provided in common components), use JSP tag library provided by Spring Framework.

Codelist can be referred from JSP in the same way as `java.util.Map` interface.

Refer to *Codelist* for details.

Display codelist in select box.

```
<form:select path="orderStatus">
    <form:option value="" label="--Select--" />
    <form:options items="${CL_ORDERSTATUS}" /> <%-- (1) --%>
</form:select>
```

Sr. No.	Description
(1)	Codelist (`java.util.Map` interface) is stored with name (`"CL_ORDERSTATUS"`) as attribute name. Therefore, in JSP, codelist (`java.util.Map` interface) can be accessed using EL expression. Codelist can be displayed in select box by passing the object of `Map` interface to `items` attribute of `<form:options>`.

Label part is displayed on the screen for the value selected in select box.

```
Order Status : ${f:h(CL_ORDERSTATUS[orderForm.orderStatus])}
```

Sr. No.	Description
(1)	In the same way as in case of creating select box, Codelist (`java.util.Map` interface) is stored with name (`"CL_ORDERSTATUS"`) as attribute name. If value selected in select box is specified as key of the fetched `Map` interface, it is possible to display the code name.

Displaying fixed text content

Strings for screen name, element name and guidance etc can be directly written in JSP when internationalization is not required.

However, when internationalization is required, display the values acquired from property file using JSP tag library provided by Spring Framework.

Display using `<spring:message>` tag provided by Spring Framework.

Refer to *Internationalization* for details.

- properties

```
# (1)
label.orderStatus=Order status
```

- jsp

```
<spring:message code="label.orderStatus" text="Order Status" /> : <%-- (2) --
↪%>
    ${f:h(CL_ORDERSTATUS[orderForm.orderStatus])}
```

Sr. No.	Description
(1)	Define the string of label in properties file.
(2)	If key in the properties file is specified in `code` attribute of `<spring:message>`, string corresponding to the key is displayed.

Note: The value specified in `text` attribute is displayed when property value could not be acquired.

Switching display according to conditions

When display is to be switched according to some value in model, use JSP tag library provided by JSTL.

Switch display using `<c:if>` tag or `<c:choose>` provided by JSP tag library of JSTL.

Switching display by using `<c:if>`.

```
<c:if test="${orderForm.orderStatus != 'complete'}"> <%-- (1) --%>
        <%-- ... --%>
</c:if>
```

Sr. No.	Description
(1)	Put the condition for entering the branch in `test` attribute of `<c:if>`. In this example, when order status is not `'complete'`, the contents of the branch will be displayed.

Switching display using `<c:choose>`.

```
<c:choose>
    <c:when test="${customer.type == 'premium'}"> <%-- (1) --%>
        <%-- ... --%>
    </c:when>
    <c:when test="${customer.type == 'general'}">
        <%-- ... --%>
```

```
    </c:when>
    <c:otherwise> <%-- (2) --%>
        <%-- ... --%>
    </c:otherwise>
</c:choose>
```

Sr. No.	Description
(1)	Put the condition for entering the branch in `test` attribute of `<c:when>`. In this example, when customer type is `'premium'`, the contents of the branch will be displayed. When condition specified in `test` attribute is `false`, next `<c:when>` tag is processed.
(2)	When result of `test` attribute of all `<c:when>` tags is `false`, `<c:otherwise>` tag is evaluated.

Note: Refer to `CHAPTER 5 Conditional Actions` of JavaServer Pages Standard Tag Library(Version 1.2)for details.

Repeated display of collection elements

To repeat display of collection stored in model, use JSP tag library provided by JSTL.

Repeated display can be done using `<c:forEach>` provided by JSP tag library of JSTL.

```
<table>
    <tr>
        <th>No</th>
        <th>Name</th>
    </tr>
    <c:forEach var="customer" items="${customers}" varStatus="status"> <%--␣
↪(1) --%>
        <tr>
            <td>${status.count}</td> <%-- (2) --%>
            <td>${f:h(customer.name)}</td> <%-- (3) --%>
        </tr>
    </c:forEach>
</table>
```

Sr. No.	Description
(1)	By specifying the collection object in `items` attribute of `<c:forEach>` tag, `<c:forEach>` tag is repeatedly executed to iterate over the collection. While iterating over the collection, if the current element is to be referred inside `<c:forEach>` tag, it can be done by specifying a variable name for the current element in `var` attribute.
(2)	By specifying a variable name in `varStatus` attribute, current position (count) of iteration in `<c:forEach>` tag can be fetched. Refer to JavaDocof `javax.servlet.jsp.jstl.core.LoopTagStatus` for attributes other than count.
(3)	This is the value acquired from the object stored in variable specified by `var` attribute of `<c:forEach>` tag.

Note: Refer to `CHAPTER 6 Iterator Actions` of JavaServer Pages Standard Tag Library(Version 1.2)for details.

Displaying link for pagination

To display links of pagination on the screen while displaying the list, use JSP tag library provided in common components.

Display the link for pagination using `<t:pagination>` provided in common components. Refer to *Pagination* for details.

Switching display according to authority

To switch display according to authority of the user who has logged in, use JSP tag library provided by Spring Security.

Switch display using `<sec:authorize>` provided by Spring Security. Refer to *Authorization* for details.

Implementing JavaScript

When it is necessary to control screen items (controls of hide/display, activate/deactivate, etc.) after screen rendering, control items using JavaScript.

Todo

TBD

Details will be included in the coming versions.

Implementing style sheet

It is recommended to specify attribute values related to screen design in style sheet (css file) and not in JSP(HTML) directly.

In JSP(HTML), specify `id` attribute to identify the items uniquely and `class` attribute which indicates classification of elements.

While, specify values related to actual location of elements or appearance should be specified in style sheet (css file).

By configuring in this way, it is possible to reduce the design related aspects from the implementation of JSP.

Simultaneously, if there is any change in design, only style sheet (css file) is modified and not JSP.

Note: When form is created using `<form:xxx>`, `id` attribute will be set automatically. Application developer should specify `class` attribute.

3.4.4 Implementing common logic

Implementing common logic to be executed before and after calling controller

Here, common logic indicates processes which are required to be executed before and after execution the controller.

Implementing Servlet Filter

Common processes independent of Spring MVC are implemented using Servlet Filter.

However, when common processes are to be executed only for the certain requests mapped with handler method of controller,

then it should be implemented using Handler Interceptor and not Servlet Filter.

Samples of Servlet Filter are given below.

In sample code, value is stored in `MDC` in order to output the log of IP address of client.

- java

```
public class ClientInfoPutFilter extends OncePerRequestFilter { // (1)

    private static final String ATTRIBUTE_NAME = "X-Forwarded-For";
    protected final void doFilterInternal(HttpServletRequest request,
            HttpServletResponse response, FilterChain filterChain) throws␣
↪ServletException, IOException {
        String remoteIp = request.getHeader(ATTRIBUTE_NAME);
        if (remoteIp == null) {
            remoteIp = request.getRemoteAddr();
        }
        MDC.put(ATTRIBUTE_NAME, remoteIp);
        try {
            filterChain.doFilter(request, response);
        } finally {
            MDC.remove(ATTRIBUTE_NAME);
        }
```

```
    }
}
```

- web.xml

```
<filter> <!-- (2) -->
    <filter-name>clientInfoPutFilter</filter-name>
    <filter-class>x.y.z.ClientInfoPutFilter</filter-class>
</filter>
<filter-mapping> <!-- (3) -->
    <filter-name>clientInfoPutFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

Sr. No.	Description
(1)	In sample, it is guaranteed that it is executed only once for similar requests by creating the Servlet Filter as subclass of `org.springframework.web.filter.OncePerRequestFilter` provided by Spring Framework.
(2)	Register the created Servlet Filter in `web.xml`.
(3)	Specify URL pattern to apply the registered Servlet Filter.

Servlet Filter can also be defined as Bean of Spring Framework.

- web.xml

```
<filter>
    <filter-name>clientInfoPutFilter</filter-name>
    <filter-class>org.springframework.web.filter.DelegatingFilterProxy</
↪filter-class> <!-- (1) -->
</filter>
<filter-mapping>
    <filter-name>clientInfoPutFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

- applicationContext.xml

```
<bean id="clientInfoPutFilter" class="x.y.z.ClientInfoPutFilter" /> <!-- (2) -
↪->
```

Sr. No.	Description
(1)	In sample, process is delegated to Servlet Filter defined in step (2) by specifying `org.springframework.web.filter.DelegatingFilterProxy` provided by Spring Framework in Servlet Filter class.
(2)	Add the Servlet Filter class to Bean definition file (`applicationContext.xml`). At this time, `id` attribute of bean definition should be assigned with the filter name (value specified in `<filter-name>` tag) specified in `web.xml`.

Implementing HandlerInterceptor

Common processes dependent on Spring MVC are implemented using HandlerInterceptor.

HandlerInterceptor can execute common processes only for the requests which are allowed by the application since it is called after the determining the handler method

to which the request is to be mapped to.

HandlerInterceptor can execute the process keeping in mind the following 3 points.

- Before executing handler method of controller

  Implemented as `HandlerInterceptor#preHandle` method.

- After successfully executing handler method of controller

  Implemented as `HandlerInterceptor#postHandle` method.

- After completion of handler method of controller (executed irrespective of Normal / Abnormal)

  Implemented as `HandlerInterceptor#afterCompletion` method.

Sample of HandlerInterceptor is given below.

In sample code, log of info level is output after successfully executing controller process.

```
public class SuccessLoggingInterceptor extends HandlerInterceptorAdapter { //␣
↪(1)

    private static final Logger logger = LoggerFactory
            .getLogger(SuccessLoggingInterceptor.class);

    @Override
```

```
    public void postHandle(HttpServletRequest request,
            HttpServletResponse response, Object handler,
            ModelAndView modelAndView) throws Exception {
        HandlerMethod handlerMethod = (HandlerMethod) handler;
        Method m = handlerMethod.getMethod();
        logger.info("[SUCCESS CONTROLLER] {}.{}", new Object[] {
                m.getDeclaringClass().getSimpleName(), m.getName()});
    }

}
```

- spring-mvc.xml

```
<mvc:interceptors>
    <!-- ... -->
    <mvc:interceptor>
        <mvc:mapping path="/**" /> <!-- (2) -->
        <mvc:exclude-mapping path="/resources/**" /> <!-- (3) -->
        <mvc:exclude-mapping path="/**/*.html" />
        <bean class="x.y.z.SuccessLoggingInterceptor" /> <!-- (4) -->
    </mvc:interceptor>
    <!-- ... -->
</mvc:interceptors>
```

Sr. No.	Description
(1)	In sample, HandlerInterceptor is created as the subclass of `org.springframework.web.servlet.handler.HandlerInterceptorAdapter` provided by Spring Framework. Since `HandlerInterceptorAdapter` provides blank implementation of `HandlerInterceptor` interface, it is not required to implement unnecessary methods in the subclass.
(2)	Specify the pattern of path, where the created HandlerInterceptor is to be applied.
(3)	Specify the pattern of path, where the created HandlerInterceptor need not be applied.
(4)	Add the created HandlerInterceptor to `<mvc:interceptors>` tag of `spring-mvc.xml`.

Implementing common processes of controller

Here, common process indicates the process that should be commonly implemented in all the controllers.

Implementing HandlerMethodArgumentResolver

When an object that is not supported by default in Spring Framework is to be passed as controller argument, HandlerMethodArgumentResolver is to be implemented in order to enable controller to be able to receive the argument.

Sample of HandlerMethodArgumentResolver is given below.

In sample code, common request parameters are converted to JavaBean which are then passed to handler method of controller.

- JavaBean

```
public class CommonParameters implements Serializable { // (1)

    private String param1;
    private String param2;
    private String param3;

    // ....

}
```

- HandlerMethodArgumentResolver

```
public class CommonParametersMethodArgumentResolver implements
 ↪HandlerMethodArgumentResolver { // (2)

    @Override
    public boolean supportsParameter(MethodParameter parameter) {
        return CommonParameters.class.equals(parameter.getParameterType()); //
 (3)
    }

    @Override
    public Object resolveArgument(MethodParameter parameter,
            ModelAndViewContainer mavContainer, NativeWebRequest webRequest,
            WebDataBinderFactory binderFactory) throws Exception {
        CommonParameters params = new CommonParameters(); // (4)
        params.setParam1(webRequest.getParameter("param1"));
        params.setParam2(webRequest.getParameter("param2"));
        params.setParam3(webRequest.getParameter("param3"));
```

```
        return params;
    }
```

- Controller

```
@RequestMapping(value = "home")
public String home(CommonParameters commonParams) { // (5)
    logger.debug("param1 : {}",commonParams.getParam1());
    logger.debug("param2 : {}",commonParams.getParam2());
    logger.debug("param3 : {}",commonParams.getParam3());
    // ...
    return "sample/home";

}
```

- spring-mvc.xml

```
<mvc:annotation-driven>
    <mvc:argument-resolvers>
        <!-- ... -->
        <bean class="x.y.z.CommonParametersMethodArgumentResolver" /> <!--␣
↪(6) -->
        <!-- ... -->
    </mvc:argument-resolvers>
</mvc:annotation-driven>
```

Sr. No.	Description
(1)	JavaBean that retains common parameters.
(2)	Implement `org.springframework.web.method.support.HandlerMethodArgumentResol` interface.
(3)	Determine parameter type. For example, when type of JavaBean that retains common parameters is specified as argument of controller, resolveArgument method of this class is called.
(4)	Fetch the values of request parameters, set the parameters and return the JavaBean that retains the value of common parameters.
(5)	Specify JavaBean that retains common parameters in the argument of handler method of controller. Object returned in step (4) is passed.
(6)	Add the created HandlerMethodArgumentResolver to `<mvc:argument-resolvers>` tag of `spring-mvc.xml`.

Note: When parameters are to be passed commonly to handler methods of all controllers, it is effective to convert the parameters to JavaBean using HandlerMethodArgumentResolver. Parameters referred here are not restricted to request parameters.

Implementing "@ControllerAdvice"

In a class with `@ControllerAdvice` annotation, implement common processes which are to be executed in multiple Controllers.

When a class with `@ControllerAdvice` annotation is created, processes implemented using

- method with `@InitBinder`

- method with `@ExceptionHandler`
- method with `@ModelAttribute`

can be applied to multiple Controllers.

Tip: `@ControllerAdvice` annotation is a mechanism added from Spring Framework 3.2; however, since the processing was applied to all Controllers, it could only implement common processes of entire application.

From Spring Framework 4.0, it has been improved in such a way that controller can be specified flexibly for applying common processes. With this improvement, it is possible to implement a common process in various granularities.

Methods to specify Controller (methods to specify attributes) for applying common processes are described below.

Sr. No.	Attribute	Description and example of specification

(1)	annotations	Specify annotation.
		Common processes are applied for the Controllers with specified annotations. An example of specifications is given below.
		`@ControllerAdvice(annotations = LoginFormModelAttributeSetter.Lo`
		`public class LoginFormModelAttributeSetter {`
		`  @Target(ElementType.TYPE)`
		`  @Retention(RetentionPolicy.RUNTIME)`
		`  public static @interface LoginFormModelAttribute {}`
		`  // ...`
		`}`
		`@LoginFormModelAttribute`
		`@Controller`
		`public class WelcomeController {`
		`  // ...`
		`}`
		`@LoginFormModelAttribute`
		`@Controller`
		`public class LoginController {`
		`  // ...`
		`}`
		In the above example, `@LoginFormModelAttribute` annotation is assigned to `WelcomeController` and `LoginController`, hence, common processes are applied to `WelcomeController` and `LoginController`.

次のページに続く

Table. 3.14 – 前のページからの続き

Sr. No.	Attribute	Description and example of specification

(2)	assignableTypes	Specify a class or an instance.
		Common processes are applied for the Controllers which can be assigned (can be casted) to the specified class or interface. When using this attribute, it is recommended to adopt a style that specifies a marker interface to indicate that it is a Controller using common process, in attribute value. When this style is adopted, only the marker interface for common processes to be used needs to be implemented at Controller side. An example of specifications is given below.
		`@ControllerAdvice(assignableTypes = ISODateInitBinder.ISODateApp`
		`public class ISODateInitBinder {`
		`  public static interface ISODateApplicable {}`
		`  // ...`
		`}`
		`@Controller`
		`public class SampleController implements ISODateApplicable {`
		`  // ...`
		`}`
		In the above example, `SampleController` implements `@ISODateApplicable` interface (Marker interface), hence, common processes are applied to `SampleController`.

次のページに続く

Table. 3.14 – 前のページからの続き

Sr. No.	Attribute	Description and example of specification

(3)	basePackageClasses	Specify a class or an interface.
		Common processes are applied for the Controllers under the package of specified class or interface.
		When using this attribute, it is recommended to adopt a style that specifies,
		• a class with `@ControllerAdvice`
		• a marker interface to identify the package
		in attribute value. An example to specify the same is given below.
		`package com.example.app`
		
		`@ControllerAdvice(basePackageClasses = AppGlobalExceptionHandler`
		`public class AppGlobalExceptionHandler {`
		`    // ...`
		`}`
		`package com.example.app.sample`
		
		`@Controller`
		`public class SampleController {`
		`    // ...`
		`}`
		In the above example, `SampleController` is stored under the package (`com.example.app`) in which (`AppGlobalExceptionHandler`) class with `@ControllerAdvice`is stored. Hence, common processes are applied to `SampleController`.
		`package com.example.app.common`
		
		`@ControllerAdvice(basePackageClasses = AppPackage.class)`
		`public class AppGlobalExceptionHandler {`
		`    // ...`
		`}`
		`package com.example.app`
		
		`public interface AppPackage {`
		`}`
		When package level of class with `@ControllerAdvice` is different than the class where Controller is stored, or when common processes are to be applied to multiple base packages, it is better to create a marker interface to identify the packages.

次のページに続く

Table. 3.14 – 前のページからの続き

Sr. No.	Attribute	Description and example of specification

(4)	`basePackages`	Specify the package name.
		Common processes are applied for the Controllers under the specified package. An example of specifying the same is given below.
		`@ControllerAdvice(basePackages = "com.example.app")`
		`public class AppGlobalExceptionHandler {`
		`    // ...`
		`}`

(5)	`value`	Alias to `basePackages`.
		The operation is same as when `basePackages` attribute is specified. An example of specifying the same is given below.
		`@ControllerAdvice("com.example.app")`
		`public class AppGlobalExceptionHandler {`
		`    // ...`
		`}`

Tip: `basePackageClasses` attribute / `basePackages` attribute / `value` attribute are the attributes to specify base package that stores the Controller for applying common processes. However, when `basePackageClasses` attribute is used,

- It is possible to prevent specifying the package that does not exist.
- It is possible to get linked and change the package name on IDE.

Therefore, it is known as Type-safe specification method.

Implementation sample of `@InitBinder` method is given below.

In sample code, date that can be specified in request parameter is set to `"yyyy/MM/dd"` .

```
@ControllerAdvice // (1)
@Order(0) // (2)
public class SampleControllerAdvice {

    // (3)
    @InitBinder
    public void initBinder(WebDataBinder binder) {
        SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy/MM/dd");
        dateFormat.setLenient(false);
        binder.registerCustomEditor(Date.class,
                new CustomDateEditor(dateFormat, true));
    }

}
```

Sr. No.	Description
(1)	It indicates that it is Bean of ControllerAdvice by assigning the `@ControllerAdvice` annotation.
(2)	Specify priority for common processes by assigning the `@Order` annotation. It should be specified when multiple ControllerAdvice have ordering for example there are dependencies between them. Otherwise it is not necessary.
(3)	Implement `@InitBinder` method. `@InitBinder` method is applied to all Controllers.

Implementation sample of `@ExceptionHandler` method is given below.

In sample code, View of lock error screen is returned by handling `org.springframework.dao.PessimisticLockingFailureException`.

```
// (1)
@ExceptionHandler(PessimisticLockingFailureException.class)
public String handlePessimisticLockingFailureException(
        PessimisticLockingFailureException e) {
    return "error/lockError";
}
```

Sr. No.	Description
(1)	Implement `@ExceptionHandler` method. `@ExceptionHandler` method is applied to all Controllers.

Implementation sample of `@ModelAttribute` method is given below.

In sample code, common request parameters are converted to JavaBean and stored in `Model`.

- ControllerAdvice

```
// (1)
@ModelAttribute
public CommonParameters setUpCommonParameters(
        @RequestParam(value = "param1", defaultValue="def1") String param1,
        @RequestParam(value = "param2", defaultValue="def2") String param2,
        @RequestParam(value = "param3", defaultValue="def3") String param3) {
    CommonParameters params = new CommonParameters();
    params.setParam1(param1);
    params.setParam2(param2);
    params.setParam3(param3);
    return params;
}
```

- Controller

```
@RequestMapping(value = "home")
public String home(@ModelAttribute CommonParameters commonParams) { // (2)
    logger.debug("param1 : {}",commonParams.getParam1());
    logger.debug("param2 : {}",commonParams.getParam2());
    logger.debug("param3 : {}",commonParams.getParam3());
    // ...
    return "sample/home";
}
```

Sr. No.	Description
(1)	Implement `@ModelAttribute` method. `@ModelAttribute` method is applied to all Controllers.
(2)	Object created by `@ModelAttribute` method is passed.

3.4.5 Prevention of double submission

Measures should be taken to prevent double submission as the same process gets executed multiple times by clicking Send button multiple times or refreshing (refresh using F5 button) the Finish screen.

For the problems occurring when countermeasures are not taken and details of implementation method, refer to *Double Submit Protection* .

3.4.6 Usage of session

In the default operations of Spring MVC, model (form object, domain object etc.) is not stored in session.

When it is to be stored in session, it is necessary to assign `@SessionAttributes` annotation to the controller class.

When input forms are split on multiple screens, usage of `@SessionAttributes` annotation should be studied since model (form object, domain object etc.)

can be shared between multiple requests for executing a series of screen transitions.

However, whether to use `@SessionAttributes` annotation should be determined after confirming the warning signs of using the session.

For details of session usage policy and implementation at the time of session usage, refer to *Session Management* .

3.5 Build development project

3.5.1 Build development project

The method to create a war file to be deployed on application server and a jar file of env module (module to store the file environment dependent file) is described below.

In case of a project created using Maven Archetype, the following 2 methods are provided as methods to create a war file.

- *Build method wherein jar file of env module is not included in war file* (**recommended**)
- *Build method wherein jar file of env module is included in war file*

Note: About the recommended build method

This guideline recommends *Build method wherein jar file of env module is not included in war file*. Other build method apart from those mentioned here can also be used.

However, **the war file and jar file to be released in test environment and production environment should not be created using the functionality provided by IDE such as Eclipse.** In some of the IDE functionalities like Eclipse, class files are created using an independent compiler which has been optimized for development, hence there could be a risk of unexpected error during the application execution due to difference in the compiler.

Note: Rule for the project structure

Consider the following project structure.

1. Make sure to have a multi-project structure. 2.As far as possible, consolidate the configuration files (ex. logback.xml, jdbc.properties) having environment dependency in one project. **Hereafter, this project is expressed as *-env.**

- ex. terasoluna-tourreservation-env

3. Projects other than *-env never have a setting for environment dependency.

- However, it is allowed to store the environment dependency configuration files for test under src/test/resources.

4. Manage all software packaged binaries by storing them in package repository.

- Deploy not only *.jar files but also *.war files as work products in package repository. Consequently, these jar/war files should not include environment dependencies.

5. Configure *-env project as follows:

- Store the configuration values required for operations on the developer's PC in the file under src/main/resources, as default values.
- Store the configuration files that differ with each environment such as test server, production server, in the folder other than src/main/resources (ex. configs/test-server). Then, use "profile" function of Maven that automatically replaces the configuration values depending on environment in order to build *-env-x.y.z.jar file.

The above structure facilitates proper development in all the scenarios of software development lifecycle.

1. In local development environment, check out both main project and *-env project and include env project in build path of the main project so as to do coding and testing in the local development environment.
2. On CI server, execute the test and perform packaging using a build tool (Maven) and deploy the artifact in package repository whenever required.
3. Application can work on test server and production server by adding *-env project built as per the application release environment, to the main project which is stored in package repository.

For details, refer to Sample application.

Warning: About build environment

In the example below, Windows environment is used for the build. However, you can use your own environment for doing the build. This guideline **recommends that you should do the build using the same OS and JDK version as that of the application execution environment.**

When build is done using Maven, confirm whether home directory of JDK which is used during compilation in the environment variable JAVA_HOME, has been specified.

If the environment variable is not set or the home directory of JDK having different version has been specified, an appropriate home directory should be specified in environment variable.

[In case of Windows]

```
echo %JAVA_HOME%
set JAVA_HOME={Please set home directory of JDK}
```

[In case of Linux]

```
echo $JAVA_HOME
JAVA_HOME={Please set home directory of JDK}
```

Note: It is advisable to set the environment variable JAVA_HOME in the user environment variable of OS user wherein build is to be done.

Build method wherein jar file of env module is not included in war file

Create war file

Open the root directory of development project.

```
cd C:\work\todo
```

Specify `warpack` in Maven profile (`-P` parameter) and run Maven install.

```
mvn -P warpack clean install
```

If the Maven package is run successfully, a war file that does not include jar file of env module is created in the target directory of web module.

(Example: `C:\work\todo\todo-web\target\todo-web.war`)

Note: About the goal to be specified

In the above example, `install` is specified in goal and war file is installed in local repository, however it is advisable to specify

- `package` in goal when only creating a war file

- `deploy` in goal when deploying in remote repository like Nexus

Create jar file of env module

Open env module directory.

```
cd C:\work\todo\todo-env
```

Specify **Profile ID to identify environment** in Maven profile (`-P` parameter) and run Maven package.

```
mvn -P test-server clean package
```

If Maven package is run successfully, jar file for the specified environment is created in target directory of env module.

(Example:
`C:\work\todo\todo-env\target\todo-env-1.0.0-SNAPSHOT-test-server.jar`)

Note: About profile ID to identify environment

In case of a project created using Maven Archetype, following profile IDs are defined by default.

- `local`: Profile for local environment of the developer (for IDE development environment) (default profile)
- `test-server`: Profile for test environment
- `production-server`: Profile for production environment

The above 3 profiles are provided by default; however you can add or modify them as per the environment configuration of the system to be developed.

Build method wherein jar file of env module is included in war file

Create war file

Warning: Points to be noted when including a jar file of env module in war file

When jar file of env module is included in war file, the war file cannot be deployed in other environment; hence war file should be managed so that it is not deployed to other environment (especially in production environment) by mistake.

Moreover, when using a method in which war file is created for each environment and released in each environment, it should be noted that war file released in production environment can never be the war file for which testing is complete. This is for the re-compilation at the time of creating war file for the production environment. When creating the war file and releasing the same for each environment, it is especially important to use the VCS (Version Control System) functionality (Tag functionality etc.) like Git or Subversion and to establish a mechanism to create a war file which is to be released in production environment and various test environments, through the use of tested source files.

Open the root directory of development project.

```
cd C:\work\todo
```

In Maven profile (`-P` parameter), specify **Profile ID to identify environment**defined in env module and `warpack-with-env`, and then run the Maven package.

```
mvn -P warpack-with-env,test-server clean package
```

If Maven package is run successfully, war file which includes jar file of env module is created in target directory of web module.

(Example: `C:\work\todo\todo-web\target\todo-web.war`)

Deploy

Deploy on Tomcat

Deployment method (procedure) when Tomcat is used as an application server is given below.

1. Specify the profile of Maven as per the AP server environment in which the application is to be released and build *-env project.
2. Place *-env-x.y.z.jar file built above in the folder of AP server decided in advance. ex. /etc/foo/bar/abcd-env-x.y.z.jar
3. Unjar the *.war file deployed in package repository under [CATALINA_HOME]/webapps.
4. If Tomcat 7 is used, add /etc/foo/bar/*.jar into class path using VirtualWebappLoader function of the Tomcat.
   - The following definition should be added in [CATALINA_HOME]/conf/[contextPath].xml file.
   - For details, refer to http://tomcat.apache.org/tomcat-7.0-doc/api/org/apache/catalina/loader/VirtualWebappLoader.html and configs folder of terasoluna-tourreservation-env.
   - Example of VirtualWebappLoader function usages :

```
<Loader className="org.apache.catalina.loader.VirtualWebappLoader"
        virtualClasspath="/etc/foo/bar/*.jar" />
```

   - In addition, VirtualWebappLoader can also be used in the Tomcat 6.
5. If Tomcat 8.x is used, add /etc/foo/bar/*.jar into class path using Resource function of the Tomcat.
   - The following definition should be added in [CATALINA_HOME]/conf/[contextPath].xml file.
   - For details, refer to https://tomcat.apache.org/migration-8.html#Web_application_resources and configs folder of terasoluna-tourreservation-env.
   - Example of Resource function usages :

```
<Resources className="org.apache.catalina.webresources.StandardRoot">
  <PreResources className="org.apache.catalina.webresources.DirResourceSet"
                base="/etc/foo/bar/"
                internalPath="/"
```

```
                webAppMount="/WEB-INF/lib" />
</Resources>
```

Note:

- autoDeploy attribute of Host tag of [CATALINA_HOME]/conf/server.xml should be set to false. Otherwise [CATALINA_HOME]/conf/[contextPath].xml gets deleted each time web application is restarted.
- When autoDeploy is disabled, Web application does not start by just placing the war file in [CATALINA_HOME]/webapps. war file should always be unjarred (unzipped).

Deployment to other application server

Pprocedure to deployment on other (not tomcat) application server is explained.

When releasing the Web application on application servers (Example: WebSphere, WebLogic, JBoss) where a mechanism for adding a class path for each web application (which is provided in VirtualWebappLoader of Tomcat) is not provided, the method to release it after adding *-env-x.y.z.jar file under WEB-INF/lib of war file is the easiest.

1. Specify profile of Maven as per the AP server environment in which application is to be released and build *-env project.
2. Copy *.war file deployed in the package repository to the working directory.
3. Add it under WEB-INF/lib of war file using add option of jar command as follows.
4. Release foo-x.y.z.war on AP server.

Note: For a method to deploy a war file on application server, refer to the manual of application server to be used.

Here, a method to embed the jar file of env module in war file using jar command is given.

Open the working directory.

Here the in the example below, work is performed in env project.

```
cd C:\work\todo\todo-env
```

Copy the created war file to the working directory.

Here in the example below, war file is fetched from Maven repository. (war file is required to be `installed` or `deployed`.)

```
mvn org.apache.maven.plugins:maven-dependency-plugin:2.5:get^
 -DgroupId=com.example.todo^
 -DartifactId=todo-web^
 -Dversion=1.0.0-SNAPSHOT^
 -Dpackaging=war^
 -Ddest=target/todo-web.war
```

If the command is run successfully, the specified war file is copied to the target directory of env module.

(Example: `C:\work\todo\todo-env\target\todo-web.war`)

Note:

- An appropriate value should be specified in `-DgroupId`, `-DartifactId`, `-Dversion`, `-Ddest`.
- When run on Linux, `^` at the end of the line should be read as `\` .

Copy the created jar file to working directory (`target\WEB-INF\lib`) once and add it to the war file.

[In case of Windows]

```
mkdir target\WEB-INF\lib
copy target\todo-env-1.0.0-SNAPSHOT-test-server.jar target\WEB-INF\lib\.
cd target
jar -uvf todo-web.war WEB-INF\lib
```

[In case of Linux]

```
mkdir -p target/WEB-INF/lib
cp target/todo-env-1.0.0-SNAPSHOT-test-server.jar target/WEB-INF/lib/.
cd target
jar -uvf todo-web.war WEB-INF/lib
```

Note: Measures to be taken when jar command is not found

The problem when jar command is not found can be resolved using either of the following measures.

- Add `JAVA_HOME/bin` to environment variable "PATH".
- Specify the jar command with full path. In case of Windows, `%JAVA_HOME%\bin\jar` and in case of Linux, `${JAVA_HOME}/bin/jar` can be specified.

Continuous deployment

Continuous deployment is constantly releasing the target software through continuous looping of project (source code tree) structure, version control, inspection, build operations and lifecycle management.

During development, release the software of SNAPSHOT version in the package repository and development AP server and execute the test. To release the software officially, tagging to source code tree in VCS needs be performed after assigning a version number. In this way, the flow of build and deployment slightly differs in the snapshot release and official release.

To deploy the application on AP server that provides Web service, irrespective of snapshot version or official release version, a group of environment dependency configuration files and *.war file should be deployed in a set as per the target release AP server environment.

Separating the operation of registering libraries (jar, war) without environment dependency settings, in Maven repository and the operation of actually deploying them on AP server facilitates deployment.

Note: In Maven, it is automatically distinguished whether it is a SNAPSHOT version or RELEASE version according to the contents of <version> tag of pom.xml.

- It is considered as SNAPSHOT if it ends with -SNAPSHOT. Example: <version>1.0-SNAPSHOT</version>

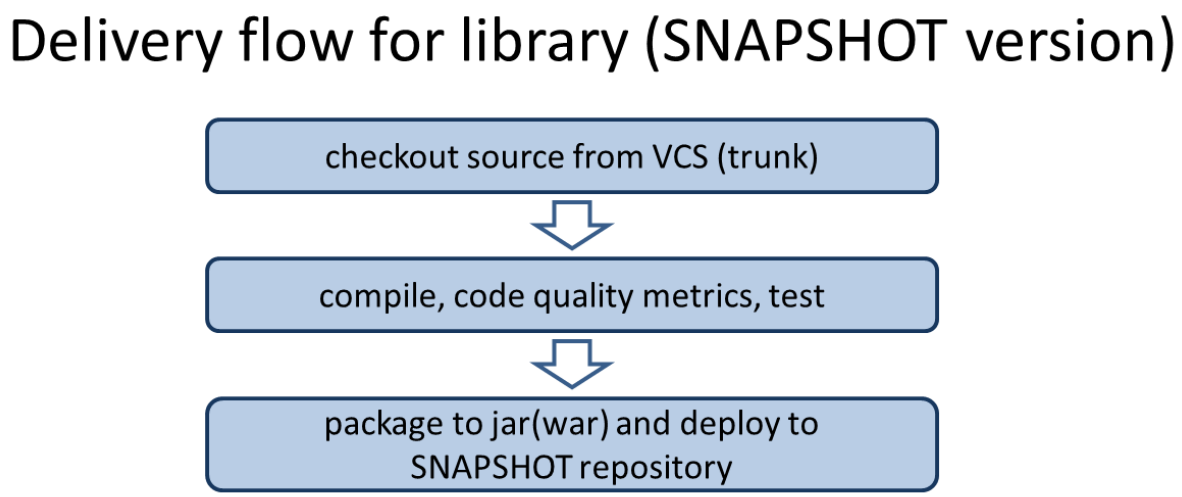
- It is considered as RELEASE if it does not end with -SNAPSHOT. Example: <version>1.0</version>

Please note that there are 2 types of repositories in Maven package repository i.e. snapshot repository and release repository with a few limitations.

- Software of SNAPSHOT version cannot be registered in release repository. release repository also cannot be registered in snapshot repository.
- In release repository, artifact having the same GAV information can be registered only once. (GAV=groupId, artifactId, version)
- In snapshot repository, artifact having the same GAV information can be re-registered many times.

Operations of SNAPSHOT version

A simple delivery flow of SNAPSHOT version software is as shown in the following figure.



1. Check out the source code from development trunk.
2. Compile, measure the code metrics and execute test.

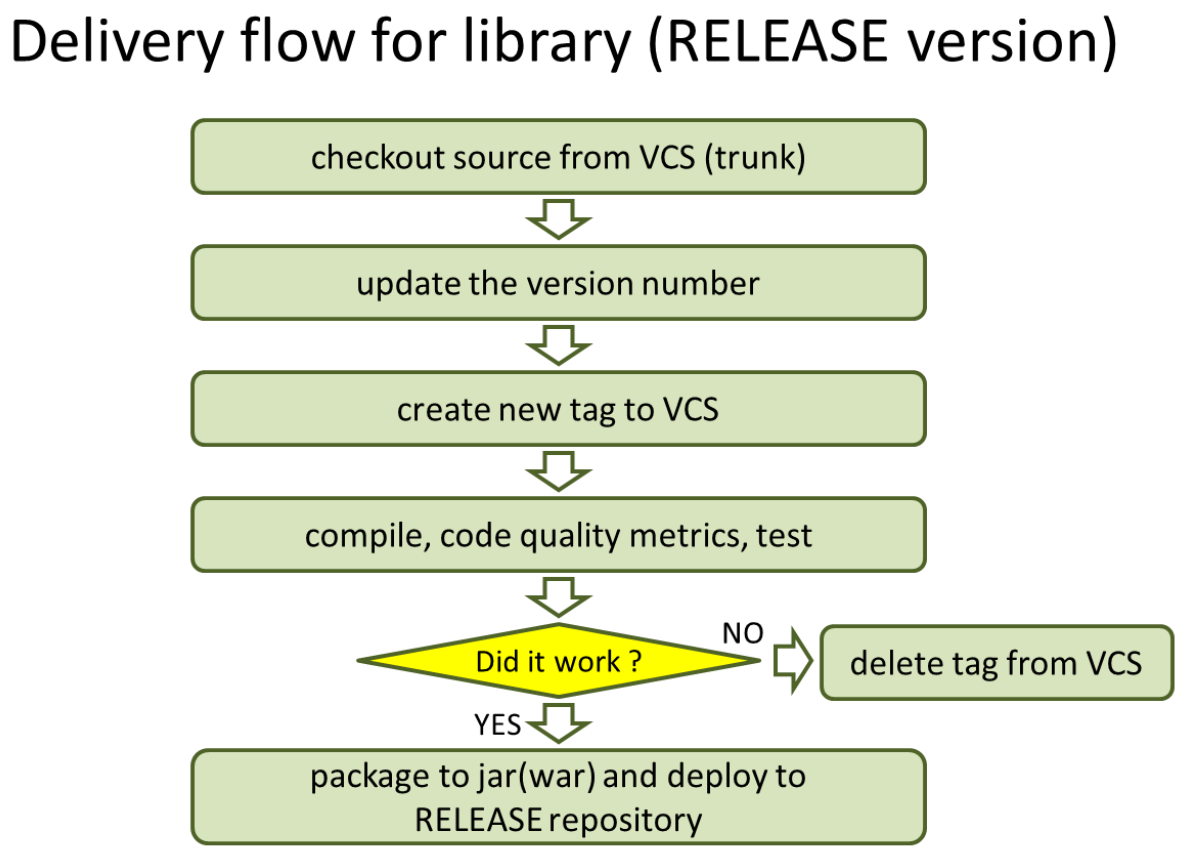
- In case of compilation error, certain violations of code metrics or in case the test fails, the subsequent operations should be stopped.

3. Upload (mvn deploy) the artifact (jar, war file) on Maven package repository server. |

Operations of RELEASE version

In case of official release, since it is necessary to assign the version number, the flow becomes slightly more complex than the SNAPSHOT release.

1. Decide the version number to be assigned for release. (Example：1.0.1)



2. Check out the source code from development trunk (or release branch).
3. Change <version> tag of pom.xml. (Example：<version>1.0.1</version>)
4. Assign tag to VCS. (Example： tags/1.0.1)
5. Compile, measure the code metrics and execute test.
   - In case of compilation error, certain violations of code metrics or in case the test fails, the subsequent operations should be stopped.
   - If the test fails, delete the tag of VCS.
6. Upload (mvn deploy) the artifact (jar, war file) on Maven package repository server.

Todo

Here, should the version tag of pom.xml of trunk source tree be written at the end till it is replaced by the next SNAPSHOT version and committed?

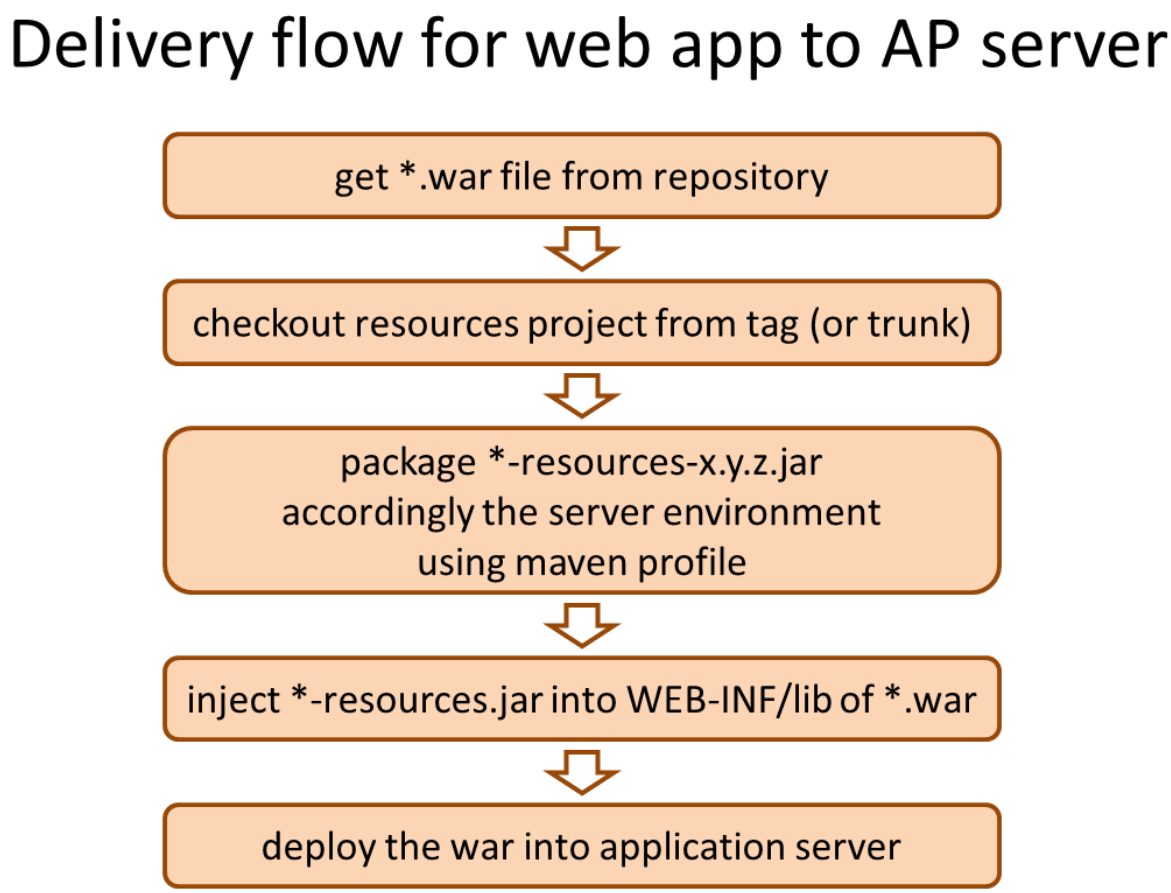
Note: <version> tag of pom.xml file can be changed in versions-maven-plugin .

```
mvn versions:set -DnewVersion=1.0.0
```

Version tag in pom.xml can be edited as <version>1.0.0</version> by the above commands.

Release on Application Server

To release the application on AP server that provides Web service, release the *.war file registered in Maven package repository and the group of environment dependency configuration files in a set according to the target release AP server environment. This has same flow irrespective of snapshot release or official release.



1. Download war file of the version to be released from Maven package repository.
2. Check out *-resources project (that consolidates environment dependency configuration files) from VCS.
3. Using “profile” function of Maven, replace the contents with group of configuration files according to the target release environment, package the resources project and create *-resources-x.y.z.jar.
4. Add the created *-resources-x.y.z.jar file under WEB-INF/lib folder of war file.

- In case of Tomcat, instead of adding *-resources-x.y.z.jar to war file, copy it to any path of Tomcat server

and specify that path in the extended class path of VirtualWebappLoader.

5. Deploy the war file on application server.

Note: War file can be downloaded from Maven package repository with "get goal" of maven-dependency-plugin.

```
mvn org.apache.maven.plugins:maven-dependency-plugin:2.5:get \
 -DgroupId=com.example \
 -DartifactId=mywebapp \
 -Dversion=0.0.1-SNAPSHOT \
 -Dpackaging=war \
 -Ddest=${WORKSPACE}/target/mywebapp.war
```

With this, mywebapp.war file is downloaded under the target directory.

Package of environment dependency configuration files can be added to mywebapp.war file using the following commands.

```
mkdir -p $WORKSPACE/target/WEB-INF/lib
cd $WORKSPACE/target
cp ./mywebapp-resources*.jar WEB-INF/lib
jar -ufv mywebapp.war WEB-INF/lib
```

For the definition of layers, refer to *Application Layering* .

4

Web Application Development Features

The web function architecture adopted in this guideline is explained in detail.

4.1 Input Validation

4.1.1 Overview

It is mandatory to check whether the value entered by the user is correct. Validation of input value is broadly classified into

1. Validation to determine whether the input value is valid by just looking at it irrespective of the context such as size and format.
2. Validation of whether the changes in input value are valid depending on the system status.

1. is an example of mandatory check and number of digits check and 2. is an example of check of whether EMail is registered and whether order count is within the count of the available stock.

In this section, 1. is explained and this check is called “Input validation”. 2. is called “Business logic check”. For business logic check, refer to *Domain Layer Implementation*.

In this guideline, validation check should be performed in application layer whereas business logic check should be performed in domain layer.

Input validation of Web application is performed at server side and client side (JavaScript). It is mandatory to check at the Server Side. However, if the same check is performed at the client side also, usability improves since validation results can be analyzed without communicating with the server.

> **Warning:** Input validation should be performed at the server side as the process at client side may be altered using JavaScript. If the validation is performed only at the client side without performing at the server side, the system may be exposed to danger.

Todo

Input validation at client side will be explained later. Only input validation at the server side is mentioned in the first version.

Classification of input validation

Input validation is classified into single item check and correlation item check.

Type	Description	Example	Implementation method
Single item check	Check completed in single field	Input mandatory check Digit check Type check	Bean Validation (Hibernate Validator is used as implementation library)
Correlation item check	Check comparing multiple fields	Password and confirm password check	Bean Validation or Validation class implementing org.springframework.validation.Validator interface

Spring supports Bean Validation which is a Java standard. This Bean Validation is used for single item check. Bean Validation or `org.springframework.validation.Validator` interface provided by Spring is used for correlation item check.

4.1.2 How to use

Adding dependent libraries

When Bean validation 1.1 (Hibernate Validator 5.x) or higher version is to be used, in addition to jar file of Hibernate Validator and jar file storing API specifications class (`javax.validation` package class) of Bean Validation, libraries that store the following classes are required.

- API specifications class of Expression Language 2.2 or higher version (`javax.el` package class)
- Reference implementation class of Expression Language 2.2 or higher version

If the file is run by deploying on application server, dependent libraries need not be added; since these libraries are provided by application server. However, when it is run in standalone environment (JUnit etc.), these libraries need to be added as dependent libraries.

An example of adding libraries which are required when running Bean Validation 1.1 or higher version in standalone environment is given below.

```
<!-- (1) -->
<dependency>
    <groupId>org.apache.tomcat.embed</groupId>
    <artifactId>tomcat-embed-el</artifactId>
    <scope>test</scope> <!-- (2) -->
</dependency>
```

Sr. No.	Description
(1)	Add a library wherein a class for Expression Language is stored, in `pom.xml` file of the project to be run in standalone environment. In the above example, libraries provided for Apache Tomcat to be embedded are specified. API specifications classes of Expression Language and reference implementation classes are both stored in jar file of `tomcat-embed-el`.
(2)	When dependent libraries are required to execute JUnit, an appropriate scope is `test`.

Note: In the above setting example, since it is assumed that the dependent library version is managed by the parent project terasoluna-gfw-parent , specifying the version in pom.xml is not necessary. The above dependent library used by terasoluna-gfw-parent is defined by Spring IO Platform.

Single item check

For the implementation of single item check following points are necessary,

- Bean Validation annotation should be assigned to the field of form class
- `@Validated` annotation should be assigned in Controller for validation
- Tag for displaying validation error message should be added to JSP

Note: `<mvc:annotation-driven>` settings are carried out in spring-mvc.xml, Bean Validation is enabled.

Basic single item check

Implementation method is explained using “New user registration” process as an example. Rules for checking “New user registration” form are provided below.

Field name	Type	Rules
name	`java.lang.String`	Mandatory input Between 1 and 20 characters
email	`java.lang.String`	Mandatory input Between 1 and 50 characters Email format
age	`java.lang.Integer`	Mandatory input Between 1 and 200

- Form class

  Assign Bean Validation annotation to each field of form class.

```
package com.example.sample.app.validation;

import java.io.Serializable;

import javax.validation.constraints.Max;
import javax.validation.constraints.Min;
import javax.validation.constraints.NotNull;
import javax.validation.constraints.Size;

import org.hibernate.validator.constraints.Email;

public class UserForm implements Serializable {

  private static final long serialVersionUID = 1L;

  @NotNull // (1)
  @Size(min = 1, max = 20) // (2)
  private String name;

  @NotNull
  @Size(min = 1, max = 50)
  @Email // (3)
  private String email;
```

```
  @NotNull // (4)
  @Min(0) // (5)
  @Max(200) // (6)
  private Integer age;

  // omitted setter/getter
}
```

Sr. No.	Description
(1)	Assign `javax.validation.constraints.NotNull` indicating that the target field is not `null`. In Spring MVC, when form is sent with input fields left blank, **empty string instead of null binds**to form object by default. This `@NotNull` checks that `name` exists as request parameter.
(2)	Assign `javax.validation.constraints.Size` indicating that the string length (or collection size) of the target field is within the specified size range. Since empty string binds to the field where string is left blank by default in Spring MVC, '1 or more character' rule indicates Mandatory input.
(3)	Assign `org.hibernate.validator.constraints.Email` indicating that the target field is in RFC2822-compliant E-mail format. When E-mail format requirements are flexible than RFC2822-compliant constraints, regular expression should be specified using `javax.validation.constraints.Pattern` instead of `@Email`.
(4)	When form is sent without entering any number in input field, `null` binds to form object so `@NotNull` indicates mandatory input of `age`.
(5)	Assign `javax.validation.constraints.Min` indicating that the target field value must be greater than the specified value.
(6)	Assign `javax.validation.constraints.Max` indicating that the target field value must be less than the specified value.

Tip: Refer to *Bean Validation check rules* and *Hibernate Validator check rules* for standard annotations of

Bean Validation and annotations provided by Hibernate.

Tip: Refer to *Binding null to blank string field* for the method of binding `null` when input field is left blank.

- Controller class

Assign `@Validated` to form class for input validation.

```
package com.example.sample.app.validation;

import org.springframework.stereotype.Controller;
import org.springframework.validation.BindingResult;
import org.springframework.validation.annotation.Validated;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

@Controller
@RequestMapping("user")
public class UserController {

  @ModelAttribute
  public UserForm setupForm() {
    return new UserForm();
  }

  @RequestMapping(value = "create", method = RequestMethod.GET, params = "form
↪")
  public String createForm() {
    return "user/createForm"; // (1)
  }

  @RequestMapping(value = "create", method = RequestMethod.POST, params =
↪"confirm")
  public String createConfirm(@Validated /* (2) */ UserForm form,␣
↪BindingResult /* (3) */ result) {
    if (result.hasErrors()) { // (4)
      return "user/createForm";
    }
    return "user/createConfirm";
  }

  @RequestMapping(value = "create", method = RequestMethod.POST)
  public String create(@Validated UserForm form, BindingResult result) { //␣
↪(5)
    if (result.hasErrors()) {
```

```
      return "user/createForm";
    }
    // omitted business logic
    return "redirect:/user/create?complete";
  }

  @RequestMapping(value = "create", method = RequestMethod.GET, params =
↪"complete")
  public String createComplete() {
    return "user/createComplete";
  }
}
```

Sr. No.	Description
(1)	Display “New user registration” form screen.
(2)	Assign `org.springframework.validation.annotation.Validated` to the form class argument to perform input validation.
(3)	Add `org.springframework.validation.BindingResult` that stores check result of input validation performed in step (2). This `BindingResult` must be specified immediately after the form argument. When not specified immediately, `org.springframework.validation.BindException` is thrown.
(4)	Use `BindingResult.hasErrors()` method to determine the check result of step (2). When the result of `hasErrors()` is `true`, return to form display screen as there is an error in input value.
(5)	**Input validation should be executed again**even at the time of submission on the confirmation screen. There is a possibility of data tempering and hence Input validation must be performed just before entering business logic.

Note: `@Validated` is not a standard Bean Validation annotation. It is an independent annotation provided by Spring. Bean Validation standard `javax.validation.Valid` annotation can also be used. However, `@Validated` is better as compared to `@Valid` annotation. Validation group can be specified in case of `@Validated` and hence `@Validated` is recommended in this guideline.

- JSP

When there is input error, it can be displayed in `<form:errors>` tag.

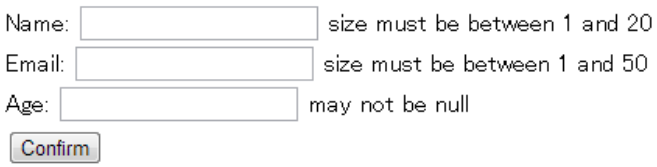
```
<!DOCTYPE html>
<html>
<%-- WEB-INF/views/user/createForm.jsp --%>
<body>
    <form:form modelAttribute="userForm" method="post"
        action="${pageContext.request.contextPath}/user/create">
        <form:label path="name">Name:</form:label>
        <form:input path="name" />
        <form:errors path="name" /><%--(1) --%>
        <br>
        <form:label path="email">Email:</form:label>
        <form:input path="email" />
        <form:errors path="email" />
        <br>
        <form:label path="age">Age:</form:label>
        <form:input path="age" />
        <form:errors path="age" />
        <br>
        <form:button name="confirm">Confirm</form:button>
    </form:form>
</body>
</html>
```

Sr. No.	Description
(1)	Specify target field name in `path` attribute of `<form:errors>` tag. Error message is displayed next to input field of each field.

Form is displayed as follows.



Error message is displayed as follows if this form is sent with all the input fields left blank.



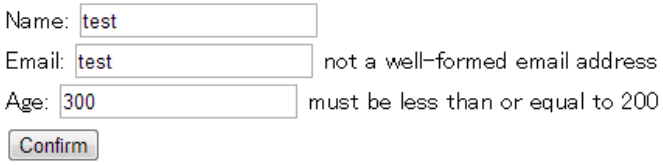
Error messages state that Name and Email are blank and Age is `null`.

Note: In Bean Validation, `null` is a valid input value except the following annotations.

- `javax.validation.constraints.NotNull`
- `org.hibernate.validator.constraints.NotEmpty`
- `org.hibernate.validator.constraints.NotBlank`

In the above example, error messages related to `@Min` and `@Max` annotations are not displayed. This is because `null` is a valid value for `@Min` and `@Max` annotations.

Next, send the form by entering any value in the field.



Error message is not displayed since input value of Name fulfills validation conditions.

Error message is displayed since input value of Email is not in Email format though it fulfills the conditions related to string length.

Error message is displayed since input value of Age exceeds maximum value.

Change the form as follows to change the style at the time of error.

```
<form:form modelAttribute="userForm" method="post"
    class="form-horizontal"
    action="${pageContext.request.contextPath}/user/create">
    <form:label path="name" cssErrorClass="error-label">Name:</form:label><%-- (1)␣
↪--%>
    <form:input path="name" cssErrorClass="error-input" /><%-- (2) --%>
    <form:errors path="name" cssClass="error-messages" /><%-- (3) --%>
```

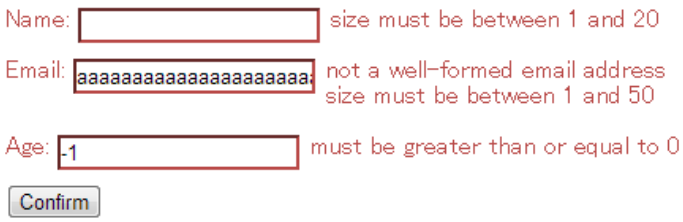
```
    <br>
    <form:label path="email" cssErrorClass="error-label">Email:</form:label>
    <form:input path="email" cssErrorClass="error-input" />
    <form:errors path="email" cssClass="error-messages" />
    <br>
    <form:label path="age" cssErrorClass="error-label">Age:</form:label>
    <form:input path="age" cssErrorClass="error-input" />
    <form:errors path="age" cssClass="error-messages" />
    <br>
    <form:button name="confirm">Confirm</form:button>
</form:form>
```

Sr. No.	Description
(1)	Specify class name for `<label>` tag in `cssErrorClass` attribute at the time of error.
(2)	Specify class name for `<input>` tag in `cssErrorClass` attribute at the time of error.
(3)	Specify class name for error messages in `cssClass` attribute.

For example, if the following CSS is applied to this JSP, error screen is displayed as follows.

```
.form-horizontal input {
    display: block;
    float: left;
}

.form-horizontal label {
    display: block;
    float: left;
    text-align: right;
    float: left;
}

.form-horizontal br {
    clear: left;
}

.error-label {
    color: #b94a48;
}

.error-input {
    border-color: #b94a48;
```

```
    margin-left: 5px;
}

.error-messages {
    color: #b94a48;
    display: block;
    padding-left: 5px;
    overflow-x: auto;
}
```



CSS can be customized as per the requirements of screen.

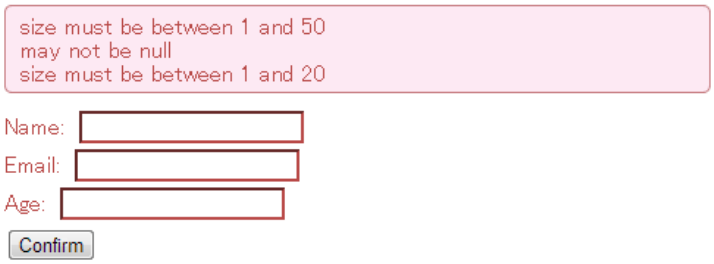
Instead of displaying the error messages next to each input field, output them collectively.

```
<form:form modelAttribute="userForm" method="post"
    action="${pageContext.request.contextPath}/user/create">
    <form:errors path="*" element="div" cssClass="error-message-list" /><%-- (1) --
↪%>

    <form:label path="name" cssErrorClass="error-label">Name:</form:label>
    <form:input path="name" cssErrorClass="error-input" />
    <br>
    <form:label path="email" cssErrorClass="error-label">Email:</form:label>
    <form:input path="email" cssErrorClass="error-input" />
    <br>
    <form:label path="age" cssErrorClass="error-label">Age:</form:label>
    <form:input path="age" cssErrorClass="error-input" />
    <br>
    <form:button name="confirm">Confirm</form:button>
</form:form>
```

Sr. No.	Description
(1)	By specifying `*` in `path` attribute of `<form:errors>` in `<form:form>` tag, all error messages related to Model specified in `modelAttribute` attribute of `<form:form>` can be output. Tag name including these error messages can be specified in `element` attribute. By default, it is `span`. However, specify `div` to output error message list as block element. Specify CSS class in `cssClass` attribute.

An example of error message is shown when the following CSS class is applied.

```
.form-horizontal input {
    display: block;
    float: left;
}

.form-horizontal label {
    display: block;
    float: left;
    text-align: right;
    float: left;
}

.form-horizontal br {
    clear: left;
}

.error-label {
    color: #b94a48;
}

.error-input {
    border-color: #b94a48;
    margin-left: 5px;
}

.error-message-list {
    color: #b94a48;
    padding:5px 10px;
    background-color: #fde9f3;
    border:1px solid #c98186;
    border-radius:5px;
    margin-bottom: 10px;
}
```



By default, field name is not included in error message, hence it is difficult to understand which error message corresponds to which field.

Therefore, when an error message is to be displayed in a list, it is necessary to define the message such that field name is included in the error message.

For method of defining error messages, refer to "*Definition of error messages*".

Note: Points to be noted when displaying error messages in a list

Error messages are output in a random order and the output order cannot be controlled by standard function. Therefore, when an output order needs to be controlled (to be kept constant), an extended implementation such as sorting the error information, etc. is required.

In the method of "Displaying error messages in a list",

- Error message definition in feed unit
- Extended implementation to control the output order of error messages

are required. Therefore, the cost is higher as compared to "displaying error messages next to input field". **This guideline recommends the method of "displaying error messages next to input field" when there are no constraints due to screen requirements.**

Further, following method can be considered as extended methods to control output order of error message. Creating inherited class of `org.springframework.validation.beanvalidation.LocalValidatorFactoryBean` provided by Spring Framework, and sorting error information by overriding `processConstraintViolations` method, etc.

Note: About @GroupSequence annotation

A mechanism of @GroupSequence annotation is provided to control the check sequence; however, add a note that this mechanism is not to control the output order of error message as operations given below are performed.

- When an error occurs, checking for subsequent groups is not executed.

- If multiple errors (errors in multiple fields) occur in the check of identical groups, then the output order of error messages would be random.

Note: Use `<spring:nestedPath>` tag to display error messages collectively outside the `<form:form>` tag.

```
<spring:nestedPath path="userForm">
    <form:errors path="*" element="div"
        cssClass="error-message-list" />
</spring:nestedPath>
<hr>
<form:form modelAttribute="userForm" method="post"
    action="${pageContext.request.contextPath}/user/create">
    <form:label path="name" cssErrorClass="error-label">Name:</form:label>
    <form:input path="name" cssErrorClass="error-input" />
    <br>
    <form:label path="email" cssErrorClass="error-label">Email:</form:label>
    <form:input path="email" cssErrorClass="error-input" />
    <br>
    <form:label path="age" cssErrorClass="error-label">Age:</form:label>
    <form:input path="age" cssErrorClass="error-input" />
    <br>
    <form:button name="confirm">Confirm</form:button>
</form:form>
```

Date and time format check

In case of performing the date and time format check, it is recommend the use of `@DateTimeFormat` annotation offered by Spring rather than the use of Bean Validation mechanism.

About how to use the `@DateTimeFormat` annotation, refer *Date and time format conversion of fields*.

It is also possible to check the date and time format using `@Pattern` annotation of the Bean Validation.

However, it is necessary to write the date and time format in regular expressions while using `@Pattern` annotation. If you want to check the date and time that does not exist, the description is complicated.

Therefore `@DateTimeFormat` annotation is more simpler than the `@Pattern` annotation.

Since `@DateTimeFormat` annotation is one of the type conversion mechanism provided by Spring, instead of the error messages of Bean Validation, the error message of the type mismatch exception (`TypeMismatchException`) has been displayed on screen at the time of input error.

In order to avoid the actual exception message gets displayed on the screen, it is necessary to configure the error message in the **property file** which will be displayed at the time of type mismatch is occurred.

For more detail, refer *Type mismatch*.

Single item check of nested Bean

The method to validate nested Bean using Bean Validation is explained below.

"Ordering" process of an EC site is considered as an example. Rules for checking "Order" form are provided below.

Field name	Type	Rules	Description
coupon	`java.lang.String`	5 or less characters Single byte alphanumeric characters	Coupon code
receiverAddress.name	`java.lang.String`	Mandatory input Between 1 and 50 characters	Receiver name
receiverAddress.postcode	`java.lang.String`	Mandatory input Between 1 and 10 characters	Receiver postal code
receiverAddress.address	`java.lang.String`	Mandatory input Between 1 and 100 characters	Receiver address
senderAddress.name	`java.lang.String`	Mandatory input Between 1 and 50 characters	Sender name
senderAddress.postcode	`java.lang.String`	Mandatory input Between 1 and 10 characters	Sender postal code
senderAddress.address	`java.lang.String`	Mandatory input Between 1 and 100 characters	Sender address

Use the same form class since `receiverAddress` and `senderAddress` are objects of the same class.

- Form class

```
package com.example.sample.app.validation;

import java.io.Serializable;

import javax.validation.Valid;
import javax.validation.constraints.NotNull;
import javax.validation.constraints.Pattern;
import javax.validation.constraints.Size;

public class OrderForm implements Serializable {
    private static final long serialVersionUID = 1L;

    @Size(max = 5)
    @Pattern(regexp = "[a-zA-Z0-9]*")
    private String coupon;

    @NotNull // (1)
    @Valid // (2)
    private AddressForm receiverAddress;

    @NotNull
    @Valid
    private AddressForm senderAddress;

    // omitted setter/getter
}
```

```
package com.example.sample.app.validation;

import java.io.Serializable;

import javax.validation.constraints.NotNull;
import javax.validation.constraints.Size;

public class AddressForm implements Serializable {
    private static final long serialVersionUID = 1L;

    @NotNull
    @Size(min = 1, max = 50)
    private String name;

    @NotNull
    @Size(min = 1, max = 10)
    private String postcode;

    @NotNull
```

```
    @Size(min = 1, max = 100)
    private String address;

    // omitted setter/getter
}
```

Sr. No.	Description
(1)	This indicates that the child form is mandatory. When not set, it will be considered as valid even if `null` is set in `receiverAddress`.
(2)	Assign `javax.validation.Valid` annotation to enable Bean Validation of the nested Bean.

- Controller class

It is not different from the Controller described earlier.

```
package com.example.sample.app.validation;

import org.springframework.stereotype.Controller;
import org.springframework.validation.BindingResult;
import org.springframework.validation.annotation.Validated;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

@RequestMapping("order")
@Controller
public class OrderController {

    @ModelAttribute
    public OrderForm setupForm() {
        return new OrderForm();
    }

    @RequestMapping(value = "order", method = RequestMethod.GET, params =
↪"form")
    public String orderForm() {
        return "order/orderForm";
    }

    @RequestMapping(value = "order", method = RequestMethod.POST, params =
↪"confirm")
    public String orderConfirm(@Validated OrderForm form, BindingResult␣
↪result) {
```

```
        if (result.hasErrors()) {
            return "order/orderForm";
        }
        return "order/orderConfirm";
    }
}
```

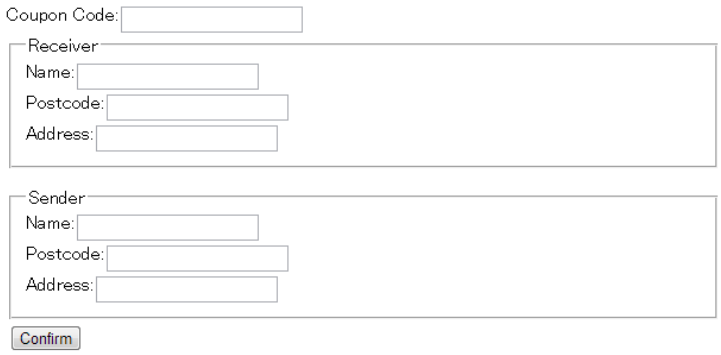
- JSP

```
<!DOCTYPE html>
<html>
<%-- WEB-INF/views/order/orderForm.jsp --%>
<head>
<style type="text/css">
  /* omitted (same as previous sample) */
</style>
</head>
<body>
    <form:form modelAttribute="orderForm" method="post"
        class="form-horizontal"
        action="${pageContext.request.contextPath}/order/order">
        <form:label path="coupon" cssErrorClass="error-label">Coupon Code:</
↪form:label>
        <form:input path="coupon" cssErrorClass="error-input" />
        <form:errors path="coupon" cssClass="error-messages" />
        <br>
    <fieldset>
        <legend>Receiver</legend>
        <%-- (1) --%>
        <form:errors path="receiverAddress"
            cssClass="error-messages" />
        <%-- (2) --%>
        <form:label path="receiverAddress.name"
            cssErrorClass="error-label">Name:</form:label>
        <form:input path="receiverAddress.name"
            cssErrorClass="error-input" />
        <form:errors path="receiverAddress.name"
            cssClass="error-messages" />
        <br>
        <form:label path="receiverAddress.postcode"
            cssErrorClass="error-label">Postcode:</form:label>
        <form:input path="receiverAddress.postcode"
            cssErrorClass="error-input" />
        <form:errors path="receiverAddress.postcode"
            cssClass="error-messages" />
        <br>
        <form:label path="receiverAddress.address"
            cssErrorClass="error-label">Address:</form:label>
        <form:input path="receiverAddress.address"
            cssErrorClass="error-input" />
```

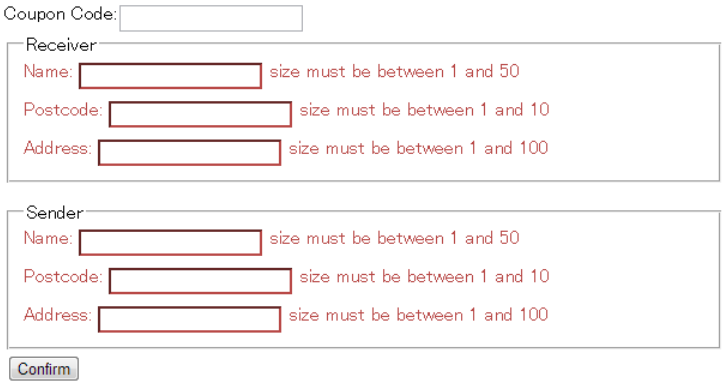
```
        <form:errors path="receiverAddress.address"
            cssClass="error-messages" />
    </fieldset>
    <br>
    <fieldset>
        <legend>Sender</legend>
        <form:errors path="senderAddress"
            cssClass="error-messages" />
        <form:label path="senderAddress.name"
            cssErrorClass="error-label">Name:</form:label>
        <form:input path="senderAddress.name"
            cssErrorClass="error-input" />
        <form:errors path="senderAddress.name"
            cssClass="error-messages" />
        <br>
        <form:label path="senderAddress.postcode"
            cssErrorClass="error-label">Postcode:</form:label>
        <form:input path="senderAddress.postcode"
            cssErrorClass="error-input" />
        <form:errors path="senderAddress.postcode"
            cssClass="error-messages" />
        <br>
        <form:label path="senderAddress.address"
            cssErrorClass="error-label">Address:</form:label>
        <form:input path="senderAddress.address"
            cssErrorClass="error-input" />
        <form:errors path="senderAddress.address"
            cssClass="error-messages" />
    </fieldset>

        <form:button name="confirm">Confirm</form:button>
    </form:form>
</body>
</html>
```

Sr. No.	Description
(1)	When `receiverAddress.name`, `receiverAddress.postcode`, `receiverAddress.address` are not sent as request parameters due to invalid operation, `receiverAddress` is considered as `null` and error message is displayed.
(2)	Fields of nested bean are specified as `[parent field name].[child field name]`.

Form is displayed as follows.



Error message is displayed as follows if this form is sent with all the input fields left blank.



Validation of nested bean is enabled for collections also.

Add a field such that up to 3 addresses can be registered in “user registration” form explained at the beginning.

- Add list of `AddressForm` as a field in the form class.

```
package com.example.sample.app.validation;

import java.io.Serializable;
import java.util.List;

import javax.validation.Valid;
import javax.validation.constraints.Max;
import javax.validation.constraints.Min;
import javax.validation.constraints.NotNull;
import javax.validation.constraints.Size;

import org.hibernate.validator.constraints.Email;

public class UserForm implements Serializable {

    private static final long serialVersionUID = 1L;

    @NotNull
```

```
    @Size(min = 1, max = 20)
    private String name;

    @NotNull
    @Size(min = 1, max = 50)
    @Email
    private String email;

    @NotNull
    @Min(0)
    @Max(200)
    private Integer age;

    @NotNull
    @Size(min = 1, max = 3) // (1)
    @Valid
    private List<AddressForm> addresses;

    // omitted setter/getter
}
```

Sr. No.	Description
(1)	It is possible to use `@Size` annotation for checking size of collection as well.

- JSP

```
<!DOCTYPE html>
<html>
<%-- WEB-INF/views/user/createForm.jsp --%>
<head>
<style type="text/css">
  /* omitted (same as previous sample) */
</style>
</head>
<body>

    <form:form modelAttribute="userForm" method="post"
        class="form-horizontal"
        action="${pageContext.request.contextPath}/user/create">
        <form:label path="name" cssErrorClass="error-label">Name:</form:label>
        <form:input path="name" cssErrorClass="error-input" />
        <form:errors path="name" cssClass="error-messages" />
        <br>
        <form:label path="email" cssErrorClass="error-label">Email:</form:
↪label>
        <form:input path="email" cssErrorClass="error-input" />
        <form:errors path="email" cssClass="error-messages" />
```

```
        <br>
        <form:label path="age" cssErrorClass="error-label">Age:</form:label>
        <form:input path="age" cssErrorClass="error-input" />
        <form:errors path="age" cssClass="error-messages" />
        <br>
        <form:errors path="addresses" cssClass="error-messages" /><%-- (1) --
↪%>
        <c:forEach items="${userForm.addresses}" varStatus="status"><%-- (2) -
↪-%>
            <fieldset class="address">
                <legend>Address${f:h(status.index + 1)}</legend>
                <form:label path="addresses[${status.index}].name"
                    cssErrorClass="error-label">Name:</form:label><%-- (3) --
↪%>
                <form:input path="addresses[${status.index}].name"
                    cssErrorClass="error-input" />
                <form:errors path="addresses[${status.index}].name"
                    cssClass="error-messages" />
                <br>
                <form:label path="addresses[${status.index}].postcode"
                    cssErrorClass="error-label">Postcode:</form:label>
                <form:input path="addresses[${status.index}].postcode"
                    cssErrorClass="error-input" />
                <form:errors path="addresses[${status.index}].postcode"
                    cssClass="error-messages" />
                <br>
                <form:label path="addresses[${status.index}].address"
                    cssErrorClass="error-label">Address:</form:label>
                <form:input path="addresses[${status.index}].address"
                    cssErrorClass="error-input" />
                <form:errors path="addresses[${status.index}].address"
                    cssClass="error-messages" />
                <c:if test="${status.index > 0}">
                    <br>
                    <button class="remove-address-button">Remove</button>
                </c:if>
            </fieldset>
            <br>
        </c:forEach>
        <button id="add-address-button">Add address</button>
        <br>
        <form:button name="confirm">Confirm</form:button>
    </form:form>
    <script type="text/javascript"
        src="${pageContext.request.contextPath}/resources/vendor/js/jquery-1.
↪10.2.min.js"></script>
    <script type="text/javascript"
        src="${pageContext.request.contextPath}/resources/app/js/
↪AddressesView.js"></script>
</body>
```

```
</html>
```

Sr. No.	Description
(1)	Display error message related to `address` field.
(2)	Process the collection of child forms in a loop using `<c:forEach>` tag.
(3)	Inside the loop, Specify the field of child form using `[parent field name][Index].[child field name].`

- Controller class

```
package com.example.sample.app.validation;

import java.util.ArrayList;
import java.util.List;

import org.springframework.stereotype.Controller;
import org.springframework.validation.BindingResult;
import org.springframework.validation.annotation.Validated;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

@Controller
@RequestMapping("user")
public class UserController {

    @ModelAttribute
    public UserForm setupForm() {
        UserForm form = new UserForm();
        List<AddressForm> addresses = new ArrayList<AddressForm>();
        addresses.add(new AddressForm());
        form.setAddresses(addresses); // (1)
        return form;
    }

    @RequestMapping(value = "create", method = RequestMethod.GET, params =
↪"form")
    public String createForm() {
        return "user/createForm";
    }
```

```
    @RequestMapping(value = "create", method = RequestMethod.POST, params =
↪"confirm")
    public String createConfirm(@Validated UserForm form, BindingResult␣
↪result) {
        if (result.hasErrors()) {
            return "user/createForm";
        }
        return "user/createConfirm";
    }
}
```

Sr. No.	Description
(1)	Edit the form object to display a single address form at the time of initial display of "user registration" form.

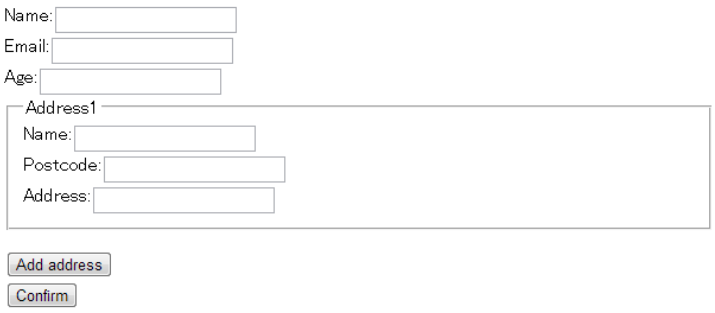
- JavaScript

Below is the JavaScript to dynamically add address input field. However, the explanation of this code is omitted as it is not required.

```
// webapp/resources/app/js/AddressesView.js

function AddressesView() {
  this.addressSize = $('fieldset.address').size();
};

AddressesView.prototype.addAddress = function() {
  var $address = $('fieldset.address');
  var newHtml = addressTemplate(this.addressSize++);
  $address.last().next().after($(newHtml));
};

AddressesView.prototype.removeAddress = function($fieldset) {
  $fieldset.next().remove(); // remove <br>
  $fieldset.remove(); // remove <fieldset>
};

function addressTemplate(number) {
  return '\
<fieldset class="address">\
    <legend>Address' + (number + 1) + '</legend>\
    <label for="addresses' + number + '.name">Name:</label>\
    <input id="addresses' + number + '.name" name="addresses[' + number + '].
↪name" type="text" value=""/><br>\
    <label for="addresses' + number + '.postcode">Postcode:</label>\
    <input id="addresses' + number + '.postcode" name="addresses[' + number +
↪'].postcode" type="text" value=""/><br>\
```

```
    <label for="addresses' + number + '.address">Address:</label>\
    <input id="addresses' + number + '.address" name="addresses[' + number +
↪'].address" type="text" value=""/><br>\
    <button class="remove-address-button">Remove</button>\
</fieldset>\
<br>\
';
}

$(function() {
  var addressesView = new AddressesView();

  $('#add-address-button').on('click', function(e) {
    e.preventDefault();
    addressesView.addAddress();
  });

  $(document).on('click', '.remove-address-button', function(e) {
    if (this === e.target) {
      e.preventDefault();
      var $this = $(this); // this button
      var $fieldset = $this.parent(); // fieldset
      addressesView.removeAddress($fieldset);
    }
  });

});
```

Form is displayed as follows.



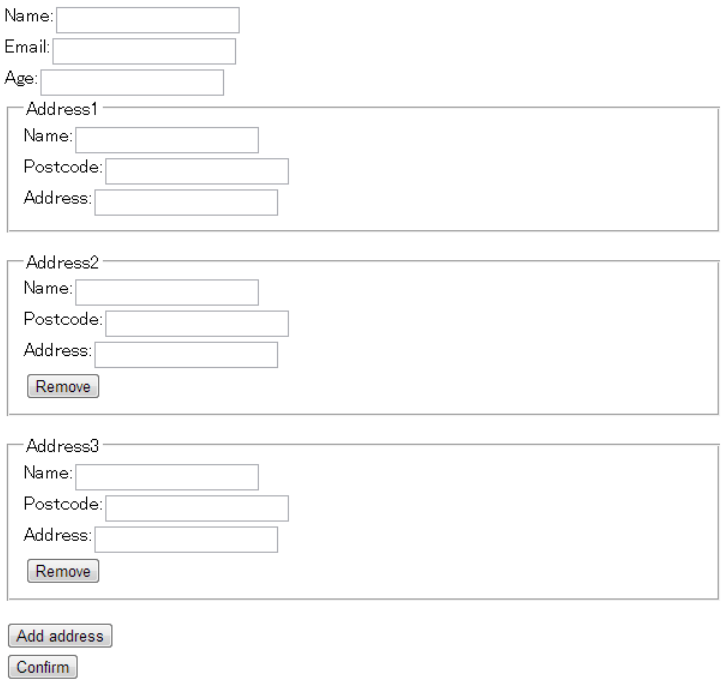
Add 2 address forms by clicking “Add address” button twice.

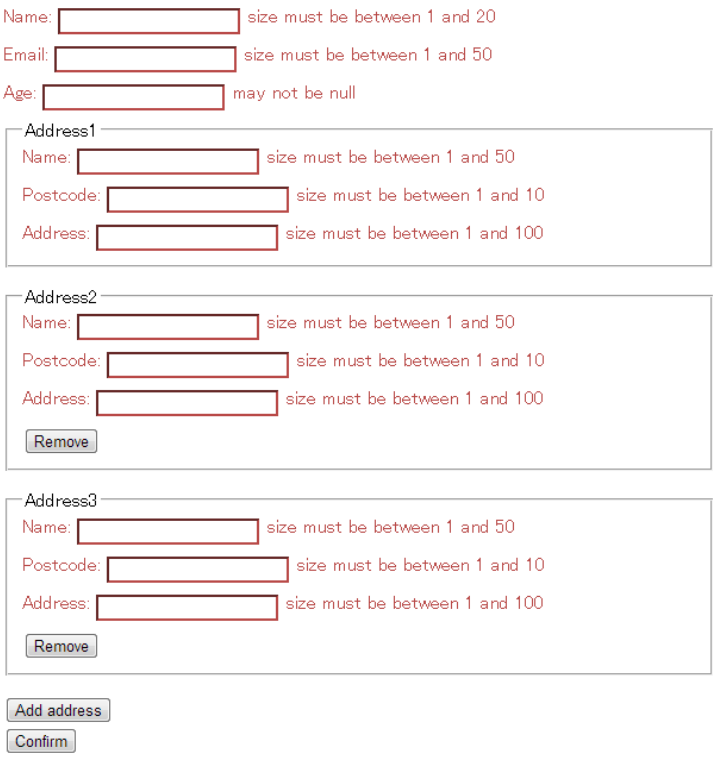
Error message is displayed as follows if this form is sent with all the input fields left blank.

Grouped validation

By creating validation group, input validation rules for a field can be specified for each group.

In the “new user registration” example, add “Must be an adult” rule for the `age` field. Add `country` field also


Name:
Email:
Age:
Address1
Name:
Postcode:
Address:
Address2
Name:
Postcode:
Address:
Remove
Address3
Name:
Postcode:
Address:
Remove
Add address
Confirm


Name: size must be between 1 and 20
Email: size must be between 1 and 50
Age: may not be null
Address1
Name: size must be between 1 and 50
Postcode: size must be between 1 and 10
Address: size must be between 1 and 100
Address2
Name: size must be between 1 and 50
Postcode: size must be between 1 and 10
Address: size must be between 1 and 100
Remove
Address3
Name: size must be between 1 and 50
Postcode: size must be between 1 and 10
Address: size must be between 1 and 100
Remove
Add address
Confirm

as “Adult” rules differ with country.

To specify group in Bean Validation, set any `java.lang.Class` object representing a group in `group` attribute of the annotation.

Create the following 3 groups (interface) here.

Group	Adult condition
`Chinese`	18 years or more
`Japanese`	20 years or more
`Singaporean`	21 years or more

An example of executing validation using these groups is shown here.

- Form class

```
package com.example.sample.app.validation;

import java.io.Serializable;
import java.util.List;

import javax.validation.Valid;
import javax.validation.constraints.Max;
import javax.validation.constraints.Min;
import javax.validation.constraints.NotNull;
import javax.validation.constraints.Size;

import org.hibernate.validator.constraints.Email;

public class UserForm implements Serializable {

    private static final long serialVersionUID = 1L;

    // (1)
    public static interface Chinese {
    };

    public static interface Japanese {
    };

    public static interface Singaporean {
    };

    @NotNull
    @Size(min = 1, max = 20)
    private String name;

    @NotNull
    @Size(min = 1, max = 50)
    @Email
    private String email;
```

```
    @NotNull
    @Min.List({ // (2)
            @Min(value = 18, groups = Chinese.class), // (3)
            @Min(value = 20, groups = Japanese.class),
            @Min(value = 21, groups = Singaporean.class)
            })
    @Max(200)
    private Integer age;

    @NotNull
    @Size(min = 2, max = 2)
    private String country; // (4)

    // omitted setter/getter
}
```

Sr. No.	Description
(1)	Define each group as an interface.
(2)	`@Min.List` annotation is used to specify multiple `@Min` rules on a single field. It is same even while using other annotations.
(3)	Specify corresponding group class in the `group` attribute, in order to define rules for each group. When `group` attribute is not specified, `javax.validation.groups.Default` group is used.
(4)	Add a field which will be used to determine which group is to be applied.

- JSP

  There are no major changes in JSP.

```
<form:form modelAttribute="userForm" method="post"
    class="form-horizontal"
    action="${pageContext.request.contextPath}/user/create">
    <form:label path="name" cssErrorClass="error-label">Name:</form:label>
    <form:input path="name" cssErrorClass="error-input" />
```

```
    <form:errors path="name" cssClass="error-messages" />
    <br>
    <form:label path="email" cssErrorClass="error-label">Email:</form:label>
    <form:input path="email" cssErrorClass="error-input" />
    <form:errors path="email" cssClass="error-messages" />
    <br>
    <form:label path="age" cssErrorClass="error-label">Age:</form:label>
    <form:input path="age" cssErrorClass="error-input" />
    <form:errors path="age" cssClass="error-messages" />
    <br>
    <form:label path="country" cssErrorClass="error-label">Country:</form:
↪label>
    <form:select path="country" cssErrorClass="error-input">
        <form:option value="cn">China</form:option>
        <form:option value="jp">Japan</form:option>
        <form:option value="sg">Singapore</form:option>
    </form:select>
    <form:errors path="country" cssClass="error-messages" />
    <br>
    <form:button name="confirm">Confirm</form:button>
</form:form>
```

- Controller class

  By giving a group name to `@Validated` annotation, the rules defined for that group will be applied.

```
package com.example.sample.app.validation;


import javax.validation.groups.Default;

import org.springframework.stereotype.Controller;
import org.springframework.validation.BindingResult;
import org.springframework.validation.annotation.Validated;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

import com.example.sample.app.validation.UserForm.Chinese;
import com.example.sample.app.validation.UserForm.Japanese;
import com.example.sample.app.validation.UserForm.Singaporean;

@Controller
@RequestMapping("user")
public class UserController {

    @ModelAttribute
    public UserForm setupForm() {
        UserForm form = new UserForm();
        return form;
```

```
    }

    @RequestMapping(value = "create", method = RequestMethod.GET, params =
↪"form")
    public String createForm() {
        return "user/createForm";
    }

    String createConfirm(UserForm form, BindingResult result) {
        if (result.hasErrors()) {
            return "user/createForm";
        }
        return "user/createConfirm";
    }

    @RequestMapping(value = "create", method = RequestMethod.POST, params␣
↪= {
            "confirm",  /* (1) */ "country=cn" })
    public String createConfirmForChinese(@Validated({ /* (2) */ Chinese.
↪class,
            Default.class }) UserForm form, BindingResult result) {
        return createConfirm(form, result);
    }

    @RequestMapping(value = "create", method = RequestMethod.POST, params␣
↪= {
            "confirm", "country=jp" })
    public String createConfirmForJapanese(@Validated({ Japanese.class,
            Default.class }) UserForm form, BindingResult result) {
        return createConfirm(form, result);
    }

    @RequestMapping(value = "create", method = RequestMethod.POST, params␣
↪= {
            "confirm", "country=sg" })
    public String createConfirmForSingaporean(@Validated({ Singaporean.
↪class,
            Default.class }) UserForm form, BindingResult result) {
        return createConfirm(form, result);
    }
}
```

Sr. No.	Description
(1)	The parameter which is used as condition to dividing between the groups must be set to `param` attribute.
(2)	All the annotations, except `@Min` of `age` field, belongs to `Default` group; hence, specifying `Default` is mandatory.

In this example, the check result of the combination of each input value is as follows.

`age` value	`country` value	Input validation result	Error message
17	cn	NG	must be greater than or equal to 18
	jp	NG	must be greater than or equal to 20
	sg	NG	must be greater than or equal to 21
18	cn	OK	
	jp	NG	must be greater than or equal to 20
	sg	NG	must be greater than or equal to 21
20	cn	OK	
	jp	OK	
	sg	NG	must be greater than or equal to 21
21	cn	OK	
	jp	OK	
	sg	OK	

Warning: Implementation of this Controller is inadequate; there is no handling when `country` value is neither “cn”, “jp” nor “sg”. 400 error is returned when unexpected `country` value is encountered.

Next, we can think of a condition where the number of countries increase and adult condition of 18 years or more is be set as a default rule.

Rules are as follows.

Group	Adult condition
`Japanese`	20 years or more
`Singaporean`	21 years or more
Country other than the above-mentioned(`Default`)	18 years or more

- Form class

  In order to specify a value to `Default` group (18 years or more), all groups should be specified explicitly in other annotations as well.

```
package com.example.sample.app.validation;

import java.io.Serializable;
import java.util.List;

import javax.validation.Valid;
import javax.validation.constraints.Max;
import javax.validation.constraints.Min;
import javax.validation.constraints.NotNull;
import javax.validation.constraints.Size;
import javax.validation.groups.Default;

import org.hibernate.validator.constraints.Email;

public class UserForm implements Serializable {

    private static final long serialVersionUID = 1L;

    public static interface Japanese {
    };

    public static interface Singaporean {
    };

    @NotNull(groups = { Default.class, Japanese.class, Singaporean.class }) //
 (1)
    @Size(min = 1, max = 20, groups = { Default.class, Japanese.class,
            Singaporean.class })
    private String name;
```

```
    @NotNull(groups = { Default.class, Japanese.class, Singaporean.class })
    @Size(min = 1, max = 50, groups = { Default.class, Japanese.class,
            Singaporean.class })
    @Email(groups = { Default.class, Japanese.class, Singaporean.class })
    private String email;

    @NotNull(groups = { Default.class, Japanese.class, Singaporean.class })
    @Min.List({
            @Min(value = 18, groups = Default.class), // (2)
            @Min(value = 20, groups = Japanese.class),
            @Min(value = 21, groups = Singaporean.class) })
    @Max(200)
    private Integer age;

    @NotNull(groups = { Default.class, Japanese.class, Singaporean.class })
    @Size(min = 2, max = 2, groups = { Default.class, Japanese.class,
            Singaporean.class })
    private String country;

    // omitted setter/getter
}
```

Sr. No.	Description
(1)	Set all groups to annotations other than `@Min` of `age` field as well.
(2)	Set the rule for `Default` group.

- JSP

  No change in JSP

- Controller class

```
package com.example.sample.app.validation;

import org.springframework.stereotype.Controller;
import org.springframework.validation.BindingResult;
import org.springframework.validation.annotation.Validated;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

import com.example.sample.app.validation.UserForm.Japanese;
import com.example.sample.app.validation.UserForm.Singaporean;
```

```
@Controller
@RequestMapping("user")
public class UserController {

    @ModelAttribute
    public UserForm setupForm() {
        UserForm form = new UserForm();
        return form;
    }

    @RequestMapping(value = "create", method = RequestMethod.GET, params =
↪"form")
    public String createForm() {
        return "user/createForm";
    }

    String createConfirm(UserForm form, BindingResult result) {
        if (result.hasErrors()) {
            return "user/createForm";
        }
        return "user/createConfirm";
    }

    @RequestMapping(value = "create", method = RequestMethod.POST, params = {
↪"confirm" })
    public String createConfirmForDefault(@Validated /* (1) */ UserForm form,
            BindingResult result) {
        return createConfirm(form, result);
    }

    @RequestMapping(value = "create", method = RequestMethod.POST, params = {
            "confirm", "country=jp" })
    public String createConfirmForJapanese(
            @Validated(Japanese.class)  /* (2) */ UserForm form,␣
↪BindingResult result) {
        return createConfirm(form, result);
    }

    @RequestMapping(value = "create", method = RequestMethod.POST, params = {
            "confirm", "country=sg" })
    public String createConfirmForSingaporean(
            @Validated(Singaporean.class) UserForm form, BindingResult␣
↪result) {
        return createConfirm(form, result);
    }
}
```

Sr. No.	Description
(1)	When the field `country` does not have a value, the request is mapped to a method in which `Default` group is specified in `@Validated` annotation.
(2)	When the field `country` has a value, the request is mapped to a method in which `Default` group is not included in `@Validated` annotation.

Till now, 2 patterns of using grouped validation have been explained.

In the previous pattern, `Default` group is used in Controller class and in the later one, `Default` group is used in form class.

Pattern	Advantages	Disadvantages	Decision points
Using `Default` group in Controller class	`group` attribute need not be set for the rules that need not be grouped.	Since all patterns of group should be defined, it is difficult to define when there are many group patterns.	Should be used when there are only a limited number of group patterns (New create group, Update group and Delete group)
Using `Default` group in form class	Since only the groups that do not belong to the default group need to be defined, it can be handled even if there are many patterns.	`group` attribute should be set for the rules that need not be grouped making the process complicated.	Should be used when there are many group patterns and majority of patterns have a common value.

If none of the above decision points are applicable, then using Bean Validation itself might not be a good idea. After reviewing the design, usage of Spring Validator or implementation of validation in business logic should be considered.

Note: In the examples explained so far, the switching of group validation is carried out using request parameter and parameter that can be specified in `@RequestMapping` annotation. It is not possible to switch between groups, if switching is to be performed based on permissions in authentication object or any information which cannot be handled by `@RequestMapping` annotation.

In such a case, `@Validated` annotation must not be used but `org.springframework.validation.SmartValidator` must be used. Group validation can be performed inside the handler method of controller.

```
@Controller
@RequestMapping("user")
public class UserController {

    @Inject
```

```
    SmartValidator smartValidator; // (1)

    // omitted

    @RequestMapping(value = "create", method = RequestMethod.POST, params =
↪"confirm")
    public String createConfirm(/* (2) */ UserForm form, BindingResult␣
↪result) {
        // (3)
        Class<?> validationGroup = Default.class;
        // logic to determine validation group
        // if (xxx) {
        //     validationGroup = Xxx.class;
        // }
        smartValidator.validate(form, result, validationGroup); // (4)
        if (result.hasErrors()) {
            return "user/createForm";
        }
        return "user/createConfirm";
    }

}
```

Sr. No.	Description
(1)	Inject `SmartValidator`. Since `SmartValidator` can be used if `<mvc:annotation-driven>` setting is carried out so there is no need to define separately.
(2)	Do not use `@Validated` annotation.
(3)	Determine a validation group. Logic to determine a validation group recommends delegating to Helper class and keeping logic in Controller in simple state.
(4)	Execute grouped validation using `validate` method of `SmartValidator`. Multiple groups can be specified in `validate` method.

Since logic should not be written in Controller, if switching is possible using request parameters in `@RequestMapping`annotation, `SmartValidator` must not be used.

Correlation item check

For the validation of correlated items, Spring Validator(`Validator` implementing `org.springframework.validation.Validator` interface) or Bean Validation must be used.

Each one of above has been explained below. However, before that, their features and usage have been explained.

Format	Features	Usage
Spring Validator	It is easy to create input validation for a particular class. It is inconvenient to use in Controller.	Input validation implementation of unique business requirements depending on specific form
Bean Validation	Creation of input validation is not as easy as Spring Validator. It is easy to use in Controller.	Common input validation implementation of development project not depending on specific form

Correlation item check implementation using Spring Validator

Implementation method is explained with the help of “reset password” process as an example.

Implement the following rules. Following rules are provided in the “reset password” form.

Field name	Type	Rules	Description
password	`java.lang.String`	Mandatory input 8 or more characters **Must be same as confirmPassword**	Password
confirmPassword	`java.lang.String`	Nothing in particular	Confirm password

Check rule “Must be same as confirmPassword” is validation of correlated items as `password` field and `confirmPassword` field should have the same value.

- Form class

  other than validation of correlated items, implement using Bean Validation annotation.

```
package com.example.sample.app.validation;

import java.io.Serializable;

import javax.validation.constraints.NotNull;
import javax.validation.constraints.Size;

public class PasswordResetForm implements Serializable {
    private static final long serialVersionUID = 1L;

    @NotNull
    @Size(min = 8)
    private String password;

    private String confirmPassword;

    // omitted setter/getter
}
```

Note: Password is normally saved in database after hashing it, hence there is no need to check the maximum number of characters.

- Validator class

Implement validation of correlated items using `org.springframework.validation.Validator` interface.

```
package com.example.sample.app.validation;

import org.springframework.stereotype.Component;
import org.springframework.validation.Errors;
import org.springframework.validation.Validator;

@Component // (1)
public class PasswordEqualsValidator implements Validator {

    @Override
    public boolean supports(Class<?> clazz) {
        return PasswordResetForm.class.isAssignableFrom(clazz); // (2)
    }

    @Override
    public void validate(Object target, Errors errors) {

        if (errors.hasFieldErrors("password")) { // (3)
            return;
        }
```

```
        PasswordResetForm form = (PasswordResetForm) target;
        String password = form.getPassword();
        String confirmPassword = form.getConfirmPassword();

        if (!password.equals(confirmPassword)) { // (4)
            errors.rejectValue(/* (5) */ "password",
            /* (6) */ "PasswordEqualsValidator.passwordResetForm.password",
            /* (7) */ "password and confirm password must be same.");
        }
    }
}
```

Sr. No.	Description
(1)	Assign `@Component` to make Validator the target of component scan.
(2)	Decide the argument is check target of this validator or not. Here `PasswordResetForm` class is the target to be checked.
(3)	If an error occurs at the target fields during a single item check, do not perform correlation check in this Validator. If it is necessary to perform the correlation check, this determination logic is not required.
(4)	Implement check logic.
(5)	Specify field name where there is error.
(6)	Specify code name of error message. Here, code is "[validator name].[form attribute name].[property name]" Refer to *Messages to be defined in application-messages.properties* for mesaage definition.
(7)	Set default message to be used when error message does not get resolved using code.

Note: Spring Validator implementation class should be placed in the same package as the Controller.

- Controller class

```
package com.example.sample.app.validation;

import javax.inject.Inject;

import org.springframework.stereotype.Controller;
import org.springframework.validation.BindingResult;
import org.springframework.validation.annotation.Validated;
import org.springframework.web.bind.WebDataBinder;
import org.springframework.web.bind.annotation.InitBinder;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

@Controller
@RequestMapping("password")
public class PasswordResetController {
    @Inject
    PasswordEqualsValidator passwordEqualsValidator; // (1)

    @ModelAttribute
    public PasswordResetForm setupForm() {
        return new PasswordResetForm();
    }

    @InitBinder
    public void initBinder(WebDataBinder binder) {
        binder.addValidators(passwordEqualsValidator); // (2)
    }

    @RequestMapping(value = "reset", method = RequestMethod.GET, params =
↪"form")
    public String resetForm() {
        return "password/resetForm";
    }

    @RequestMapping(value = "reset", method = RequestMethod.POST)
    public String reset(@Validated PasswordResetForm form, BindingResult␣
↪result) { // (3)
        if (result.hasErrors()) {
            return "password/resetForm";
        }
        return "redirect:/password/reset?complete";
    }
```

```
    @RequestMapping(value = "reset", method = RequestMethod.GET, params =
↪"complete")
    public String resetComplete() {
        return "password/resetComplete";
    }
}
```

Sr. No.	Description
(1)	Inject Spring Validator to be used.
(2)	In the method having `@InitBinder` annotation, add Validators using `WebDataBinder.addValidators` method. By this, the added Validator is called when validation is executed with the use of `@Validated` annotation.
(3)	Implement input validation as per the process performed so far.

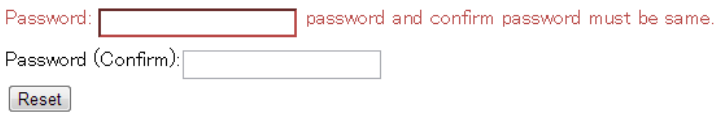
- JSP

  There are no points to mention for JSP.

```
<!DOCTYPE html>
<html>
<%-- WEB-INF/views/password/resetForm.jsp --%>
<head>
<style type="text/css">
/* omitted */
</style>
</head>
<body>
    <form:form modelAttribute="passwordResetForm" method="post"
        class="form-horizontal"
        action="${pageContext.request.contextPath}/password/reset">
        <form:label path="password" cssErrorClass="error-label">Password:</
↪form:label>
        <form:password path="password" cssErrorClass="error-input" />
        <form:errors path="password" cssClass="error-messages" />
        <br>
        <form:label path="confirmPassword" cssErrorClass="error-label">
↪Password (Confirm):</form:label>
        <form:password path="confirmPassword"
            cssErrorClass="error-input" />
```

```
        <form:errors path="confirmPassword" cssClass="error-messages" />
        <br>
        <form:button>Reset</form:button>
    </form:form>
</body>
</html>
```

Error message as shown below is displayed when form is sent by entering different values in `password` field and `confirmPassword` fields.



Note: When `<form:password>` tag is used, data gets cleared at the time of redisplay.

Note: Error information can be set for multiple fields for correlation check. However, displaying error messages and applying style must always be performed in a set and only one part of the tasks cannot not be performed.

When you want to apply style to both the fields wherein correlation check error has occurred however you want to display only one error message, it can be done by setting a null string in the error message. An example is given below wherein style is applied to `password` field and `confirmPassword` field and error message is displayed only in `password` field.

```
package com.example.sample.app.validation;

import org.springframework.stereotype.Component;
import org.springframework.validation.Errors;
import org.springframework.validation.Validator;

@Component
public class PasswordEqualsValidator implements Validator {

    @Override
    public boolean supports(Class<?> clazz) {
        return PasswordResetForm.class.isAssignableFrom(clazz);
    }

    @Override
    public void validate(Object target, Errors errors) {

        // omitted
        if (!password.equals(confirmPassword)) {
            // register a field error for password
```

```
            errors.rejectValue("password",
                    "PasswordEqualsValidator.passwordResetForm.password",
                    "password and confirm password must be same.");

            // register a field error for confirmPassword
            errors.rejectValue("confirmPassword", // (1)
                    "PasswordEqualsValidator.passwordResetForm.
↪confirmPassword", // (2)
                    ""); // (3)
        }
    }
}
```

Sr. No.	Description
(1)	Register error in `confirmPassword`field.
(2)	Specify code name of error message. Specify a null string in the corresponding error message at that time. For message definition, refer *Messages to be defined in application-messages.properties*.
(3)	Set a default message to be used when error message could not be resolved in the code. A null string is set in the example above.

Note: When multiple forms are used in a single controller, model name should be specified in `@InitBinder("xxx")` in order to limit the target of Validator.

```
@Controller
@RequestMapping("xxx")
public class XxxController {
    // omitted
    @ModelAttribute("aaa")
    public AaaForm() {
        return new AaaForm();
    }

    @ModelAttribute("bbb")
    public BbbForm() {
        return new BbbForm();
    }
```

```
    @InitBinder("aaa")
    public void initBinderForAaa(WebDataBinder binder) {
        // add validators for AaaForm
        binder.addValidators(aaaValidator);
    }

    @InitBinder("bbb")
    public void initBinderForBbb(WebDataBinder binder) {
        // add validators for BbbForm
        binder.addValidators(bbbValidator);
    }
    // omitted
}
```

Note: To change the check contents of correlated items check rules in accordance with a validation group (for example: To implement correlated items check only when specific validation group is specified, etc.), it is better to switch the process within validate method by implementing `org.springframework.validation.SmartValidator` interface instead of implementing `org.springframework.validation.Validator` interface.

```
package com.example.sample.app.validation;

import org.apache.commons.lang3.ArrayUtils;
import org.springframework.stereotype.Component;
import org.springframework.validation.Errors;
import org.springframework.validation.SmartValidator;

@Component
public class PasswordEqualsValidator implements SmartValidator { //␣
↪Implements SmartValidator instead of Validator interface

    @Override
    public boolean supports(Class<?> clazz) {
        return PasswordResetForm.class.isAssignableFrom(clazz);
    }

    @Override
    public void validate(Object target, Errors errors) {
        validate(target, errors, new Object[] {});
    }

    @Override
    public void validate(Object target, Errors errors, Object...␣
↪validationHints) {
        // Check validationHints(groups) and apply validation logic only when
↪'Update.class' is specified
```

```
        if (ArrayUtils.contains(validationHints, Update.class)) {
            PasswordResetForm form = (PasswordResetForm) target;
            String password = form.getPassword();
            String confirmPassword = form.getConfirmPassword();

            // omitted...
        }
    }
}
```

Implementation of input check of correlated items using Bean Validation

Independent validation rules should be added to implement validation of correlated items using Bean Validation.

It is explained in *How to extend*.

Definition of error messages

Method to change error messages of input validation is explained.

Error messages of Bean Validation in Spring MVC are resolved in the following order.

1. If there is any message which matches with the rule, among the messages defined in `org.springframework.context.MessageSource`, then it is to be used as error message (Spring rule).

   For default rules of Spring, refer to "JavaDoc of DefaultMessageCodesResolver of DefaultMessageCodesResolver".

2. If message cannot be found as mentioned in step 1, then error message is acquired from the `message` attribute of the annotation. (Bean Validation rule)

   1. When the value of `message` attribute is not in "{message key}" format, use that text as error message.
   2. When the value of `message` attribute is in "{message key}" format, search messages corresponding to message key from ValidationMessages.properties under classpath.

      1. When message corresponding to message key is defined, use that message
      2. When message corresponding to message key is not defined, use "{message key}" as error message

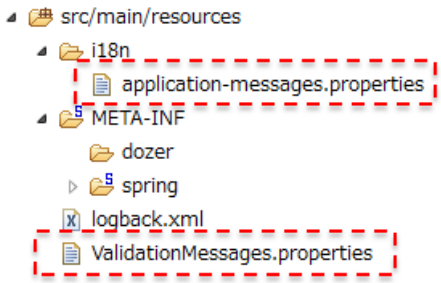
Basically, it is recommended to define error messages in properties file.

Messages should be defined at following places.

- properties file read by `org.springframework.context.MessageSource`
- ValidationMessages.properties under classpath

Considering that the following settings are done in applicationContext.xml, former is called as "application-messages.properties" and latter is called "ValidationMessages.properties".

```
<bean id="messageSource"
    class="org.springframework.context.support.ResourceBundleMessageSource">
    <property name="basenames">
        <list>
            <value>i18n/application-messages</value>
        </list>
    </property>
</bean>
```



Warning: Multiple `ValidationMessages.properties` files should not exist directly under class path.

If multiple `ValidationMessages.properties` files exist directly under class path, an appropriate message may not be displayed, as either one file of them is read leaving rest of the files unread.

- When adopting multi project structure, please take care so that `ValidationMessages.properties` file is not placed in multiple projects.
- When distributing common parts for Bean Validation as jar file, please take care so that `ValidationMessages.properties` file is not included in jar file.

Further, when a project is created from Blank project of version 1.0.2.RELEASE or higher, `ValidationMessages.properties` is stored directly under `xxx-web/src/main/resources`.

This guideline classifies the definition as follows.

Properties file name	Contents to be defined
ValidationMessages.properties	Default error messages of Bean Validation specified by the system
application-messages.properties	Error message of Bean Validation to be overwritten separately Error message of input validation implemented in Spring Validator

When ValidationMessages.properties is not provided, *Default messages provided by Hibernate Validator* is used.

By working with `MessageSource`, you can handle Japanese messages directly without Native to Ascii. For details,please refer *Reading messages which are not converted from Native to Ascii*

Messages to be defined in ValidationMessages.properties

Define messages for message key specified in `message` attribute of Bean Validation annotation of ValidationMessages.properties under class path (normal src/main/resources).

It is explained below using the following form used at the beginning of *Basic single item check*.

- Form class (re-displayed)

```
public class UserForm implements Serializable {

    @NotNull
    @Size(min = 1, max = 20)
    private String name;

    @NotNull
    @Size(min = 1, max = 50)
    @Email
    private String email;

    @NotNull
    @Min(0)
    @Max(200)
    private Integer age;

    // omitted getter/setter
}
```

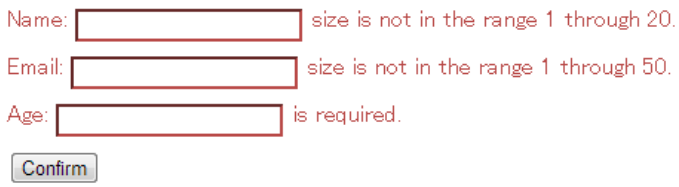
- ValidationMessages.properties

Change error messages of `@NotNull`, `@Size`, `@Min`, `@Max`, `@Email`.

```
javax.validation.constraints.NotNull.message=is required.
# (1)
javax.validation.constraints.Size.message=size is not in the range {min}␣
↪through {max}.
javax.validation.constraints.Min.message=cannot be less than {value}.
javax.validation.constraints.Max.message=cannot be greater than {value}.
org.hibernate.validator.constraints.Email.message=is an invalid e-mail␣
↪address.
```

Sr. No.	Description
(1)	It is possible to embed the value of attributes specified in the annotation using `{Attribute name}`.

When the form is sent with input fields left blank after adding the above settings, changed error messages are displayed as shown below.



Warning: Since `{FQCN of annotation.message}` is set in `message` attribute in Bean Validation standard annotations and independent Hibernate Validator annotations, messages can be defined in properties file in the above format. However, since all annotations may not be in this format, Javadoc or source code of the target annotation should be checked.

```
FQCN of annotation.message = Message
```

Add `{0}` to message as shown below when field name is to be included in error message.

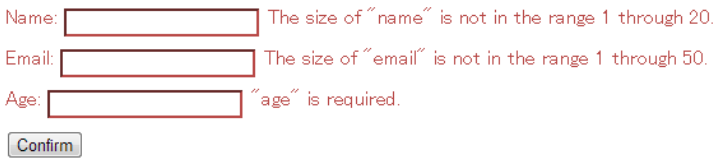
- ValidationMessages.properties

  Change error message of `@NotNull`, `@Size`, `@Min`, `@Max` and `@Email`.

  ```
  javax.validation.constraints.NotNull.message="{0}" is required.
  javax.validation.constraints.Size.message=The size of "{0}" is not in the␣
  ↪range {min} through {max}.
  javax.validation.constraints.Min.message="{0}" cannot be less than {value}.
  javax.validation.constraints.Max.message="{0}" cannot be greater than
  ↪{value}.
  ```

```
org.hibernate.validator.constraints.Email.message="{0}" is an invalid e-
↪mail address.
```

Error message is changed as follows.



In this way, property name of form class gets displayed on the screen and so it is not user friendly. To display an appropriate field name, it should be defined in **application-messages.properties** in the following format.

```
form property name=field name to be displayed
```

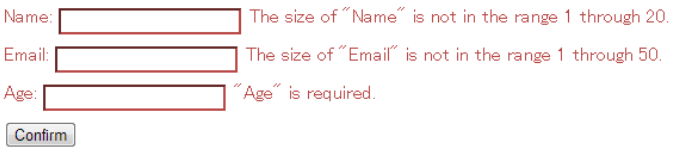
It is good to define field names in the form.

Adding the same to our example.

- application-messages.properties

```
name=Name
email=Email
age=Age
```

Error messages are changed as follows.



Note: Inserting field name in place of `{0}` is the functionality of Spring and not of Bean Validation. Therefore, the settings for changing field name should be defined in application-messages.properties(`ResourceBundleMessageSource`) which is directly under Spring management.

Tip: In Bean Validation 1.1, it is possible to use Expression Language (hereafter referred to as "EL expression") in a message specified in `ValidationMessages.properties`. Hibernate Validator 5.x supports Expression Language 2.2 or higher version.

Executable EL expression version differs depending on the version of application server. Therefore when EL

expression is to be used, **it should be used after confirming the version of EL expression supported by application server.**

Following is an example of using EL expression in a message which is defined in `ValidationMessages.properties` provided by Hibernate Validator by default.

```
# ...
# (1)
javax.validation.constraints.DecimalMax.message  = must be less than $
↪{inclusive == true ? 'or equal to ' : ''}{value}
# ...
```

Sr. No.	Description
(1)	An EL expression is a part of "`${inclusive == true ? 'or equal to ' : ''}`" in a message. From the above mentioned definition of message, 2 patterns of messages are created as given below. • must be less than or equal to {value} • must be less than {value} (A value specified in `value` attribute of `@DecimalMax` annotation is embedded in `{value}` part) Former is created when `true` is specified (or when not specified) in `inclusive` attribute of `@DecimalMax` annotation, Latter is created when `false` is specified in `inclusive` attribute of `@DecimalMax` annotation. For handling of EL expressions in Bean Validation refer to: Hibernate Validator Reference Guide(Interpolation with message expressions).

Further, the value to be checked can be included in the error message by using `${validatedValue}` in the message which is specified in `ValidationMessages.properties`.

How to use `${validatedValue}` is shown below.

```
# ...
# (1)
javax.validation.constraints.Pattern.message = The value entered "$
↪{validatedValue}" is invalid.
# ...
```

Sr. No.	Description
(1)	Messages that are actually generated from the message definition above are embedded with the values input in the form, in `${validatedValue}` part. If confidential information is included in the input value, care must be taken not to use `${validatedValue}` so that confidential information is not displayed. For details, refer Hibernate Validator Reference Guide(Interpolation with message expressions).

Messages to be defined in application-messages.properties

Default messages to be used in system are defined in ValidationMessages.properties however, depending on the screen, they may have to be changed from the default value.

In this case, define messages in the following format in application-messages.properties.

```
[annotation name].[form attribute name].[property name] = [target message]
```

Apply "*Messages to be defined in ValidationMessages.properties*" and override the message for `email` and `age` field using the below settings.

- application-messages.properties

```
# override messages
# for email field
Size.userForm.email=The size of "{0}" must be between {2} and {1}.
# for age field
NotNull.userForm.age="{0}" is compulsory.
Min.userForm.age="{0}" must be greater than or equal to {1}.
Max.userForm.age="{0}" must be less than or equal to {1}.

# filed names
name=Name
email=Email
age=Age
```

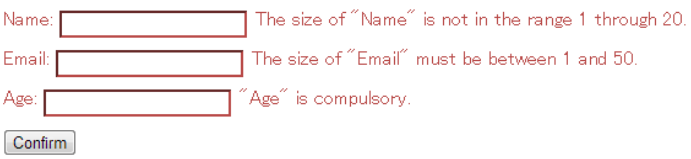
Value of attributes of the annotation gets inserted after `{1}` onwards. Incidentally, index position of attribute values are alphabetical ordering(ascending order) of attribute names.

For example, index positions of `@Size` are as follow:

- `{0}` : property name (physical name or logical name)
- `{1}` : value of `max` attribute
- `{2}` : value of `min` attribute

For specification details, refer to JavaDoc of SpringValidatorAdapter.

Error messages are changed as follows.



Note: There are other formats as well for the message key format of application-messages.properties; however, if

it is used with the purpose of overwriting some default messages, it should be in `[annotation name].[form attribute name].[property name]` format.

4.1.3 How to extend

Other than standard check rules, bean validation has a mechanism to develop annotations for independent rules .

The method of creating independent rules can be widely classified into the following two broader criteria.

- Combination of existing rules
- Creation of new rules

Basically, the below template can be used to create annotation for each rule.

```
package com.example.common.validation;

import java.lang.annotation.Documented;
import java.lang.annotation.Retention;
import java.lang.annotation.Target;
import javax.validation.Constraint;
import javax.validation.Payload;
import static java.lang.annotation.ElementType.ANNOTATION_TYPE;
import static java.lang.annotation.ElementType.CONSTRUCTOR;
import static java.lang.annotation.ElementType.FIELD;
import static java.lang.annotation.ElementType.METHOD;
import static java.lang.annotation.ElementType.PARAMETER;
import static java.lang.annotation.RetentionPolicy.RUNTIME;

@Documented
@Constraint(validatedBy = {})
@Target({ METHOD, FIELD, ANNOTATION_TYPE, CONSTRUCTOR, PARAMETER })
@Retention(RUNTIME)
public @interface Xxx {
    String message() default "{com.example.common.validation.Xxx.message}";

    Class<?>[] groups() default {};

    Class<? extends Payload>[] payload() default {};

    @Target({ METHOD, FIELD, ANNOTATION_TYPE, CONSTRUCTOR, PARAMETER })
    @Retention(RUNTIME)
    @Documented
    public @interface List {
```

```
        Xxx[] value();
    }
}
```

Creation of Bean Validation annotation by combining existing rules

Consider the following restrictions at the system level and domain level respectively.

At the system level,

- String must be single byte alphanumeric characters
- Numbers must be positive

Or at the domain level,

- "User ID" must be between 4 and 20 single byte characters
- "Age" must be between 1 year and 150 years

These can be implemented by combining `@Pattern`, `@Size`, `@Min`, `@Max` of the existing rules.
However, if the same rules are to be used at multiple places, settings get distributed and maintainability worsens.

One rule can be created by combining multiple rules. There is an advantage to be able to have not only common regular expression pattern and maximum/minimum values but also error message when an independent annotation is created. By this, reusability and maintainability increases. Even if multiple rules are not combined, it also proves beneficial if used only to give specific value to an attribute.

Implementation example is shown below.

- Implementation example of `@Alphanumeric` annotation which is restricted to single byte alphanumeric characters

```
package com.example.common.validation;

import java.lang.annotation.Documented;
import java.lang.annotation.Retention;
import java.lang.annotation.Target;
import javax.validation.Constraint;
```

```
import javax.validation.Payload;
import javax.validation.ReportAsSingleViolation;
import javax.validation.constraints.Pattern;

import static java.lang.annotation.ElementType.ANNOTATION_TYPE;
import static java.lang.annotation.ElementType.CONSTRUCTOR;
import static java.lang.annotation.ElementType.FIELD;
import static java.lang.annotation.ElementType.METHOD;
import static java.lang.annotation.ElementType.PARAMETER;
import static java.lang.annotation.RetentionPolicy.RUNTIME;

@Documented
@Constraint(validatedBy = {})// (1)
@Target({ METHOD, FIELD, ANNOTATION_TYPE, CONSTRUCTOR, PARAMETER })
@Retention(RUNTIME)
@ReportAsSingleViolation // (2)
@Pattern(regexp = "[a-zA-Z0-9]*") // (3)
public @interface AlphaNumeric {
    String message() default "{com.example.common.validation.AlphaNumeric.
↪message}"; // (4)

    Class<?>[] groups() default {};

    Class<? extends Payload>[] payload() default {};

    @Target({ METHOD, FIELD, ANNOTATION_TYPE, CONSTRUCTOR, PARAMETER })
    @Retention(RUNTIME)
    @Documented
    public @interface List {
        AlphaNumeric[] value();
    }
}
```

Sr. No.	Description
(1)	When implementing using existing annotations, `validatedBy`needs to be empty
(2)	This will consolidate error messages and return only the message of this annotation at the time of error.
(3)	Define rules used by this annotation.
(4)	Define default value of error message.

- Implementation example of `@NotNegative` annotation which is restricted to positive number

```
package com.example.common.validation;

import java.lang.annotation.Documented;
import java.lang.annotation.Retention;
import java.lang.annotation.Target;
import javax.validation.Constraint;
import javax.validation.Payload;
import javax.validation.ReportAsSingleViolation;
import javax.validation.constraints.Min;

import static java.lang.annotation.ElementType.ANNOTATION_TYPE;
import static java.lang.annotation.ElementType.CONSTRUCTOR;
import static java.lang.annotation.ElementType.FIELD;
import static java.lang.annotation.ElementType.METHOD;
import static java.lang.annotation.ElementType.PARAMETER;
import static java.lang.annotation.RetentionPolicy.RUNTIME;

@Documented
@Constraint(validatedBy = {})
@Target({ METHOD, FIELD, ANNOTATION_TYPE, CONSTRUCTOR, PARAMETER })
@Retention(RUNTIME)
@ReportAsSingleViolation
@Min(value = 0)
public @interface NotNegative {
    String message() default "{com.example.common.validation.NotNegative.
↪message}";

    Class<?>[] groups() default {};
```

```
    Class<? extends Payload>[] payload() default {};

    @Target({ METHOD, FIELD, ANNOTATION_TYPE, CONSTRUCTOR, PARAMETER })
    @Retention(RUNTIME)
    @Documented
    public @interface List {
        NotNegative[] value();
    }
}
```

- Implementation example of `@UserId` annotation which regulates the format of "User ID".

```
package com.example.sample.domain.validation;

import java.lang.annotation.Documented;
import java.lang.annotation.Retention;
import java.lang.annotation.Target;
import javax.validation.Constraint;
import javax.validation.Payload;
import javax.validation.ReportAsSingleViolation;
import javax.validation.constraints.Pattern;
import javax.validation.constraints.Size;

import static java.lang.annotation.ElementType.ANNOTATION_TYPE;
import static java.lang.annotation.ElementType.CONSTRUCTOR;
import static java.lang.annotation.ElementType.FIELD;
import static java.lang.annotation.ElementType.METHOD;
import static java.lang.annotation.ElementType.PARAMETER;
import static java.lang.annotation.RetentionPolicy.RUNTIME;

@Documented
@Constraint(validatedBy = {})
@Target({ METHOD, FIELD, ANNOTATION_TYPE, CONSTRUCTOR, PARAMETER })
@Retention(RUNTIME)
@ReportAsSingleViolation
@Size(min = 4, max = 20)
@Pattern(regexp = "[a-z]*")
public @interface UserId {
    String message() default "{com.example.sample.domain.validation.UserId.
↪message}";

    Class<?>[] groups() default {};

    Class<? extends Payload>[] payload() default {};

    @Target({ METHOD, FIELD, ANNOTATION_TYPE, CONSTRUCTOR, PARAMETER })
    @Retention(RUNTIME)
    @Documented
    public @interface List {
        UserId[] value();
```

```
    }
}
```

- Implementation example of `@Age` annotation which regulates the constraints on "Age"

```
package com.example.sample.domain.validation;

import java.lang.annotation.Documented;
import java.lang.annotation.Retention;
import java.lang.annotation.Target;
import javax.validation.Constraint;
import javax.validation.Payload;
import javax.validation.ReportAsSingleViolation;
import javax.validation.constraints.Max;
import javax.validation.constraints.Min;

import static java.lang.annotation.ElementType.ANNOTATION_TYPE;
import static java.lang.annotation.ElementType.CONSTRUCTOR;
import static java.lang.annotation.ElementType.FIELD;
import static java.lang.annotation.ElementType.METHOD;
import static java.lang.annotation.ElementType.PARAMETER;
import static java.lang.annotation.RetentionPolicy.RUNTIME;

@Documented
@Constraint(validatedBy = {})
@Target({ METHOD, FIELD, ANNOTATION_TYPE, CONSTRUCTOR, PARAMETER })
@Retention(RUNTIME)
@ReportAsSingleViolation
@Min(1)
@Max(150)
public @interface Age {
    String message() default "{com.example.sample.domain.validation.Age.
↪message}";

    Class<?>[] groups() default {};

    Class<? extends Payload>[] payload() default {};

    @Target({ METHOD, FIELD, ANNOTATION_TYPE, CONSTRUCTOR, PARAMETER })
    @Retention(RUNTIME)
    @Documented
    public @interface List {
        Age[] value();
    }
}
```

Note: If multiple rules are set in a single annotation, their AND condition forms the composite annotation. In Hibernate Validator, `@ConstraintComposition` annotation is provided to implement OR

condition. Refer to Hibernate Validator document for details.

How to check the values in the collection using Bean Validation's annotation

When dealing with multiple selectable screen items (such as checkboxes and multiple selection dropdown), it is common to treat screen items as collections of basic types such as `String`in form classes. In the standard specification of Bean Validation, each value in the collection can not be checked by using Bean Validation annotation, but by using original features of Java SE 8 and Hibernate Validator, or if original functions of Java SE 8 and Hibernate Validator are not used, a wrapper class for the value of the screen item can be created and it can be treated as a collection so that the value in the collection using Bean Validation can be checked.

Here, annotations to check whether the input value provided by the common library is a code value defined in the code list, input checking for Strings in the collection using `org.terasoluna.gfw.common.codelist.ExistInCodeList`is explained as an example.

Implementation method for matching `@ExistInCodeList`annotation with multiple selectable screen items (check boxes, multiple selection drop down, etc.) is shown below.

- *Implementation with Java SE 8 and Hibernate Validator 5.2+* Method of implementing original annotations that can be added to `List`of `String`by using the original functions of Java SE 8 and Hibernate Validator. **It is easier and simpler to implement than the latter, so we recommend this method in environments where the original features of Java SE 8 and Hibernate Validator can be used.** In addition, this method will be standardized in the future version of Bean Validation in future.
- *Implementation of wrapper class of String using Java bean* Method of setting the `@ExistInCodeList`annotation for the property wrapped in the Java bean class. For users who do not use Java SE 8 and Hibernate Validator's original features. In environments where the original features of Java SE 8 and Hibernate Validator can be used, *Implementation with Java SE 8 and Hibernate Validator 5.2+*is recommended.

Implementation with Java SE 8 and Hibernate Validator 5.2+

Here,by creating original annotation `@ExistInCodeList`provided by the common library is wrapped, and how to make it correspond to collection is shown.

`java.lang.annotation.ElementType.TYPE_USE`has been added in Java SE 8. This makes it possible to add annotations to not only declarations of conventional classes and methods, but also general types (types of local variables, etc.), Hibernate Validator 5.2+ for Java SE 8 supports constraint annotations attached to param-

eterized types such as `Collection`, `Map`, `Optional`.By reading it, it is possible to check the values in the collection.

By combining Java SE 8 with Hibernate Validator 5.2+, like `List<@NotNullForTypeArgument String>`, Annotations that can be added to the type specification part of the list can be created, and the values in the collection can be checked. For details,please refer Hibernate Validator document(Type argument constraints).

`@ExistInCodeList`provided by the common library does not correspond to `TYPE_USE`because it is Java SE 7 compatible, By creating original annotation that can be added to the type specifying part of the list as described above, it is possible to check the value in the collection.

The main steps are as follows.

- Use "TYPE_USE" and implement annotations extending `@ExistInCodeList`which can be added to the type usage place.
- Set the annotation to be checked.

Input check for Role (`List`of `String`) that can set multiple items is used as an example to explain.

Multiple item configurable Role

Implementation example of `@ExistInCodeListForTypeArgument`which can be added to type usage.

```
package com.example.common.validation;

import static java.lang.annotation.ElementType.TYPE_USE;
import static java.lang.annotation.RetentionPolicy.RUNTIME;
import java.lang.annotation.Documented;
import java.lang.annotation.Retention;
import java.lang.annotation.Target;
import javax.validation.Constraint;
import javax.validation.OverridesAttribute;
import javax.validation.Payload;
import javax.validation.ReportAsSingleViolation;
import org.terasoluna.gfw.common.codelist.ExistInCodeList;

@Documented
@Constraint(validatedBy = {})
@Target(TYPE_USE) // (1)
@Retention(RUNTIME)
@ReportAsSingleViolation
@ExistInCodeList(codeListId = "") // (2)
public @interface ExistInCodeListForTypeArgument {
    String message() default "{com.example.common.validation.
↪ExistInCodeListForTypeArgument.message}"; // (3)

    @OverridesAttribute(constraint = ExistInCodeList.class, name = "codeListId
↪") // (4)
    String codeListId();
```

```
    Class<?>[] groups() default {};

    Class<? extends Payload>[] payload() default {};

    @Target(TYPE_USE) // (1)
    @Retention(RUNTIME)
    @Documented
    @interface List {
        ExistInCodeListForTypeArgument[] value();
    }
}
```

Sr.No.	Description
(1)	Set `TYPE_USE`, so that this annotation can be added at the type usage place.
(2)	Define the rules (`@ExistInCodeList`) used by this annotation.
(3)	Define the default value of the error message. Also define any error message in ValidationMessages.properties.
(4)	Override the `codeListId` attribute of the `@ExistInCodeList`annotation.

- Form class

```
package com.example.sample.app.validation;

import java.util.List;

import javax.validation.constraints.NotNull;

import com.example.common.validation.ExistInCodeListForTypeArgument;

public class SampleForm {
    @NotNull
    @Valid // (1)
```

```
    private List<@ExistInCodeListForTypeArgument(codeListId = "CL_ROLE")␣
↪String> roles; // (2)

    public List<String> getRoles() {
        return roles;
    }

    public void setRoles(List<String> roles) {
        this.roles = roles;
    }
}
```

Sr.No.	Description
(1)	Give `javax.validation.Valid`annotation to enable input checking of element values in `List`
(2)	Set the `@ExistInCodeListForTypeArgument`annotation for the type specification part of the collection to be checked for input. Specify the code list as the check source in the `codeListId` parameter of the annotation.

- JSP

```
<form:form modelAttribute="sampleForm">
    <!-- (1) -->
    <form:checkboxes path="roles" items="${CL_ROLE}"/>
    <form:errors path="roles*"/>
    <form:button>Submit</form:button>
</form:form>
```

Sr.No.	Description
(1)	Implement `<form:checkboxes>`.

Implementation of wrapper class of String using Java bean

The implementation introduced here is for users who do not use Java SE 8 and Hibernate Validator's original functions. In environments where the original features of Java SE 8 and Hibernate Validator can be used, *Implementation with Java SE 8 and Hibernate Validator 5.2+*is recommended.

Since you can not use Bean Validation annotation for elements in the collection as described above when Java SE 8 and Hibernate Validator's original functions are not used, `String`is wrapped with a Java bean and input check is done by appending `@ExistInCodeList`to the property of the nested bean.

When using a Spring-supplied tag library for a wrapper class, it is necessary to perform type conversion between a character string and a wrapper class in order to bind to a form. This is a *type conversion mechanism provided by Spring (Formatter) <http://docs.spring.io/spring/docs/4.3.5.RELEASE/spring-framework-reference/htmlsingle/#format>* _ is used for implementation.

By adding a type conversion from `String`to `Role`, `Role`to `String`in Formatter,as with `List<String>`, it can be implemented using `<form:checkboxes>`without complicated implementation.

The main steps are as follows.

- Set the `@ExistInCodeList`annotation for checking.
- Create a class that implements the `Formatter`class which is the interface for conversion.
- Using `ConversionServiceFactoryBean`register the created `Formatter`in Spring.

Also, in order to display items that have been successfully selected with `<form:checkboxes>`, you need to override the `toString`method of the wrapper class in addition to the Formatter implementation .

`<form:checkboxes>`displays the elements of the collection specified by the `items`attribute as selected items, if the value of the selected item specified by the `path`attribute matches with the value of the property, it is displayed as the selected item. To determine this consistency, Formatter is used when the property is a single item, and in the case of an array or collection, the result of the `toString`method of the specified property is used.

In order to display it as a selected item successfully, you need to override the `toString`method in a wrapper class and return the wrapped value string as in the example below.

Note: For details, please refer the actual determination method of selected `org.springframework.web.servlet.tags.form.SelectedValueComparator`class's javadoc

---

Input check for `Role`(Java Bean's `List`) that can set multiple items is used as an example for explaination.

- Form class

```
package com.example.sample.app.validation;

import java.util.List;

import javax.validation.Valid;
import javax.validation.constraints.NotNull;

import com.example.sample.domain.model.Role;

public class SampleForm {
    @NotNull
    @Valid // (1)
    private List<Role> roles; // (2)

    public List<Role> getRoles() {
        return roles;
    }

    public void setRoles(List<Role> roles) {
        this.roles = roles;
    }
}
```

Sr.No.	Description
(1)	`javax.validation.Valid`annotation to enable bean validation of nested beans.
(2)	You cannot add `@ExistInCodeList`to `List`of `String`but by wrapping `String`with a Java bean you can add `@ExistInCodeList`to the nested bean's `String`property.

- JavaBean class

```
package com.example.sample.domain.model

import org.terasoluna.gfw.common.codelist.ExistInCodeList;

public class Role {
    @ExistInCodeList(codeListId = "CL_ROLE") // (1)
    private String value;
```

```
    public String getValue() {
        return value;
    }

    public void setValue(String value) {
        this.value = value;
    }

    @Override // (2)
    public String toString() {
        return getValue();
    }
}
```

Sr.No.	Description
(1)	Set the `@ExistInCodeList`annotation for the property wrapped in the `Role`class to perform input check, and specify the code list to check `codeListId`.
(2)	Override in order to draw the screen properly when multiple selection is made and return the character string of the wrapped value.

Implement the Formatter class that performs type conversion and register it in Spring.

As mentioned above, it is necessary to wrap `String`with `Role`(Java bean) to perform input checking. Implement type conversion by `Formatter`to convert from screen input (`String`) to wrapped `Role`and vice versa.

By adding type conversion, mutual conversion between `String`and `Role`is done automatically. On the Controller side, it is treated as `List`of `Role`, on JSP side `List`of `String`.

- `Formatter`class

  Implementation of `Formatter`which performs mutual conversion between `String`and `Role`.

```
package com.example.sample.app.validation.formatter;

import java.text.ParseException;
import java.util.Locale;

import org.springframework.format.Formatter;
```

```
import com.example.usermanagement.domain.model.Role;

public class RoleFormatter implements Formatter<Role> { //(1)

    @Override
    public String print(Role source, Locale locale) {
        return source.getValue();
    }

    @Override
    public Role parse(String source, Locale locale) throws ParseException {
        Role role = new Role();
        role.setValue(source);
        return role;
    }

}
```

Sr.No.	Description
(1)	Implement the interface `org.springframework.format.Formatter<T>`.

- Bean definition for applying original `Formatter`

```
<!-- (1) -->
<bean id="conversionService"
    class="org.springframework.format.support.
↪FormattingConversionServiceFactoryBean">
    <property name="formatters">
        <list>
            <!-- (2) -->
            <bean class="com.example.sample.app.validation.formatter.
↪RoleFormatter" />
        </list>
    </property>
</bean>
```

```
<!-- (3) -->
<mvc:annotation-driven conversion-service="conversionService">
    <!-- omitted -->
</mvc:annotation-driven>
```

Sr.No.	Description
(1)	Add bean definition of `FormattingConversionServiceFactoryBean`.
(2)	Set the created Formatter class(`RoleFormatter`).
(3)	To use the customized type conversion, set the bean defined in (1) to the `conversion-service`attribute of `mvc:annotation-driven`.

- JSP

```
<form:form modelAttribute="sampleForm">
    <!-- (1) -->
    <form:checkboxes path="roles" items="${CL_ROLE}"/>
    <form:errors path="roles*"/>
    <form:button>Submit</form:button>
</form:form>
```

Sr.No.	Description
(1)	`<form:checkboxes>`can be used same as `List<String>`.

Creation of Bean Validation annotation by implementing new rules

Any rule can be created by implementing `javax.validation.ConstraintValidator` interface and creating annotation that uses this Validator.

The following 3 ways can be cited as applications.

- Rules that cannot be implemented by combining the existing rules
- check rule for correlated items
- Business logic check

Rules that cannot be implemented by combining the existing rules

For the rules that cannot be implemented by combining `@Pattern`, `@Size`, `@Min`, `@Max`, implement `javax.validation.ConstraintValidator`.

For example, rules that check ISBN (International Standard Book Number)-13 format are given.

- Annotation

```
package com.example.common.validation;

import java.lang.annotation.Documented;
import java.lang.annotation.Retention;
import java.lang.annotation.Target;
import javax.validation.Constraint;
import javax.validation.Payload;
import static java.lang.annotation.ElementType.ANNOTATION_TYPE;
import static java.lang.annotation.ElementType.CONSTRUCTOR;
import static java.lang.annotation.ElementType.FIELD;
import static java.lang.annotation.ElementType.METHOD;
import static java.lang.annotation.ElementType.PARAMETER;
import static java.lang.annotation.RetentionPolicy.RUNTIME;

@Documented
@Constraint(validatedBy = { ISBN13Validator.class }) // (1)
@Target({ METHOD, FIELD, ANNOTATION_TYPE, CONSTRUCTOR, PARAMETER })
@Retention(RUNTIME)
public @interface ISBN13 {
    String message() default "{com.example.common.validation.ISBN13.message}";

    Class<?>[] groups() default {};

    Class<? extends Payload>[] payload() default {};

    @Target({ METHOD, FIELD, ANNOTATION_TYPE, CONSTRUCTOR, PARAMETER })
    @Retention(RUNTIME)
    @Documented
    public @interface List {
        ISBN13[] value();
    }
}
```

Sr. No.	Description
(1)	Specify the `ConstraintValidator` implementation class which will get executed when this annotation is used. Multiple constraints can also be specified.

- Validator

```
package com.example.common.validation;

import javax.validation.ConstraintValidator;
import javax.validation.ConstraintValidatorContext;

public class ISBN13Validator implements ConstraintValidator<ISBN13, String> {␣
↪// (1)

    @Override
    public void initialize(ISBN13 constraintAnnotation) { // (2)
    }

    @Override
    public boolean isValid(String value, ConstraintValidatorContext context)
↪{ // (3)
        if (value == null) {
            return true; // (4)
        }
        return isISBN13Valid(value); // (5)
    }

    // This logic is written in http://en.wikipedia.org/wiki/International_
↪Standard_Book_Number
    static boolean isISBN13Valid(String isbn) {
        if (isbn.length() != 13) {
            return false;
        }
        int check = 0;
        try {
            for (int i = 0; i < 12; i += 2) {
                check += Integer.parseInt(isbn.substring(i, i + 1));
            }
            for (int i = 1; i < 12; i += 2) {
                check += Integer.parseInt(isbn.substring(i, i + 1)) * 3;
            }
            check += Integer.parseInt(isbn.substring(12));
        } catch (NumberFormatException e) {
            return false;
        }
        return check % 10 == 0;
    }
}
```

Sr. No.	Description
(1)	Specify target annotation and field type in generics parameter.
(2)	Implement initialization process in `initialize` method.
(3)	Implement input validation in `isValid` method.
(4)	Consider input value as correct in case of `null`.
(5)	Check ISBN-13 format.

Tip: Example of *Bean Validation of file upload* is classified in this category. Further, in common library as well, *@ExistInCodeList* is implemented in this way.

Check rules for correlated items

Check of correlated items, which span over multiple fields, can be done using Bean Validation as explained in *Correlation item check*.

It is recommended to target generalized rules, in case of deciding to use Bean Validation for check of correlated items.

An example of implementing the rule, "Contents of a field should match with its confirmation field" is given below.

Set constraint of assigning "confirm" as the prefix of confirmation field.

- Annotation

  Define such that annotation for validation of correlated items can be used at class level as well.

```
package com.example.common.validation;

import java.lang.annotation.Documented;
```

```
import java.lang.annotation.Retention;
import java.lang.annotation.Target;
import javax.validation.Constraint;
import javax.validation.Payload;
import static java.lang.annotation.ElementType.ANNOTATION_TYPE;
import static java.lang.annotation.ElementType.TYPE;
import static java.lang.annotation.RetentionPolicy.RUNTIME;

@Documented
@Constraint(validatedBy = { ConfirmValidator.class })
@Target({ TYPE, ANNOTATION_TYPE }) // (1)
@Retention(RUNTIME)
public @interface Confirm {
    String message() default "{com.example.common.validation.Confirm.message}
↪";

    Class<?>[] groups() default {};

    Class<? extends Payload>[] payload() default {};

    /**
     * Field name
     */
    String field(); // (2)

    @Target({ TYPE, ANNOTATION_TYPE })
    @Retention(RUNTIME)
    @Documented
    public @interface List {
        Confirm[] value();
    }
}
```

Sr. No.	Description
(1)	Narrow down the target of using this annotation, such that it can be added only to class or annotation.
(2)	Define parameter to pass to annotation.

- Validator

```
package com.example.common.validation;

import javax.validation.ConstraintValidator;
import javax.validation.ConstraintValidatorContext;
```

```
import org.springframework.beans.BeanWrapper;
import org.springframework.beans.BeanWrapperImpl;
import org.springframework.util.ObjectUtils;
import org.springframework.util.StringUtils;

public class ConfirmValidator implements ConstraintValidator<Confirm, Object>
↪{
    private String field;

    private String confirmField;

    private String message;

    public void initialize(Confirm constraintAnnotation) {
        field = constraintAnnotation.field();
        confirmField = "confirm" + StringUtils.capitalize(field);
        message = constraintAnnotation.message();
    }

    public boolean isValid(Object value, ConstraintValidatorContext context) {
        BeanWrapper beanWrapper = new BeanWrapperImpl(value); // (1)
        Object fieldValue = beanWrapper.getPropertyValue(field); // (2)
        Object confirmFieldValue = beanWrapper.getPropertyValue(confirmField);
        boolean matched = ObjectUtils.nullSafeEquals(fieldValue,
                confirmFieldValue);
        if (matched) {
            return true;
        } else {
            context.disableDefaultConstraintViolation(); // (3)
            context.buildConstraintViolationWithTemplate(message)
                    .addPropertyNode(field).addConstraintViolation(); // (4)
            return false;
        }
    }

}
```

Sr. No.	Description
(1)	Use `org.springframework.beans.BeanWrapper` which is very convenient to access JavaBean properties.
(2)	Acquire property value from the form object through `BeanWrapper`.
(3)	Disable the creation of default `ConstraintViolation` object.
(4)	Create an independent `ConstraintViolation` object. Define message to be output in `ConstraintValidatorContext.buildConstraintViolationWithTemplate`. Specify field name to output error message in `ConstraintViolationBuilder.addPropertyNode`. Refer to JavaDoc of ConstraintValidatorContext for details.

Tip: `ConstraintViolationBuilder.addPropertyNode` method has been added from the Bean Validation 1.1.

`ConstraintViolationBuilder.addNode` method had been used in Bean Validation 1.0; however, it is a deprecated API from Bean Validation 1.1.

For deprecated API of Bean Validation, refer to Bean Validation API Document (Deprecated API).

Note: As introduced in correlation item check using Spring Validator, even in Bean Validation, *Set error information for multiple fields for correlation check* can be performed.

An example is shown below wherein styles are applied to `password`field and `confirmPassword`field in Bean Validation and error message is displayed only for `password`field.

```
// omitted
public class ConfirmValidator implements ConstraintValidator<Confirm, Object>
 ↪{
    private String field;

    private String confirmField;
```

```
    private String message;

    public void initialize(Confirm constraintAnnotation) {
        // omitted
    }

    public boolean isValid(Object value, ConstraintValidatorContext context) {
        // omitted
        if (matched) {
            return true;
        } else {
            context.disableDefaultConstraintViolation();

            //new ConstraintViolation to be generated for field
            context.buildConstraintViolationWithTemplate(message)
                    .addPropertyNode(field).addConstraintViolation();

            //new ConstraintViolation to be generated for confirmField
            context.buildConstraintViolationWithTemplate("") // (1)
                    .addPropertyNode(confirmField).addConstraintViolation();

            return false;
        }
    }

}
```

Sr. No.	Description
(1)	Register error of `confirmPassword`field. A null string is set in error message at that time.

Check below for the changes, if the “Reset password” is re-implemented using `@Confirm` annotation.

- Form class

```
package com.example.sample.app.validation;

import java.io.Serializable;

import javax.validation.constraints.NotNull;
import javax.validation.constraints.Size;

import com.example.common.validation.Confirm;

@Confirm(field = "password") // (1)
public class PasswordResetForm implements Serializable {
    private static final long serialVersionUID = 1L;
```

```
    @NotNull
    @Size(min = 8)
    private String password;

    private String confirmPassword;

    // omitted getter/setter
}
```

Sr. No.	Description
(1)	Assign `@Confirm` annotation at class level. By this, form object is passed as an argument to `ConstraintValidator.isValid`.

- Controller class

There is no need to inject Validator and add Validator using `@InitBinder`.

```
package com.example.sample.app.validation;

import org.springframework.stereotype.Controller;
import org.springframework.validation.BindingResult;
import org.springframework.validation.annotation.Validated;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

@Controller
@RequestMapping("password")
public class PasswordResetController {

    @ModelAttribute
    public PasswordResetForm setupForm() {
        return new PasswordResetForm();
    }

    @RequestMapping(value = "reset", method = RequestMethod.GET, params =
↪"form")
    public String resetForm() {
        return "password/resetForm";
    }

    @RequestMapping(value = "reset", method = RequestMethod.POST)
    public String reset(@Validated PasswordResetForm form, BindingResult␣
↪result) {
        if (result.hasErrors()) {
            return "password/resetForm";
```

```
        }
        return "redirect:/password/reset?complete";
    }

    @RequestMapping(value = "reset", method = RequestMethod.GET, params =
↪"complete")
    public String resetComplete() {
        return "password/resetComplete";
    }
}
```

Business logic check

Business logic check should implement *Implementation in Service of domain layer* and store result message in `ResultMessages` object.

Accordingly, it is expected that, *they will be normally displayed at the top of the screen*.

However, there are cases, where business logic error message (such as, "whether the entered user name is already registered") of target input field is to be displayed next to the field. In such a case, service class is injected in Validator class and business logic check is executed in `ConstraintValidator.isValid`.

An example of implementing, "whether the entered user name is already registered" in Bean Validation is shown below.

- Service class

  Implementation class (UserServiceImpl) is omitted.

```
package com.example.sample.domain.service.user;

public interface UserService {

    boolean isUnusedUserId(String userId);

    // omitted other methods
}
```

- Annotation

```
package com.example.sample.domain.validation;

import java.lang.annotation.Documented;
import java.lang.annotation.Retention;
import java.lang.annotation.Target;
import javax.validation.Constraint;
import javax.validation.Payload;
```

```
import static java.lang.annotation.ElementType.ANNOTATION_TYPE;
import static java.lang.annotation.ElementType.CONSTRUCTOR;
import static java.lang.annotation.ElementType.FIELD;
import static java.lang.annotation.ElementType.METHOD;
import static java.lang.annotation.ElementType.PARAMETER;
import static java.lang.annotation.RetentionPolicy.RUNTIME;

@Documented
@Constraint(validatedBy = { UnusedUserIdValidator.class })
@Target({ METHOD, FIELD, ANNOTATION_TYPE, CONSTRUCTOR, PARAMETER })
@Retention(RUNTIME)
public @interface UnusedUserId {
    String message() default "{com.example.sample.domain.validation.
↪UnusedUserId.message}";

    Class<?>[] groups() default {};

    Class<? extends Payload>[] payload() default {};

    @Target({ METHOD, FIELD, ANNOTATION_TYPE, CONSTRUCTOR, PARAMETER })
    @Retention(RUNTIME)
    @Documented
    public @interface List {
        UnusedUserId[] value();
    }
}
```

- Validator class

```
package com.example.sample.domain.validation;

import javax.inject.Inject;
import javax.validation.ConstraintValidator;
import javax.validation.ConstraintValidatorContext;

import org.springframework.stereotype.Component;

import com.example.sample.domain.service.user.UserService;

@Component // (1)
public class UnusedUserIdValidator implements
                                    ConstraintValidator<UnusedUserId, String> {

    @Inject // (2)
    UserService userService;

    @Override
    public void initialize(UnusedUserId constraintAnnotation) {
    }
```

```
    @Override
    public boolean isValid(String value, ConstraintValidatorContext context) {
        if (value == null) {
            return true;
        }

        return userService.isUnusedUserId(value); // (3)
    }

}
```

Sr. No.	Description
(1)	Settings to enable component scan of the Validator class. Target package must be included in `<context:component-scan base-package="..." />` of Bean definition file.
(2)	Inject the service class to be called.
(3)	Return the result of business logic error check. Process should be delegated to service class. Logic must not be written directly in the `isValid` method.

Method Validation

A method to check validity of actual argument and return value of method using Bean Validation, is described. In order to explain, the method is called Method Validation in this chapter. While performing defensive programming etc, method I/O should be checked in the class other than Controller. If Bean Validation library is used at that time, constraint annotation of Bean Validation used in Controller can be reused.

Application settings

When Method Validation offered by Spring Framework is used, a bean must be defined for `org.springframework.validation.beanvalidation.MethodValidationPostProcessor` class offered by Spring Framework.

Bean definition file which defines `MethodValidationPostProcessor` differs depending on where you use

the Method Validation.

Here, a setup example is given wherein Method Validation is used in the multi-project environment recommended in this guideline.

Setup for both the projects below must be changed

- Project (`projectName-web`) for application layer
- Project (`projectName-domain`) for domain layer
- `projectName-domain/src/main/resources/META-INF/spring/projectName-domain.xml`

```
<!-- (1) -->
<bean id="validator"
      class="org.springframework.validation.beanvalidation.
↪LocalValidatorFactoryBean"/>

<!-- (2) -->
<bean class="org.springframework.validation.beanvalidation.
↪MethodValidationPostProcessor">
    <property name="validator" ref="validator" />
</bean>
```

- `projectName-web/src/main/resources/META-INF/spring/spring-mvc.xml`

```
<!-- (3) -->
<mvc:annotation-driven validator="validator">
    <!-- ... -->
</mvc:annotation-driven>

<!-- (4) -->
<bean class="org.springframework.validation.beanvalidation.
↪MethodValidationPostProcessor">
    <property name="validator" ref="validator" />
</bean>
```

Sr. No.	Description
(1)	Define a bean for `LocalValidatorFactoryBean`.
(2)	Define a bean for `MethodValidationPostProcessor` and ensure that Method Validation is executed for a domain layer class method. Specify a bean defined in (1), in `validator` property.
(3)	Specify a bean defined in (1), in the `validator` attribute of `<mvc:annotation-driven>` element. A `Validator` instance is generated that is different from the instance created in (1) in the absence of this setup.
(4)	Define a bean for `MethodValidationPostProcessor` and ensure that Method Validation is executed for an application layer class method. Specify Bean defined in (1), in `validator` property.

Tip: `LocalValidatorFactoryBean` is a class to generate a wrapper `Validator` object in order to link `Validator` class provided by Bean Validation(Hibernate Validator) and Spring Framework.

By using the wrapper `Validator` generated by this class, message management function (`MessageSource`) offered by Spring Framework and DI container can be linked.

Tip: In Spring Framework, Method Validation for calling the method of Bean which is managed by DI container is executed by using AOP system.

`MethodValidationPostProcessor` is a class to apply AOP in order to execute Method Validation.

Note: In the example above, an identical `Validator` object (instance) is set for `validator` property of each Bean, but this is not necessarily required. However, it is recommended to set an identical object (instance) unless there is a reason to do otherwise.

How to define for the method for Method Validation target

When Method Validation is applied to the method, annotation which indicates inclusion of target method and the constraint annotation of Bean Validation should be specified in class level and, method and dummy argument respectively.

AOP which executes Method Validation is not applicable for "*Application settings*". It is necessary to assign `@ org.springframework.validation.annotation.Validated` annotation to interface or class in order to apply AOP which executes Method Validation.

Here, a method to specify an annotation for interface is introduced.

```
package com.example.domain.service;

import org.springframework.validation.annotation.Validated;

@Validated // (1)
public interface HelloService {
    // ...
}
```

Sr. No.	Description
(1)	Specify `Validated` annotation in the interface acting as a target for Method Validation. In the example above, AOP which executes Method Validation is applied to implementation method of `HelloService` interface.

Tip: By specifying a group interface in `value` attribute of `@Validated` annotation, a validation belonging to a specified group can alone be executed as well.

Further, the validation groups can be changed for each method by assigning `Validated` annotation in method level.

Refer "*Grouped validation*" for validation group.

Next, a method is described wherein constraint annotation of Bean Validation is specified in the method and dummy argument. Basically,

Specify constraint annotation of Bean Validation for

- Method arguments
- JavaBean field specified in method argument

and constraint annotation of Bean Validation for

- Return value of method
- JavaBean field returned as return value of method.

A basic specification method is described below. A method to specify annotation in the interface is introduced in the description hereafter.

First, a method that specifies constraint annotation is described for the method using basic types (primitive type or primitive wrapper type etc) as a signature of the method

```
package com.example.domain.service;

import org.springframework.validation.annotation.Validated;

import javax.validation.constraints.NotNull;

@Validated
public interface HelloService {

    // (2)
    @NotNull
    String hello(@NotNull /* (1) */ String message);

}
```

Sr. No.	Description
(1)	Specify a constraint annotation of Bean Validation as an argument annotation of method. `@NotNull` is a constraint that signifies that `message` argument does not allow Null value. When Null value is specified in the argument, `javax.validation.ConstraintViolationException` is thrown.
(2)	Specify constraint annotation of Bean Validatio as a method annotation. In the above example, it is shown that the return value is not a Null value, when Null value is returned as a return value, `javax.validation.ConstraintViolationException` is thrown.

Next, a method that specifies constraint annotation of Bean Validation is described for the method using JavaBean as a signature of method.

Here, a method which species annotation for interface is introduced.

Note: The main point is to specify `@javax.validation.Valid` annotation. The specification method is

described below in detail using sample code.

Service interface

```
package com.example.domain.service;

import org.springframework.validation.annotation.Validated;

import javax.validation.constraints.NotNull;

@Validated
public interface HelloService {

    @NotNull // (3)
    @Valid   // (4)
    HelloOutput hello(@NotNull /* (1) */ @Valid /* (2) */ HelloInput input);

}
```

Sr. No.	Description
(1)	Specify constraint annotation of Bean Validation as an argument annotation. It indicates that `input` argument (JavaBean) does not allow Null value. When Null value is specified in the argument, `javax.validation.ConstraintViolationException` is thrown.
(2)	Specify `@javax.validation.Valid` annotation as an argument annotation of method. By assigning `@Valid` annotation, constraint annotation of Bean Validation specified in JavaBean field of argument becomes valid. When the constraint specified in JavaBean is not fulfilled, `javax.validation.ConstraintViolationException` is thrown.
(3)	Specify constraint annotation of Bean Validation as a method annotation. It indicates that JavaBean of return value is not Null value. When Null value is returned as a return value, an exception is thrown.
(4)	Specify `@Valid` annotation as a method annotation. By assigning `@Valid` annotation, constraint annotation of Bean Validation specified in JavaBean field of return value becomes valid. When constraint specified in JavaBean is not fulfilled, `javax.validation.ConstraintViolationException` is thrown.

Implementation sample of JavaBean is introduced below.

Basically, only constraint annotation of Bean Validation is specified, however care must be taken while performing further nesting of JavaBean in JavaBean.

JavaBean for Input

```
package com.example.domain.service;

import javax.validation.constraints.NotNull;
import javax.validation.constraints.Past;
import java.util.Date;

public class HelloInput {

    @NotNull
    @Past
    private Date visitDate;

    @NotNull
    private String visitMessage;

    private String userId;

    // ...

}
```

JavaBean for Output

```
package com.example.domain.service;

import com.example.domain.model.User;

import java.util.Date;

import javax.validation.Valid;
import javax.validation.constraints.NotNull;
import javax.validation.constraints.Past;

public class HelloOutput {

    @NotNull
    @Past
    private Date acceptDate;

    @NotNull
    private String acceptMessage;

    @Valid // (5)
    private User user;

    // ...

}
```

JavaBean nested in JavaBean for Output

```
package com.example.domain.model;

import javax.validation.constraints.NotNull;
import javax.validation.constraints.Past;
import java.util.Date;

public class User {

    @NotNull
    private String userId;

    @NotNull
    private String userName;

    @Past
    private Date dateOfBirth;

    // ...

}
```

Sr. No.	Description
(5)	When constraint annotation of Bean Validation specified in nested JavaBean is to be enabled, specify `@Valid` annotation as field annotation. By assigning `@Valid` annotation, constraint annotation of Bean Validation specified in nested JavaBean field becomes valid. When constraint specified in nested JavaBean is not fulfilled, `javax.validation.ConstraintViolationException` is thrown.

Exception handling at the time of violation of constraint

When a constraint violation occurs, `javax.validation.ConstraintViolationException` is thrown.

When `ConstraintViolationException` is thrown, the method generated from stack trace can be identified, however, basic violation details cannot be identified.

An exception handling class should be created which outputs log by handling `ConstraintViolationException` exception in order to identify violation details.

An example to show how to create an exception handling class is given below.

```
package com.example.app;
```

```
import javax.validation.ConstraintViolationException;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;

@ControllerAdvice
public class ConstraintViolationExceptionHandler {

    private static final Logger log = LoggerFactory.
↪getLogger(ConstraintViolationExceptionHandler.class);

    // (1)
    @ExceptionHandler
    public String handleConstraintViolationException(ConstraintViolationException␣
↪e){
        // (2)
        if (log.isErrorEnabled()) {
            log.error("ConstraintViolations[\n{}\n]", e.getConstraintViolations());
        }
        return "common/error/systemError";
    }

}
```

Sr. No.	Description
(1)	Create a `@ExceptionHandler` method to handle `ConstraintViolationException`. Receive `ConstraintViolationException` as an argument of method.
(2)	Output violation details (`Set` of `ConstraintViolation`) in a log which are retained by `ConstraintViolationException`received by method argument.

Note: Refer to "*Implementing "@ControllerAdvice"*" for details of `@ControllerAdvice` annotation.

Warning: An error message can occur by using the `ConstraintViolation#getMessage`method, but since the message completion by Spring's function is not done, field names in error message at `{0}`cannot be embedded.

Instead, field names can be obtained with the `ConstraintViolation#getPropertyPath`method.

For message completion by Spring's function, please see the note of:ref:*Validation_message_in_validationmessages*.

For details about `ConstraintViolation`,please refer Hibernate Validator reference.

4.1.4 Appendix

Input validation rules provided by Hibernate Validator

Hibernate Validator provides additional validation annotations, in addition to the annotations defined in Bean Validation.

Refer to Herefor the annotation list that can be used for validation.

Bean Validation check rules

Bean Validation standard annotations(`javax.validation.*`) are shown below.

Refer to Chapter 7 Bean Validation specification for details.

Annotation	Target type	Description	Usage example
`@NotNull`	Arbitrary	Validates that the target field is not `null`.	`@NotNull` `private String id;`
`@Null`	Arbitrary	Validates that the target field is `null`. (Example: usage in group validation)	`@Null(groups={Update.cla` `private String id;`
`@Pattern`	`String`	Whether the target field matches with the regular expression (In case of Hibernate Validator implementation, it is also possible to use it with arbitrary implementation class of `CharSequence` interface)	`@Pattern(regexp = "[0-9]` `private String tel;`

次のページに続く

Table. 4.3 – 前のページからの続き

Annotation	Target type	Description	Usage example
`@Min`	`BigDecimal`, `BigInteger`, `byte`, `short`, `int`, `long` and wrapper (In case of Hibernate Validator implementation, it is also possible to use it with arbitrary implementation class of `CharSequence` interface, `Number` inherited class. However, only in cases where string can be converted to a number.)	Validate whether the value is greater than the minimum value.	Refer `@Max`

`@Max`	`BigDecimal`, `BigInteger`, `byte`, `short`, `int`, `long` and wrapper
	(In case of Hibernate Validator implementation, it is also possible to use it with arbitrary implementation class of `CharSequence` interface, `Number` inherited class. However, only in cases where string can be converted to a number.)
		Validate whether the value is less than the maximum value.	`@Min(1)`
			`@Max(100)`
			`private int quantity;`

`@DecimalMin`	`BigDecimal`, `BigInteger`, `String`, `byte`, `short`, `int`, `long` and wrapper (In case of Hibernate Validator implementation, it is also possible to use it with arbitrary implementation class of `CharSequence` interface, `Number` inherited class.)	Validate whether the Decimal value is greater than or equal to minimum value. By specifying `inclusive = false`, it is possible to change to an operation so as to validate whether the value is greater than the minimum value.	Refer `@DecimalMax`

次のページに続く

Table. 4.3 – 前のページからの続き

Annotation	Target type	Description	Usage example

`@DecimalMax`	`BigDecimal`, `BigInteger`, `String`, `byte`, `short`, `int`, `long` and wrapper (In case of Hibernate Validator implementation, it is also possible to use it with arbitrary implementation class of `CharSequence` interface, `Number` inherited class.)	Validate whether the decimal value is less than or equal to the maximum value.
		By specifying `inclusive = false`, it is possible to change to an operation so as to validate whether the value is less than the maximum value.
			`@DecimalMin("0.0")`
			`@DecimalMax("99999.99")`
			`private BigDecimal price`

`@Size`	`String`(string length), `Collection`(elements size), `Map`(elements size), Array(array length) (In case of Hibernate Validator implementation, it is also possible to use it with arbitrary implementation class of `CharSequence` interface.)	Validate whether elements length or size is between `min` and `max`. It is possible to omit `min` and `max`. In that case, `min=0` and `max=Integer.MAX_VALUE` by default.	`@Size(min=4, max=64)` `private String password;`
`@Digits`	`BigDecimal`, `BigInteger`, `String`, `byte`, `short`, `int`, `long` and wrapper	Check whether the value is a numerical value within the specified range. Specify the maximum number of integer digits permitted in the `integer` and the maximum value of decimal digits permitted in `fraction`.	`@Digits(integer=6, fract` `private BigDecimal price`
`@AssertTrue`	`boolean`,`Boolean`	Validate that the target field is `true`(Example: Whether it agrees to the terms)	`@AssertTrue` `private boolean checked;`
`@AssertFalse`	`boolean`,`Boolean`	Validate whether the target field is `false`	`@AssertFalse` `private boolean checked;`

次のページに続く

Table. 4.3 – 前のページからの続き

Annotation	Target type	Description	Usage example
`@Future`	`Date`, `Calendar`(In case of Hibernate Validator implementation, it is also applicable to Joda-Time class)	Validate whether it is a future date.	`@Future` `private Date eventDate;`
`@Past`	`Date`, `Calendar`(In case of Hibernate Validator implementation, it is also applicable to Joda-Time class)	Validate whether it is a past date.	`@Past` `private Date eventDate;`

`@Valid`	Optional non-primitive type	Validate recursively for related objects.	`@Valid`
			`private List<Employer> e`
			
			`@Valid`
			`private Dept dept;`

Tip: `inclusive` attribute of `@DecimalMin` and `@DecimalMax` annotation is, an attribute added from Bean Validation 1.1.

By specifying `true` (to allow same value of specified threshold) to the default value of `inclusive` attribute, compatibility with Bean Validation 1.0 is maintained.

Warning: In `@Size` annotation, characters represented by char type 2 (32 bits) called as surrogate pair are not considered.

When a string consisiting of a surrogate pair is excluded from the check, adequate care must be taken since number of characters that are counted are likely to be more than the actual number of characters.

For length of the string including surrogate pair, refer *Fetching string length*.

Hibernate Validator check rules

All the major annotations(`org.hibernate.validator.constraints.*`) of Hibernate Validator are shown below.

Refer to Hibernate Validator specifications for details.

Annotation	Target type	Description	Usage example
`@CreditCardNumber`	It is applicable to any implementation class of `CharSequence` interface	Validate whether the credit card number is valid as per Luhn algorithm. It does not necessarily check whether the credit card number is available. By specifying `ignoreNonDigitCharacters = true`, it is possible to validate by ignoring non-numeric characters.	`@CreditCardNumber` `private String cardNumbe`
`@Email`	It is applicable to any implementation class of `CharSequence` interface	Validate whether the Email address is complaint with RFC2822.	`@Email` `private String email;`
`@URL`	It is applicable to any implementation class of `CharSequence` interface	Verify that it is valid as URL. Character string is validated by using `java.net.URL` constructor and the protocol that is validated as a URL is dependent on the protocols supported by JVM (`http`,`https`,`file`,`jar` etc).	`@URL` `private String url;`
`@NotBlank`	It is applicable to any implementation class of `CharSequence` interface	Validate that it is not `null`, empty string (`""`) and space only.	`@NotBlank` `private String userId;`
`@NotEmpty`	It is applicable to `Collection`, `Map`, Array, and any implementation class of `CharSequence` interface	Validate that it is not `null` or empty. `@NotEmpty` should be used when check is to be done in `@NotNull` + `@Min(1)` combination.	`@NotEmpty` `private String password;`

Tip: In `@URL`, when the verification is to be performed for validating the protocol not supported by JVM, `org.hibernate.validator.constraintvalidators.RegexpURLValidator` provided by Hibernate is used. This class is a Validator class corresponding to `@URL` annotation and it can be checked whether it is a URL format by a regular expression. It can also verify the validation for the protocols not supported by JVM.

- If the check rules of `@URL` of overall application can be changed, Validator class is changed to `RegexpURLValidator` in XML, as described in JavaDoc.
- When only some of the items are to be validated by using regular expressions and default rules are to be used for `@URL`, a new annotation and an implementation class `javax.validation.ConstraintValidator` that performs validation similar to `RegexpURLValidator` are created and the validation by the annotation thus created is applied to the required items.

These can be applied as per the intended use.

For details of check rule change by XML, refer to Hibernate referenceand for how to create a new annotation, refer to *Creation of Bean Validation annotation by implementing new rules* respectively..

Default messages provided by Hibernate Validator

In hibernate-validator-<version>.jar, there is a ValidationMessages.properties file at org/hibernate/validator location, which contains the default messages of all Hibernate provided annotations.

```
javax.validation.constraints.AssertFalse.message = must be false
javax.validation.constraints.AssertTrue.message  = must be true
javax.validation.constraints.DecimalMax.message  = must be less than ${inclusive␣
↪== true ? 'or equal to ' : ''}{value}
javax.validation.constraints.DecimalMin.message  = must be greater than $
↪{inclusive == true ? 'or equal to ' : ''}{value}
javax.validation.constraints.Digits.message      = numeric value out of bounds (<
↪{integer} digits>.<{fraction} digits> expected)
javax.validation.constraints.Future.message      = must be in the future
javax.validation.constraints.Max.message         = must be less than or equal to
↪{value}
javax.validation.constraints.Min.message         = must be greater than or equal␣
↪to {value}
javax.validation.constraints.NotNull.message     = may not be null
javax.validation.constraints.Null.message        = must be null
javax.validation.constraints.Past.message        = must be in the past
javax.validation.constraints.Pattern.message     = must match "{regexp}"
javax.validation.constraints.Size.message        = size must be between {min} and
↪{max}

org.hibernate.validator.constraints.CreditCardNumber.message        = invalid␣
↪credit card number
org.hibernate.validator.constraints.EAN.message                     = invalid
↪{type} barcode
org.hibernate.validator.constraints.Email.message                   = not a well-
↪formed email address
org.hibernate.validator.constraints.Length.message                  = length must␣
↪be between {min} and {max}
```

```
org.hibernate.validator.constraints.LuhnCheck.message              = The check␣
↪digit for ${validatedValue} is invalid, Luhn Modulo 10 checksum failed
org.hibernate.validator.constraints.Mod10Check.message             = The check␣
↪digit for ${validatedValue} is invalid, Modulo 10 checksum failed
org.hibernate.validator.constraints.Mod11Check.message             = The check␣
↪digit for ${validatedValue} is invalid, Modulo 11 checksum failed
org.hibernate.validator.constraints.ModCheck.message               = The check␣
↪digit for ${validatedValue} is invalid, ${modType} checksum failed
org.hibernate.validator.constraints.NotBlank.message               = may not be␣
↪empty
org.hibernate.validator.constraints.NotEmpty.message               = may not be␣
↪empty
org.hibernate.validator.constraints.ParametersScriptAssert.message = script␣
↪expression "{script}" didn't evaluate to true
org.hibernate.validator.constraints.Range.message                  = must be␣
↪between {min} and {max}
org.hibernate.validator.constraints.SafeHtml.message               = may have␣
↪unsafe html content
org.hibernate.validator.constraints.ScriptAssert.message           = script␣
↪expression "{script}" didn't evaluate to true
org.hibernate.validator.constraints.URL.message                    = must be a␣
↪valid URL

org.hibernate.validator.constraints.br.CNPJ.message                = invalid␣
↪Brazilian corporate taxpayer registry number (CNPJ)
org.hibernate.validator.constraints.br.CPF.message                 = invalid␣
↪Brazilian individual taxpayer registry number (CPF)
org.hibernate.validator.constraints.br.TituloEleitoral.message     = invalid␣
↪Brazilian Voter ID card number
```

Input check rules provided by a common library

A common library provides an independent annotation for verification. Here, how to specify input check rules which use annotation provided by common library is explained.

terasoluna-gfw-common check rules

Annotation provided by terasoluna-gfw-common (`org.terasoluna.gfw.common.codelist.*`) is shown below.

Annotation	Target type	Description	Usage example
`@ExistInCodeList`	`Character` Implementation class of `CharSequence` (`String`, `StringBuilder`etc)	Verify whether value is incorporated in the code list.	Refer *@ExistInCodeList*

terasoluna-gfw-codepoints check rules

Annotation (`org.terasoluna.gfw.common.codepoints.*`) offered by terasoluna-gfw-codepoints is shown below. Further, `terasoluna-gfw-codepoints` can be used in 5.1.0.RELEASE and subsequent versions.

Annotation	Target type	Description	Usage example
`@ConsistOf`	Implementation class of `CharSequence` (`String`, `StringBuilder` etc)	Verify whether all the character strings to be checked are included in the specified code point set.	Refer *@ConsistOf*

terasoluna-gfw-validator check rules

Annotation (`org.terasoluna.gfw.common.validator.constraints.*`) offered by terasoluna-gfw-validator is shown below. Further, `terasoluna-gfw-validator`can be used in 5.1.0.RELEASE and subsequent versions.

Annotation	Target type	Description	Usage example
`@ByteMin`	Implementation class of `CharSequence` (`String`, `StringBuilder` etc)	Verify whether byte length value is greater than or equal to minimum value. **[Annotation attributes]** `long value` - Specify minimum value of byte length. `String charset` - Specify string character set used while encoding the value in byte sequence. Default value is `UTF-8`.	Refer `@ByteMax`

`@ByteMax`	Implementation class of `CharSequence` (`String`, `StringBuilder` etc)	Verify whether byte length value is less than or equal to maximum value.
		
		[Annotation attribute]
		`long value` - Specify maximum value for byte length.
		`String charset` - Specify string character set used while encoding the value in byte sequence. Default value is `UTF-8`.
			`@ByteMin(1)`
			`@ByteMax(value = 100,`
			`charset = "Shift`
			`private String id;`

次のページに続く

Table. 4.4 – 前のページからの続き

Annotation	Target type	Description	Usage example

`@Compare`	Implementation class of `Comparable` interface can be applied to any JavaBean with a property	Verify that the magnitude relation of specified property value is correct.
		
		[Annotation attribute]
		`String left` - Specify the property name to be used as a comparison source in the object. A message is displayed in the property in case of a validation error.
		`String right` - Specify property name to be used as a comparison destination in the object.
		
		`org.terasoluna.gfw.common.validator.constraints.C`
		`operator` - Specify Enum type `Operator` value which shows expected magnitude relation. Values that can be specified are as below.
		• `EQUAL` : `left = right`
		• `GREATER_THAN` : `left > right`
		• `GREATER_THAN_OR_EQUAL` : `left >= right`
		• `LESS_THAN` : `left < right`
		• `LESS_THAN_OR_EQUAL` : `left <= right`
		
		`boolean requireBoth` - Specify whether values must be entered in the fields specified by both `left` attribute and `right` attribute (should not be `null`).
			Check whether the mail address matches with the mail address entered for checking. Implementation is as below when entire form is to be displayed as an error message.
			`@Compare(left = "email", right = "confirm operator = Compa requireBoth = tr node = Compare.N`
			`public class UserRegiste private String email private String confi }`
			
			When period start date and period end date are both entered, check that start date is earlier than the end date. Implementation is as below when error message is displayed for period start date.
			`@Compare(left = "form", right = "to", operator = Compa`
			`public class Period { private Date from; private Date to; }`

		• `true` : Considered as a validation error if value is entered in only one of the fields. However, con-	

Table. 4.4 – 前のページからの続き

Annotation	Target type	Description	Usage example

Note: For mandatory input during correlated item check

For unit item check, whether a value is entered in the input field (should not be `null`) can be checked by using `@NotNull` in combination. However, in correlated item check, the check like "if value is entered in one field, value is entered forcefully in another field" cannot be implemented by using `@NotNull` alone. Hence, `@Compare` provides `requireBoth` attribute which controls mandatory input for checking which can then be used for implementing the check whenever required.

Also, when a value is not entered in the input field, `requireBoth` attribute can be used only when `null` is bound. If a form is sent in Spring MVC when a value is not entered in input field of string, it must be noted that empty string is bound in the form object by default instead of `null`. When a value is not entered in the string field, refer *Binding null to blank string field* to bind `null` in form object instead of empty string.

Expected check requirements and configuration example are shown below using "checking whether period start date is earlier than end date" as an example.

Check requirements	Configuration example

`from` and `to` both are mandatory, compare `from` and `to`.	Assign `@NotNull`in `from` and `to` and use default value （`false`) in `requireBoth` attribute.
	`@Compare(left = "from", right = "to", operator =`
	`public class Period {`
	`  @NotNull`
	`  LocalDate from;`
	`  @NotNull`
	`  LocalDate to;`
	`}`

Only `from` is mandatory, however carry out comparison check when a value is entered in `to` as well.	Assign `@NotNull`only in `from` and use default value （`false`) in `requireBoth` attribute.
	`@Compare(left = "from", right = "to", operator =`
	`public class Period {`
	`  @NotNull`
	`  LocalDate from;`
	`  LocalDate to;`
	`}`

次のページに続く	

Table. 4.5 – 前のページからの続き

Check requirements	Configuration example

from and to are not both required, carry out comparison check only when values are entered in both from and to. When the value is entered in only one of the fields, comparison check is not carried out.	Do not assign @NotNulland use default value (false) in requireBoth attribute.
	`@Compare(left = "from", right = "to", operator =`
	`public class Period {`
	`  LocalDate from;`
	`  LocalDate to;`
	`}`

from and to are not both required, when values are entered in either of the fromor to, carry out comparison check by always entering values in both fields.	Do not assign @NotNulland specify true in requireBoth attribute.
	`@Compare(left = "from", right = "to", operator =`
	`public class Period {`
	`  LocalDate from;`
	`  LocalDate to;`
	`}`

How to apply check rules of common library

Apply check rules of common library using the procedure below.

Add a dependent library for the rules that are to be used. An example for how to add `terasoluna-gfw-validator` is shown below.

```
<dependencies>
    <dependency>
        <groupId>org.terasoluna.gfw</groupId>
        <artifactId>terasoluna-gfw-validator</artifactId>
    </dependency>
</dependencies>
```

Note: In the above setting example, since it is assumed that the dependent library version is managed by the parent project terasoluna-gfw-parent , specifying the version in pom.xml is not necessary.

Next, a message definition corresponding to annotation is added to `ValidationMessages.properties` as explained in *Messages to be defined in ValidationMessages.properties*.

```
# (1)
org.terasoluna.gfw.common.validator.constraints.ByteMin.message = must be greater␣
↪than or equal to {value} Bytes
org.terasoluna.gfw.common.validator.constraints.ByteMax.message = must be less␣
↪than or equal to {value} Bytes
org.terasoluna.gfw.common.validator.constraints.Compare.message = not match '{left}
↪' and '{right}'
```

Sr. No.	Description
(1)	Add a message definition for each annotation. Annotation attribute value can be incorporated in the message by using placeholder (`{attribute name}` format).

In the end, assign an annotation to JavaBean property as explained in *Basic single item check*.

Note: When validation cannot be implemented in Bean Validation due to invalid attribute value of annotation, `javax.validation.ValidationException` is thrown. Modify the attribute value to a valid value by referring the reason that is output in stack trace.

Refer Chapter 9 of Bean Validation specification for details.

How to extend check rules of common library

Any rule can be created by using check rules provided by common library.

An example is introduced below wherein `@Confirm` annotation independently implemented by *Check rules for correlated items* is created by using check rules provided by common library.

Create `@Confirm` annotation by using `@Compare` as described in *Creation of Bean Validation annotation by combining existing rules*.

```
package com.example.sample.domain.validation;

import static java.lang.annotation.ElementType.ANNOTATION_TYPE;
import static java.lang.annotation.ElementType.TYPE;
import static java.lang.annotation.RetentionPolicy.RUNTIME;

import java.lang.annotation.Documented;
import java.lang.annotation.Retention;
import java.lang.annotation.Target;

import javax.validation.Constraint;
import javax.validation.OverridesAttribute;
import javax.validation.Payload;
import javax.validation.ReportAsSingleViolation;

import org.terasoluna.gfw.common.validator.constraints.Compare;

@Documented
@Constraint(validatedBy = {})
@Target({ TYPE, ANNOTATION_TYPE }) // (1)
@Retention(RUNTIME)
@ReportAsSingleViolation // (2)
@Compare(left = "", right = "", operator = Compare.Operator.EQUAL, requireBoth =␣
↪true) // (3)
public @interface Confirm {

    String message() default "{com.example.sample.domain.validation.Confirm.
↪message}"; // (4)

    Class<?>[] groups() default {};

    Class<? extends Payload>[] payload() default {};

    @OverridesAttribute(constraint = Compare.class, name = "left") // (5)
    String field();

    @OverridesAttribute(constraint = Compare.class, name = "right") // (6)
    String confirmField();

    @Documented
```

```
    @Target({ TYPE, ANNOTATION_TYPE })
    @Retention(RUNTIME)
    public @interface List {
        Confirm[] value();
    }
}
```

Sr. No.	Description
(1)	Restrict the location where the annotation can be assigned to class or annotation.
(2)	Message specified in `message` attribute of this annotation should be used at the time of error.
(3)	Specify `Compare.Operator.EQUAL`(should be equivalent value) in `operator` attribute of `@Compare` annotation. Specify `true` in `requireBoth`attribute since an error occurs if a value is not entered in either of the fields.
(4)	Define default value of error message.
(5)	Override `left` attribute of `@Compare` annotation and change attribute name to `field`.
(6)	Similarly, override `right` attribute and change attribute name to `confirmField`.

Use annotation created above instead of annotation implemented in *Check rules for correlated items*.

```
package com.example.sample.app.validation;

import java.io.Serializable;

import javax.validation.constraints.NotNull;
import javax.validation.constraints.Size;

import com.example.common.validation.Confirm;

@Confirm(field = "password", confirmField = "confirmPassword") // (1)
public class PasswordResetForm implements Serializable {
    private static final long serialVersionUID = 1L;
```

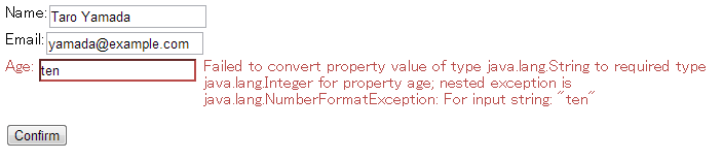
```
    @NotNull // (2)
    @Size(min = 8)
    private String password;

    @NotNull // (3)
    private String confirmPassword;

    // omitted geter/setter
}
```

Sr. No.	Description
(1)	Assign `@Confirm` annotation to class level.
(2)	Since `@Confirm` verification is passed when `password` field is `null`, perform `null` check by assigning `@NotNull` annotation.
(3)	Similarly assign `@NotNull` annotation in `confirmPassword` field as well.

Type mismatch

Pertaining the non-String fields of the form object, if values which cannot be converted to the corresponding data-type have been submitted, `org.springframework.beans.TypeMismatchException` is thrown.

In the "New user registration" example, The data-type of "Age" is `Integer`. However, if the value entered in this field cannot be converted to an integer, error message is displayed as follows.



Cause of exception is displayed as it is and is not appropriate as an error message. It is possible to define the error message for type mismatch in the properties file (application-messages.properties) which is read by `org.springframework.context.MessageSource`.

Error messages can be defined with the following rules.

Message key	Message contents	Application
`typeMismatch`	Default message for all type mismatch errors	Default value for entire system
`typeMismatch.[target FQCN]`	Default message for specific type mismatch error	Default value for entire system
`typeMismatch.[form attribute name].[property name]`	Type mismatch error message for specific form field	Customized message for each screen

When the following messages are added to application-messages.properties,

```
# typemismatch
typeMismatch="{0}" is invalid.
typeMismatch.int="{0}" must be an integer.
typeMismatch.double="{0}" must be a double.
typeMismatch.float="{0}" must be a float.
typeMismatch.long="{0}" must be a long.
typeMismatch.short="{0}" must be a short.
typeMismatch.java.lang.Integer="{0}" must be an integer.
typeMismatch.java.lang.Double="{0}" must be a double.
typeMismatch.java.lang.Float="{0}" must be a float.
typeMismatch.java.lang.Long="{0}" must be a long.
typeMismatch.java.lang.Short="{0}" must be a short.
typeMismatch.java.util.Date="{0}" is not a date.

# field names
name=Name
email=Email
age=Age
```

Error message gets changed as shown below.



Field name can be inserted in the message by specifying `{0}` in the message; This is as per the description in *Messages to be defined in application-messages.properties*.

Default messages should be defined.

Tip: Refer to Javadoc of DefaultMessageCodesResolver for the details of message key rules.

Binding null to blank string field

When the form is sent with input fields left blank in Spring MVC, empty string instead of `null` binds to form object by default.

In this case, the conditions like "blank is allowed but if specified, it must have at least 6 characters" is not satisfied by the use of existing annotations.

To bind `null` instead of empty string to the properties without any input, `org.springframework.beans.propertyeditors.StringTrimmerEditor` can be used as follows.

```
@Controller
@RequestMapping("xxx")
public class XxxController {

    @InitBinder
    public void initBinder(WebDataBinder binder) {
        // bind empty strings as null
        binder.registerCustomEditor(String.class, new StringTrimmerEditor(true));
    }

    // omitted ...
}
```

With the help of this configuration, it can be specified in the controller whether empty strings which are to be considered as `null` or not.

@ControllerAdvice can be set as a configuration common to the entire project when you want to set reply empty string to `null` in the entire project.

Tip: About attribute of @ControllerAdvice annotation added from Spring Framework 4.0

By specifying attribute of `@ControllerAdvice` annotation, it has been improved to allow flexibility in specifying Controller to apply a method implemented in a class wherein `@ControllerAdvice` is assigned. For details about attribute refer to *Attribute of @ControllerAdvice*.

```
@ControllerAdvice
public class XxxControllerAdvice {

    @InitBinder
    public void initBinder(WebDataBinder binder) {
        // bind empty strings as null
```

```
        binder.registerCustomEditor(String.class, new StringTrimmerEditor(true));
    }

    // omitted ...
}
```

When this setting is made, all empty String values set to String fields of form object become `null`.

Therefore it must be noted that, it becomes necessary to specify `@NotNull` in case of mandatory check.

Reading messages which are not converted from Native to Ascii

A method to read Bean Validation message （`ValidationMessage.properties`） is introduced without converting the message from Native to Ascii.

When the Japanese message is to be handled directly without converting from Native to Ascii, it can be easily implemented if it is linked with `MessageSource` of Spring.

If it is defined as below,, message read by `MessageSource` function can be used in `Hibernate Validator`.

- Bean definition

  `*-domain.xml`

```
<!-- (1) -->
<bean id="validator" class="org.springframework.validation.beanvalidation.
↪LocalValidatorFactoryBean">
    <property name="validationMessageSource">
        <!-- (2) -->
        <bean class="org.springframework.context.support.
↪ResourceBundleMessageSource">
            <property name="basenames">
                <list>
                    <value>ValidationMessages</value> <!-- (3) -->
                </list>
            </property>
            <property name="defaultEncoding" value="UTF-8" />
        </bean>
    </property>
</bean>

<!-- (4) -->
<bean class="org.springframework.validation.beanvalidation.
↪MethodValidationPostProcessor">
    <property name="validator" ref="validator" />
</bean>
```

spring-mvc.xml

```
<!-- (5) -->
<mvc:annotation-driven validator="validator">
    <!-- ommited -->
</mvc:annotation-driven>

<!-- (6) -->
<bean class="org.springframework.validation.beanvalidation.
↪MethodValidationPostProcessor">
    <property name="validator" ref="validator" />
</bean>
```

Sr. No.	Description
(1)	Define a Bean for `LocalValidatorFactoryBean`.
(2)	Define `MessageSource`. Here, `ResourceBundleMessageSource` is used.
(3)	Specify a resource bundle to be read in `ApplicationContext`.
(4)	Specify a Bean defined by (1) in `validator` property of `MethodValidationPostProcessor` while using *Method Validation*. When Method Validation is not used, the Bean definition is not required.
(5)	Specify a Bean defined in (1), in `validator`attribute of `<mvc:annotation-driven>`element.
(6)	Same as (4).

Note: By using `MessageSource`function, property file is not necessarily restricted to be placed just under class path. Further, multiple property files can also be specified.

OS Command Injection Countermeasures

OS command injection, a type of security vulnerability and its countermeasures are explained here.

What is OS command injection -

OS command injection is an issue which occurs when malicious commands are supplied to the user input string resulting in illegal manipulation of the computer, when command execution strings are assembled from user input strings in the application.

Tip: For details, refer Description page of OWASP.

In Java, OS command injection can occur when following are used as the commands to be executed when commands are executed using `exec` method of `ProcessBuilder`class and `Runtime` class.

- `/bin/sh` (in case of Unix system) or `cmd.exe` (In case of Windows)
- String input by user

An example wherein OS command injection occurs when `/bin/sh` is used, is shown below.

```
ProcessBuilder pb = new ProcessBuilder("/bin/sh", "-c", script); // (1)
Process p = pb.start();
```

Sr. No.	Description
(1)	For example, when “exec.sh ; cat /etc/passwd” is input in the `script`, semicolon in the string is interpreted as a delimiter by `/bin/sh` and “cat /etc/passwd” is executed. Hence, `/etc/passwd` is likely to be output depending on how the standard output is handled.

Warning: How to use ScriptEngine and ScriptTemplateViewResolver

Another language (`Ruby`or `Python`etc) can be used on JVM , in `ScriptEngine` added from Java SE 6 or `ScriptTemplateViewResolver` added from Spring Framework 4.2.

When code of another language is to be executed using this function, care must be taken while using the function since OS command injection is likely to occur during writing the code.

Countermeasures

Running external processes should be avoided as much as possible in order to prevent occurrence of OS command injection. However, if it is necessary to execute the external process due to certain circumstances, it must be

executed after implementing the countermeasures below.

- Commands which use `/bin/sh` (in case of Unix system) and `cmd.exe` (in case of Windows) should not be executed as far as possible.
- Check whether the characters entered by user are allowed by the application using the whitelist system

The rules to check whether commands and arguments entered by the user are configured by the specified string are shown below.

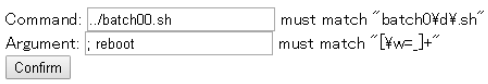
```
@Pattern(regexp = "batch0\\d\\.sh") // (1)
private String cmdStr;

@Pattern(regexp = "[\\w=_]+")  // (2)
private String arg;
```

Sr. No.	Description
(1)	Specify rule to allow only `batch0X.sh`(X is a single byte numeral from 0 to 9).
(2)	Specify rule to allow only the strings configured from innocuous alphanumeric characters (\w)、=、_) as arguments.

Note: In this example, directory traversal is prevented by establishing a rule wherein the path is not included in the commands or arguments.

When `@Pattern` is used, regular expressions specified in `@Pattern` are output as error messages and are considered invalid as messages due to following points.

- Significance of error is not clear, not user-friendly
- logic for the countermeasures to the vulnerability is exposed to the user



A valid message is defined in application-messages.properties to conceal the logic by clarifying the significance of error. For how to define a message, refer *Messages to be defined in application-messages.properties*.

```
Pattern.cmdForm.cmdStr = permit command name: batch00.sh - batch09.sh
Pattern.cmdForm.arg = permit parameter characters and symbols: alphanumeric, =, _
```

Command: ../batch00.sh permit command name: batch00.sh ~ batch09.sh
Argument: ; reboot permit parameter characters and symbols: alphanumeric, =, _
Confirm

4.2 Exception Handling

This chapter describes exception handling mechanism for web applications created using this guideline.

4.2.1 Overview

This section illustrates handling of exceptions occurring within the boundary of Spring MVC. The scope of description is as follows:

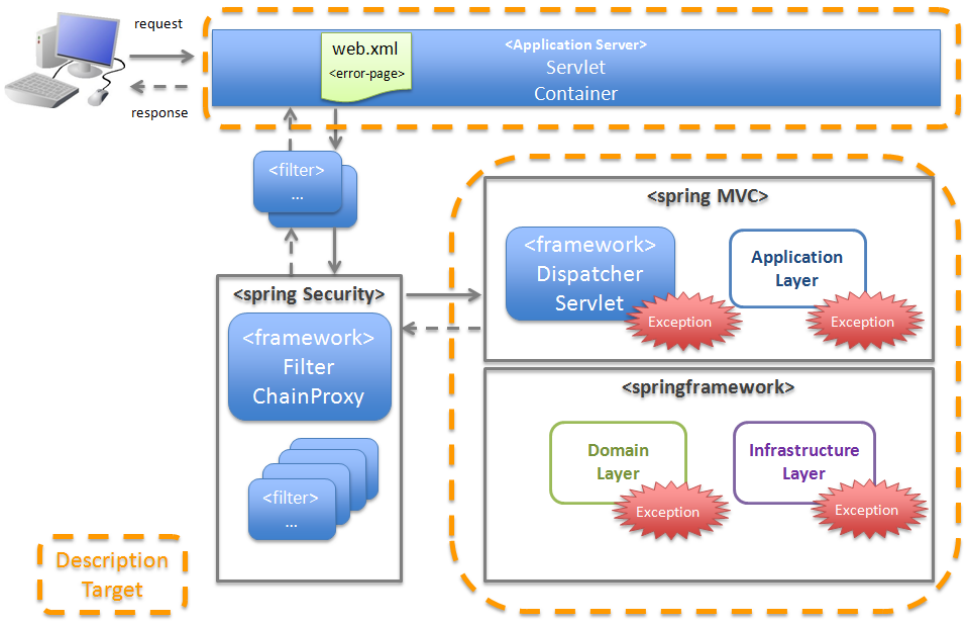


Figure.4.1 **Figure - Description Targets**

1. *Classification of exceptions*
2. *Exception handling methods*

Classification of exceptions

The exceptions that occur when an application is running, can be broadly classified into following 3 categories.

Table.4.6 **Table - Types of exceptions that occur when an application is running**

Sr. No.	Classification	Description	*Types of exceptions*
(1)	Exceptions wherein cause can be resolved if the user re-executes the operation (by changing input values etc.)	The exceptions wherein cause can be resolved if the user re-executes the operation, are handled in application code.	1. *Business exception* 2. *Library exceptions that occurs during normal operation*
(2)	Exceptions wherein cause cannot be resolved even if the user re-executes the operation	The exceptions wherein cause cannot be resolved even if the user re-executes the operation, are handled using the framework.	1. *System Exception* 2. *Unexpected System Exception* 3. *Fatal Errors*
(3)	Exceptions due to invalid requests from the client	The exceptions which occur due to invalid requests from the client, are handled using the framework.	1. *Framework exception in case of invalid requests*

Note: Who is a target audience for exceptions?

- Application Developer should be aware of exception (1).
- Application Architect should be aware of exceptions (2) and (3).

Exception handling methods

The exceptions which occur when the application is running, are handled using following 4 methods.

For details on flow of each handling method, refer to *Basic Flow of Exception Handling*.

Table.4.7: **Table - Exception Handling Methods**

Sr. No.	Handling Method	Description	*Exception Handling Patterns*
(1)	Use `try-catch` in the application code to carry out exception handling.	Use this in order to handle exceptions at request (Controller method) level. For details, refer to *Basic flow when the Controller class handles the exception at request level*.	1. *When notifying partial redo of a use case (from middle)*
(2)	Use `@ExceptionHandler` annotation to carry out exception handling in application code.	Use this when exceptions are to be handled at use case (Controller) level. For details, refer to *Basic flow when the Controller class handles the exception at use case level*.	1. *When notifying redo of a use case (from beginning)*
(3)	Use HandlerExceptionResolver mechanism provided by the framework to carry out exception handling.	Use this in order to handle exceptions at servlet level. When `<mvc:annotation-driven>` is specified, HandlerExceptionResolver uses *automatically registered class*, and `SystemExceptionResolver` provided by common library. For details, refer to *Basic flow when the framework handles the exception at servlet level*.	1. *When notifying that the system or application is not in a normal state* 2. *When notifying that the request contents are invalid*

次のページに続く

Table. 4.7 – 前のページからの続き

Sr. No.	Handling Method	Description	*Exception Handling Patterns*
(4)	Use the error-page function of the servlet container to carry out exception handling.	Use this in order to handle fatal errors and exceptions which are out of the boundary of Spring MVC. For details, refer to *Basic flow when the servlet container handles the exception at web application level*.	1. *When a fatal error has been detected* 2. *When notifying that an exception has occurred in the presentation layer (JSP etc.)*

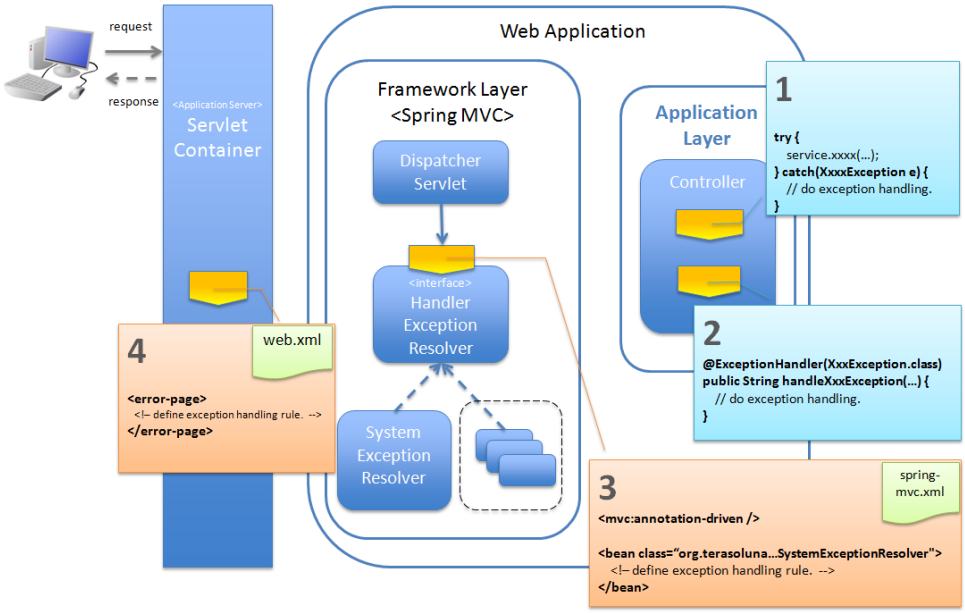


Figure.4.2 **Figure - Exception Handling Methods**

Note: Who will carry out exception handling?

- Application Developer should design and implement (1) and (2).
- Application Architect should design and configure (3) and (4).

Note: About automatically registered HandlerExceptionResolver

When <mvc:annotation-driven> is specified, the roles of HandlerExceptionResolver, which is registered automatically, are as follows:

The priority order will be as given below.

Sr. No.	Class (Priority order)	Role
(1)	ExceptionHandlerExceptionResolver (order=0)	This is a class for handling the exceptions by calling the methods of Controller class with `@ExceptionHandler` annotation. This class is necessary for implementing the handling method No. 2.
(2)	ResponseStatusExceptionResolver (order=1)	This is a class for handling the exceptions wherein `@ResponseStatus` is applied as class level annotation. `HttpServletResponse#sendError(int sc,String msg)` is called with the value specified in `@ResponseStatus`.
(3)	DefaultHandlerExceptionResolver (order=2)	This is a class for handling framework exceptions in Spring MVC. `HttpServletResponse#sendError(int sc)` is called using the value of HTTP response code corresponding to framework exception. For details on HTTP response code to be set, refer to *HTTP response code set by DefaultHandlerExceptionResolver*.

Note: What is a role of SystemExceptionResolver provided by common library?

This is a class for handling exceptions which are not handled by HandlerExceptionResolver which is registered automatically when <mvc:annotation-driven> is specified. Therefore, the order of priority of this class should be set after DefaultHandlerExceptionResolver.

Note: About @ControllerAdvice annotation added from Spring Framework 3.2

`@ControllerAdvice` made exception handling possible using `@ExceptionHandler` at servlet level. If methods with `@ExceptionHandler` annotation are defined in a class with `@ControllerAdvice` annotation, then exception handling carried out in the methods with `@ExceptionHandler` annotation can be applied to all the Controllers in Servlet. When implementing

the above in earlier versions, it was necessary to define methods with `@ExceptionHandler` annotation in Controller base class and inherit each Controller from the base class.

About attribute of @ControllerAdvice annotation added from Spring Framework 4.0

By specifying attribute of `@ControllerAdvice` annotation, it has been improved in such a way that Controller that applies a method implemented in class with `@ControllerAdvice`, can be specified flexibly. For details on attribute, refer to *attribute of @ControllerAdvice*.

Note: Where to use @ControllerAdvice annotation?

1. When logic other than determining the View name and HTTP response code is necessary for exception handling to be carried out at servlet level, (If only determining View name and HTTP response code is sufficient, it can be implemented using `SystemExceptionResolver`)
2. In case of exception handling carried out at servlet level, when response data is to be created by serializing the error model (JavaBeans) in JSON or XML formats without using template engines such as JSP. (Used for error handling at the time of creating Controller for AJAX or REST).

4.2.2 Detail

1. *Types of exceptions*
2. *Exception Handling Patterns*
3. *Basic Flow of Exception Handling*

Types of exceptions

There are 6 types of exceptions that occur in a running application.

Table.4.8 **Table - Types of Exceptions**

Sr. No.	Types of Exceptions	Description
(1)	*Business exception*	Exception to notify a violation of a business rule
(2)	*Library exceptions that occurs during normal operation*	Amongst the exceptions that occur in framework and libraries, the exception which is likely to occur when the system is operating normally.
(3)	*System Exception*	Exception to notify detection of a state which should not occur when the system is operating normally
(4)	*Unexpected System Exception*	Unchecked exception that does not occur when the system is operating normally
(5)	*Fatal Errors*	Error to notify an occurrence of a fatal error impacting the entire system (application)
(6)	*Framework exception in case of invalid requests*	Exception to notify that the framework has detected invalid request contents

Business exception

Exception to notify a violation of a business rule

This exception is generated in domain layer.

The situations in the system such as below are pre-defined and hence need not be dealt by the system administrator.

- If the reservation date is past the deadline when making travel reservations
- If a product is out of stock when it is ordered
- etc ...

Note: Corresponding Exception Class

- `org.terasoluna.gfw.common.exception.BusinessException` (Class provided by common library).
- When handling is to be carried out at detailed level, exception class that inherits BusinessException should be created.
- If the requirements cannot be met by the exception class provided by common library, a business exception class should be created for each project.

Library exceptions that occurs during normal operation

Amongst the exceptions that occur in the framework and libraries, **the exception which is likely to occur when the system is operating normally.**

The exceptions that occur in the framework and libraries cover the exception classes in Spring Framework or other libraries.

Situations such are below are pre-defined and hence need not be dealt by the system administrator.

- Optimistic locking exception and pessimistic locking exception that occur if multiple users attempt to update same data simultaneously.
- Unique constraint violation exception that occurs if multiple users attempt to register same data simultaneously.
- etc ...

Note: Examples of corresponding Exception Classes

- `org.springframework.dao.OptimisticLockingFailureException` (Exception that occurs if there is an error due to optimistic locking).
- `org.springframework.dao.PessimisticLockingFailureException` (Exception that occurs if there is an error due to pessimistic locking).
- `org.springframework.dao.DuplicateKeyException` (Exception that occurs if there is a unique constraint violation).
- etc ...

Todo

Currently it has been found that unexpected errors occur if JPA(Hibernate) is used.

- In case of unique constraint violation, `org.springframework.dao.DataIntegrityViolationException` occurs and not `DuplicateKeyException`.

System Exception

Exception to notify that a state which should not occur when the system is operating normally has been detected.

This exception should be generated in application layer and domain layer.

The exception needs to be dealt by the system administrator.

- If the master data, directories, files, etc. those should have pre-existed, do not exist
- Amongst the checked exceptions that occur in the framework, libraries, if exceptions classified as system errors are caught (IOException during file operations etc.).
- etc ...

Note: Corresponding Exception Class

- `org.terasoluna.gfw.common.exception.SystemException` (Class provided by common library).
- When the error screen of View and HTTP response code need to be switched on the basis of the cause of each error, SystemException should be inherited for each error cause and the mapping between the inherited exception class and the error screen should be defined in the bean definition of SystemExceptionResolver.
- If the requirements are not met by the system exception class provided in the common library, a system exception class should be created in each respective project.

Unexpected System Exception

Unchecked exception that does not occur when the system is operating normally.

Action by system administrator or analysis by system developer is necessary.

Unexpected system exceptions should not be handled (try-catch) in the application code.

- When bugs are hidden in the application, framework or libraries
- When DB Server etc. is down
- etc ...

Note: Example of Corresponding Exception Class

- `java.lang.NullPointerException` (Exception caused due to bugs).
- `org.springframework.dao.DataAccessResourceFailureException` (Exception that occurs when DB server is down).
- etc ...

Fatal Errors

Error to notify that a fatal problem impacting the entire system (application), has occurred.

Action/recovery by system administrator or system developer is necessary.

Fatal Errors (error that inherits java.lang.Error) must not be handled (try-catch) in the application code.

- If the memory available in Java virtual machine is inadequate
- etc ...

Note: Example of corresponding Error Class

- `java.lang.OutOfMemoryError` (Error due to inadequate memory).
- etc ...

Framework exception in case of invalid requests

Exception to notify that the framework has detected invalid request contents.

This exception occurs in the framework (Spring MVC).

The cause lies at client side; hence it need not be dealt by the system administrator.

- Exception that occurs when a request path for which only POST method is permitted, is accessed using GET method.
- Exception that occurs when type-conversion fails for the values extracted from URI using @PathVariable annotation.
- etc ...

Note: Example of corresponding Exception Class

- `org.springframework.web.HttpRequestMethodNotSupportedException` (Exception that occurs when the access is made through a HTTP method which is not supported).
- `org.springframework.beans.TypeMismatchException` (Exception that occurs if the specified value cannot be converted to URI).
- etc ...

Class for which HTTP status code is "4XX" in the list given at *HTTP response code set by DefaultHandlerExceptionResolver*.

Exception Handling Patterns

There are 6 types of exception handling patterns based on the purpose of handling.

(1)-(2) should be handled at use case level and (3)-(6) should be handled at the entire system (application) level.

Table.4.9: **Table - Exception Handling Patterns**

Sr. No.	Purpose of handling	Types of exceptions	Handling method	Handling location
(1)	*When notifying partial redo of a use case (from middle)*	1. *Business exception*	Application code (try-catch)	Request
(2)	*When notifying redo of a use case (from beginning)*	1. *Business exception* 2. *Library exceptions that occurs during normal operation*	Application code (@ExceptionHandler)	Use case
(3)	*When notifying that the system or application is not in a normal state*	1. *System Exception* 2. *Unexpected System Exception*	Framework (Handling rules are specified in `spring-mvc.xml`)	Servlet
(4)	*When notifying that the request contents are invalid*	1. *Framework exception in case of invalid requests*	Framework	Servlet
(5)	*When a fatal error has been detected*	1. *Fatal Errors*	Servlet container (Handling rules are specified in `web.xml`)	Web application

次のページに続く

Table. 4.9 – 前のページからの続き

Sr. No.	Purpose of handling	Types of exceptions	Handling method	Handling location
(6)	*When notifying that an exception has occurred in the presentation layer (JSP etc.)*	1. All the exceptions and errors that occur in the presentation layer	Servlet container (Handling rules are specified in `web.xml`)	Web application

When notifying partial redo of a use case (from middle)

When partial redo (from middle) of a use case is to be notified, catch (try-catch) the exception in the application code of Controller class and carry out exception handling at request level.

Note: Example of notifying partial redo of a use case

- When an order is placed through a shopping site, if business exception notifying stock shortage occurs.

  In such a case, order can be placed if the no. of items to be ordered is reduced; hence display a screen on which no. of items can be changed and prompt a message asking to change the same.

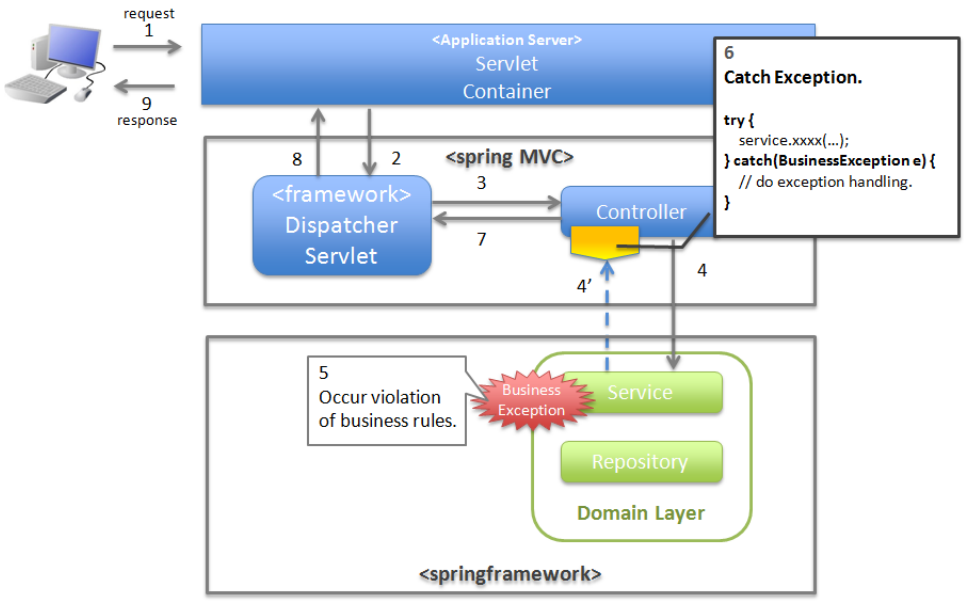


Figure.4.3 **Figure - Handling method when notifying partial redo of a use case (from middle)**

When notifying redo of a use case (from beginning)

When redo of a use case (from beginning) is to be notified, catch the exception using `@ExceptionHandler`, and carry out exception handling at use case level.

Note: Example when notifying redo of a use case (from beginning)

- At the time of changing the product master on shopping site (site for administrator), it has been changed by other operator (in case of optimistic locking exception).

  In such a case, the operation needs to be carried out after verifying the changes made by the other user; hence display the front screen of the use case (for example: search screen of product master) and prompt a message asking the user to perform the operation again.

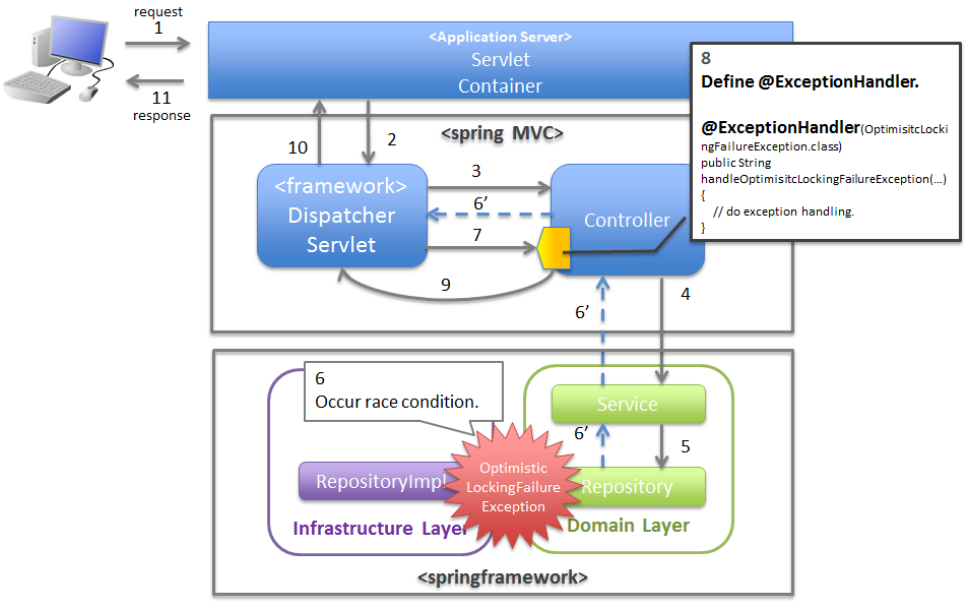


Figure.4.4 **Figure - Handling method when notifying redo of a use case (from beginning)**

When notifying that the system or application is not in a normal state

When an exception to notify that system or application is not in a normal state is detected, catch the exception using SystemExceptionResolver and carry out exception handling at servlet level.

Note: Examples for notifying that the system or application is not in a normal state

- In case of a use case for connecting to an external system, if an exception occurs notifying that the external system is blocked.

  In such a case, since use case cannot be executed until external system resumes service, display the error screen, and notify that the use case cannot be executed till the external system resumes service.

- When searching master information with the value specified in the application, if the corresponding master information does not exist.

  In such a case, there is a possibility of bug in master maintenance function or of error (release error) in data input by the system administrator; hence display the system error screen and notify that a system error has occurred.

- When IOException occurs from the API during file operations.

  This could be a case of disk failure etc.; hence display the system error screen and notify that system error has occurred.

- etc ...

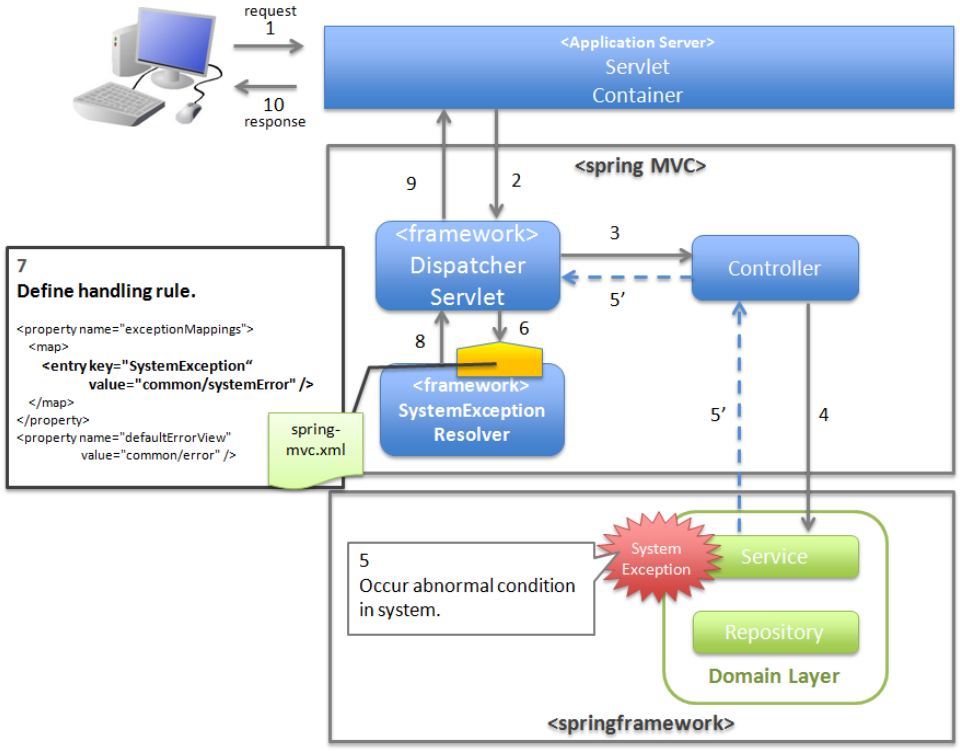


Figure.4.5 **Figure - Handling method when an exception to notify that system or application is not in a normal state is detected**

When notifying that the request contents are invalid

When notifying that an invalid request is detected by the framework, catch the exception using DefaultHandlerExceptionResolver, and carry out exception handling at servlet level.

Note: Example when notifying that the request contents are invalid

- When a URI is accessed using GET method, while only POST method is permitted.

  In such a case, it is likely that the access has been made directly using the Favorites feature etc. of the browser; hence display the error screen and notify that the request contents are invalid.

- When value cannot be extracted from URI using `@PathVariable` annotation

  In such a case, it is likely that the access has been made directly by replacing the value of the address bar on the browser; hence display the error screen and notify that the request contents are invalid.

- etc ...

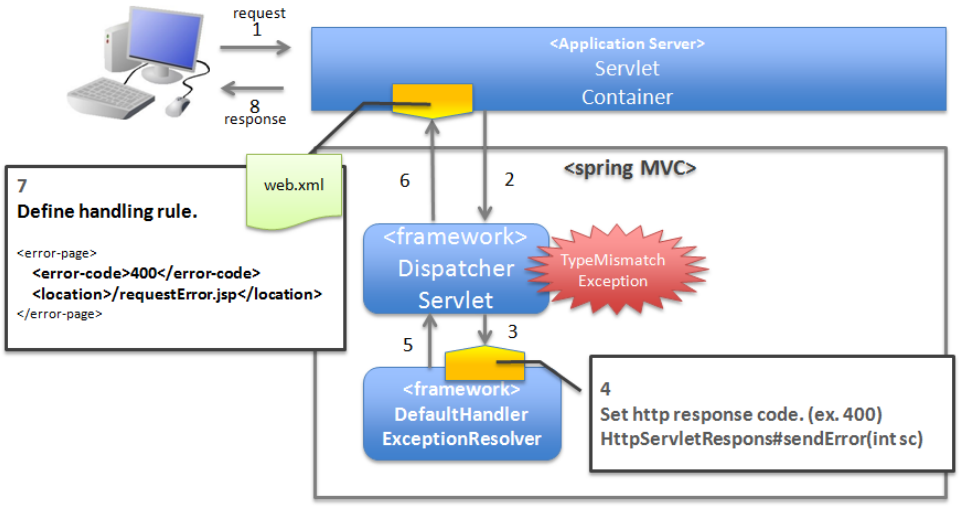


Figure.4.6 **Figure - Handling method when notifying that the request contents are invalid**

When a fatal error has been detected

When a fatal error has been detected, catch the exception using servlet container and carry out exception handling at web application level.

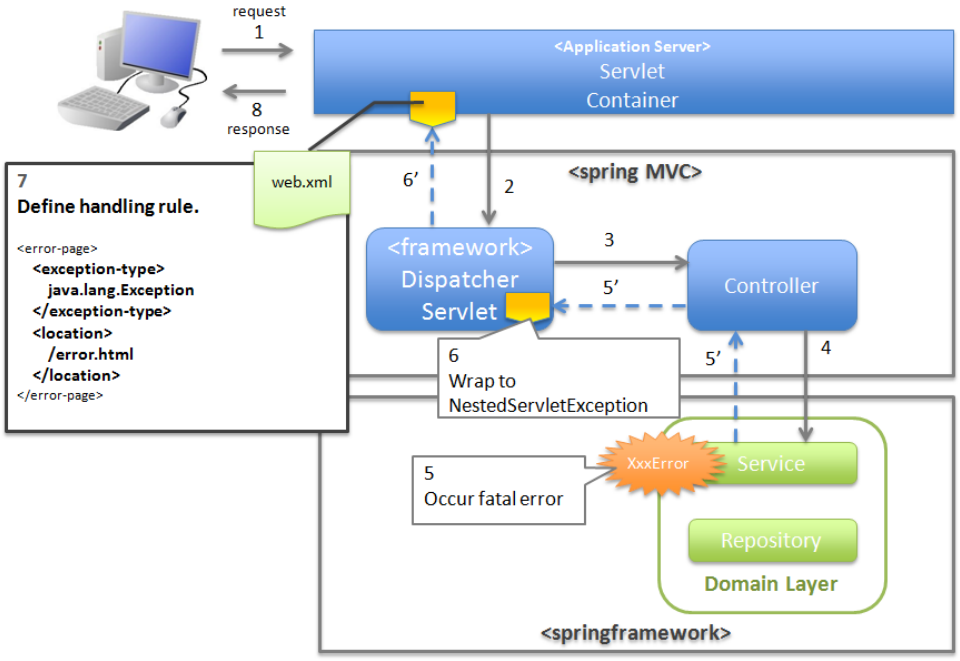


Figure.4.7 **Figure - Handling method when a fatal error has been detected**

Warning: Regarding handling fatal errors using @ExceptionHandler and SystemExceptionResolver

Fatal error (`java.lang.Error` and its subclass) and the `org.springframework.web.util.NestedServletExcep` wrapped by `java.lang.Throwable` can be captured from Spring Framework 4.3 by using exception handler (`HandlerExceptionResolver`) of Spring MVC. As a result of this change, the fatal error and `Throwable` are likely to get accidentally captured by `SystemExceptionResolver` offered by common library (inherit `HandlerExceptionResolver`) and the method which assigns `@ExceptionHandler`(run on `HandlerExceptionResolver`system).

In order to capture the fatal errors by servlet containers, it is necessary to notify servlet container without using the methods which assign `SystemExceptionResolver` and `@ExceptionHandler` for handling `NestedServletException`. "How to use" explains the method for not handling `NestedServletException`.

- For `SystemExceptionResolver`, refer to *Application Layer Settings*.
- For `@ExceptionHandler`, refer to *Method to handle exception at use case level*.

When notifying that an exception has occurred in the presentation layer (JSP etc.)

When notifying that an exception has occurred in the presentation layer (JSP etc.), catch the exception using servlet container and carry out exception handling at web application level.

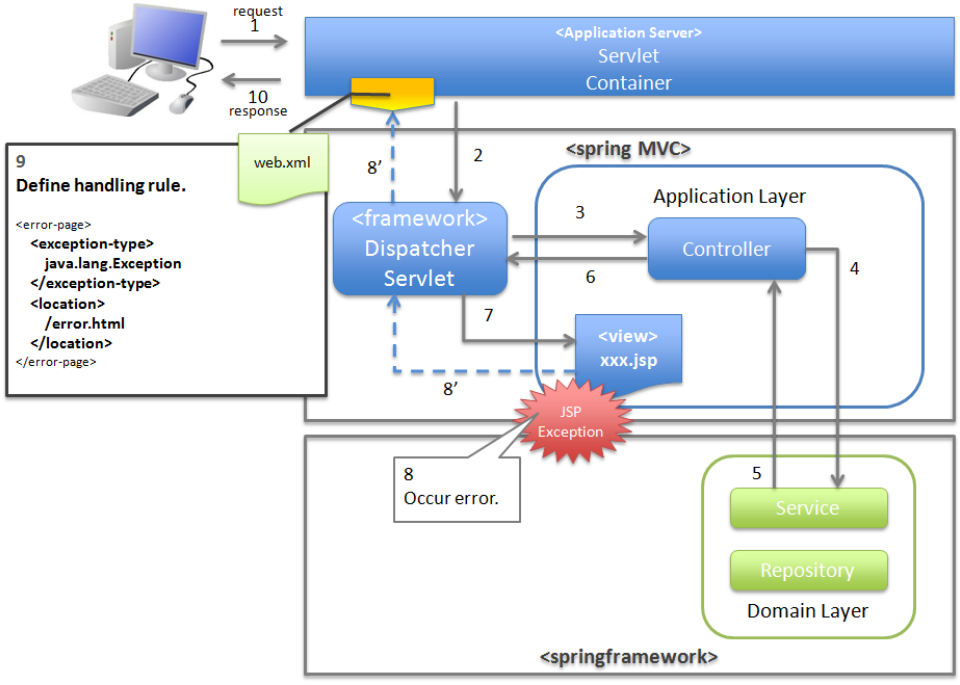


Figure.4.8 **Figure - Handling method when notifying an occurrence of exception in the presentation layer (JSP etc.)**

Basic Flow of Exception Handling

The basic flow of exception handling is shown below.

For an overview of classes provided by common library, refer to *Exception handling classes provided by the common library*.

The processing to be implemented in the application code is indicated in Bold.

The log of stack trace and exception messages is output by the classes (Filter and Interceptor class) provided by the common library.

When any information other than exception messages and stack trace needs to be logged, it should be implemented separately in each of the logic.

This section describes the flow of exception handling; hence the description related to flow till the calling of Service class is omitted.

1. *Basic flow when the Controller class handles the exception at request level*
2. *Basic flow when the Controller class handles the exception at use case level*
3. *Basic flow when the framework handles the exception at servlet level*
4. *Basic flow when the servlet container handles the exception at web application level*

Basic flow when the Controller class handles the exception at request level

In order to handle the exception at request level, catch (try-catch) the exception in the application code of the Controller class.

Refer to the figure below:

It illustrates the basic flow at the time of handling a business exception (`org.terasoluna.gfw.common.exception.BusinessException`) provided by common library.

Log is output using interceptor (`org.terasoluna.gfw.common.exception.ResultMessagesLoggingInterceptor`) which records that an exception holding the result message has occurred.

4. In Service class, BusinessException is generated and thrown.
5. ResultMessagesLoggingInterceptor calls ExceptionLogger, and outputs log of warn level (monitoring log and application log). ResultMessagesLoggingInterceptor class outputs logs only when sub exception (BusinessException/ResourceNotFoundException) of ResultMessagesNotificationException occurs.
6. **Controller class catches BusinessException, extracts the message information (ResultMessage) from BusinessException and sets it to Model for screen display(6').**
7. **Controller class returns the View name.**

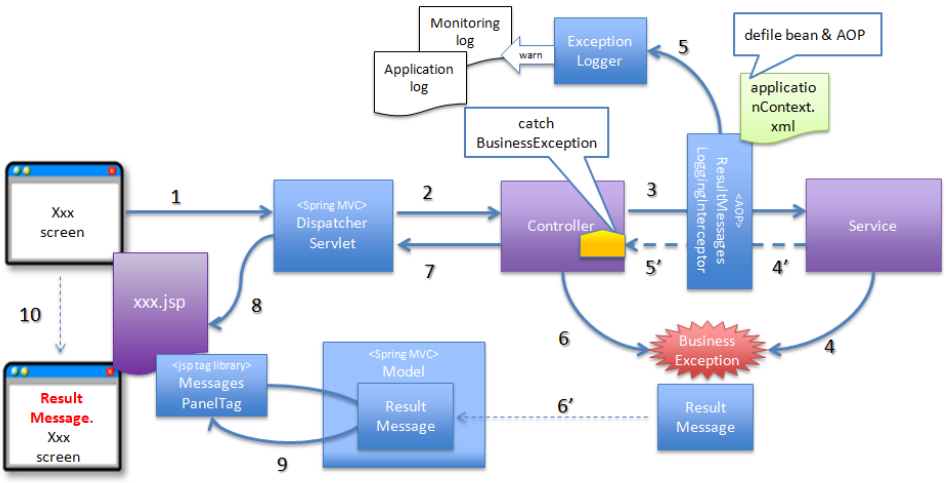


Figure.4.9 **Figure - Basic flow when the Controller class handles the exception at request level**

8. DispatcherServlet calls JSP corresponding to the returned View name.
9. **JSP acquires message information (ResultMessage) using MessagesPanelTag and generates HTML code for message display.**
10. The response generated by JSP is displayed.

Basic flow when the Controller class handles the exception at use case level

In order to handle the exception at use case level, catch the exception using `@ExceptionHandler` of Controller class.

Refer to the figure below.

It illustrates the basic flow at the time of handling an arbitrary exception (`XxxException`).

Log is output using interceptor (`org.terasoluna.gfw.web.exception.HandlerExceptionResolverLoggingInterceptor`). This interceptor records that the exception is handled using HandlerExceptionResolver.

3. Exception (XxxException) is generated in the Service class called from Controller class.
4. DispatcherServlet catches XxxException and calls ExceptionHandlerExceptionResolver.
5. ExceptionHandlerExceptionResolver calls exception handling method provided in Controller class.
6. **Controller class generates message information (ResultMessage) and sets it to the Model for screen display.**
7. **Controller class returns the View name.**
8. ExceptionHandlerExceptionResolver returns the View name returned by the Controller.

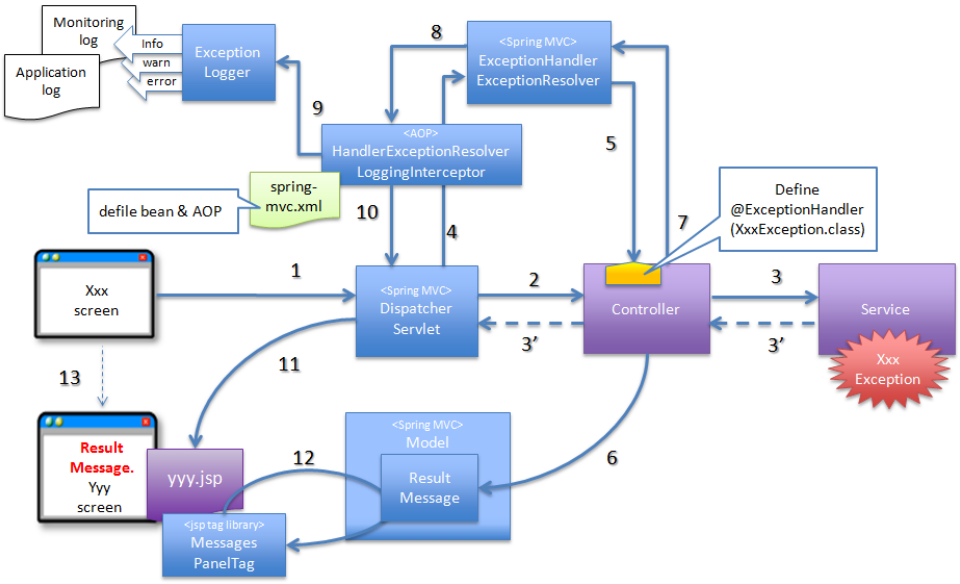


Figure.4.10 **Figure - Basic flow when the Controller class handles the exception at use case level**

9. HandlerExceptionResolverLoggingInterceptor calls ExceptionLogger and outputs logs (monitoring log and application log) (at info, warn, error levels) corresponding to the exception code.
10. HandlerExceptionResolverLoggingInterceptor returns View name returned by ExceptionHandlerExceptionResolver.
11. DispatcherServlet calls JSP that corresponds to the returned View name.
12. **JSP acquires the message information (ResultMessage) using MessagesPanelTag and generates HTML code for message display.**
13. The response generated by JSP is displayed.

Basic flow when the framework handles the exception at servlet level

In order to handle the exception using the framework (at servlet level), catch the exception using SystemExceptionResolver.

Refer to the figure below.

It illustrates the basic flow at the time of handling the system exception (`org.terasoluna.gfw.common.exception.SystemException`) provided by common library using `org.terasoluna.gfw.web.exception.SystemExceptionResolver`.

Log is output using interceptor (`org.terasoluna.gfw.web.exception.HandlerExceptionResolverLoggingInterceptor`) which records the exception specified in the argument of exception handling method.

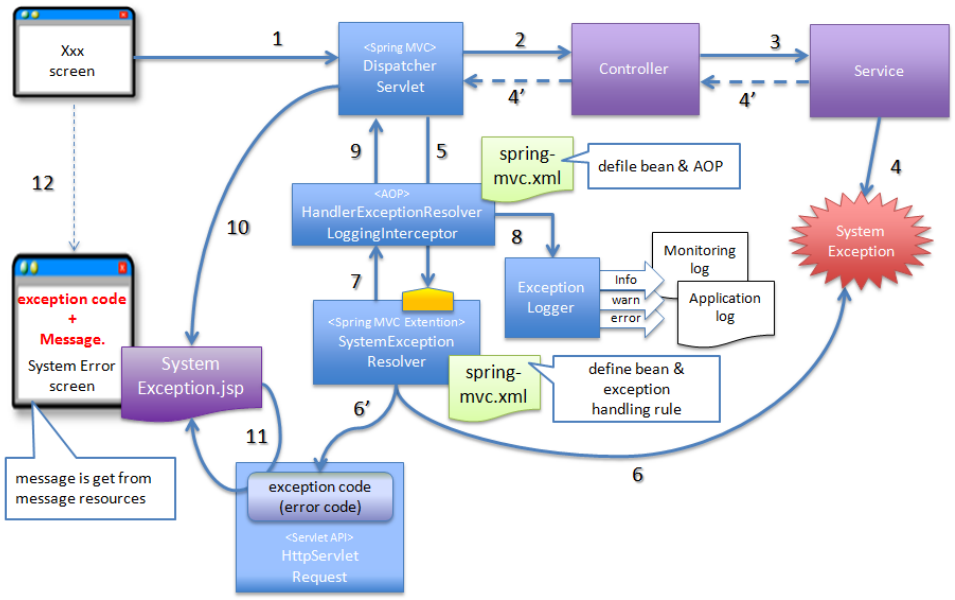


Figure.4.11 **Figure - Basic flow when the framework handles the exception at servlet level**

4. A state equivalent to the system exception is detected in Service class; hence throw a SystemException.
5. DispatcherServlet catches SystemException, and calls SystemExceptionResolver.
6. SystemExceptionResolver acquires exception code from SystemException and sets it to HttpServletRequest for screen display (6').
7. SystemExceptionResolver returns the View name corresponding to SystemException.
8. HandlerExceptionResolverLoggingInterceptor calls ExceptionLogger and outputs logs (monitoring log and application log) (at info, warn, error levels) corresponding to the exception code.
9. HandlerExceptionResolverLoggingInterceptor returns the View name returned by SystemExceptionResolver.
10. DispatcherServlet calls the JSP corresponding to the returned View name.
11. **JSP acquires the exception code from HttpServletRequest and inserts it in HTML code for message display.**
12. The response generated by JSP is displayed.

Basic flow when the servlet container handles the exception at web application level

In order to handle exceptions at web application level, catch the exception using servlet container.

Fatal errors, exceptions which are not handled using the framework (exceptions in JSP etc.) and exceptions occurred in Filter are to be handled using this flow.

Refer to the figure below.

It illustrates the basic flow at the time of handling java.lang.Exception by “error page”.

Log is output using servlet filter (`org.terasoluna.gfw.web.exception.ExceptionLoggingFilter`) which records that an unhandled exception has occurred.

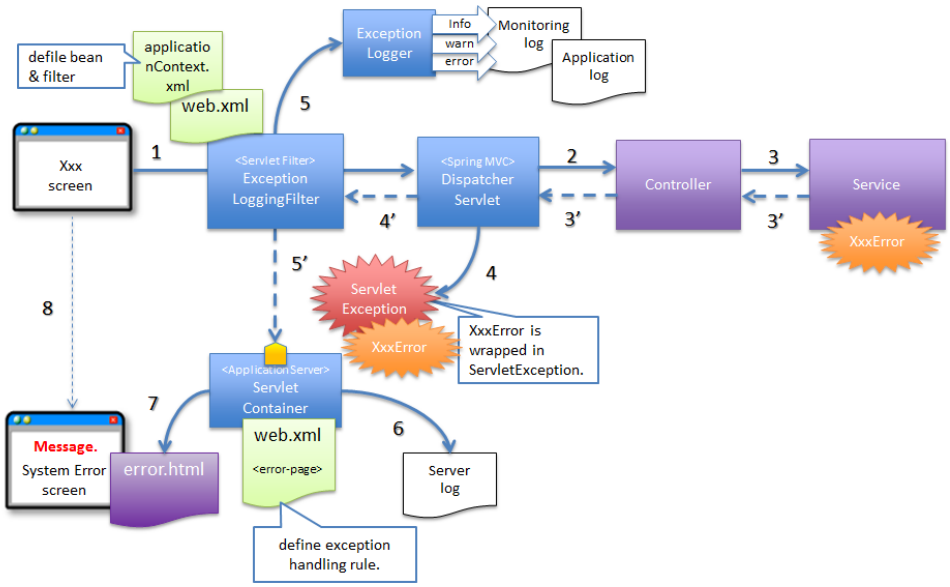


Figure.4.12 **Figure - Basic flow when servlet container handles the exception at web application level**

4. DispatcherServlet catches XxxError, wraps it in ServletException and then throws it.
5. ExceptionLoggingFilter catches ServletException and calls ExceptionLogger. ExceptionLogger outputs logs (monitoring log and application log) (at info, warn, error levels) corresponding to the exception code. ExceptionLoggingFilter re-throws ServletException.
6. ServletContainer catches ServletException and outputs logs to server log. Log level varies depending on the application server.
7. ServletContainer calls the View (HTML etc.) defined in `web.xml`.
8. The response generated by the View which is called by the ServletContainer, is displayed.

4.2.3 How to use

The usage of exception handling functionality is described below.

For exception handling classes provided by common library, refer to *Exception handling classes provided by the common library*.

1. *Application Settings*
2. *Coding Points (Service)*
3. *Coding Points (Controller)*
4. *Coding points (JSP)*

Application Settings

The application settings required for using exception handling are shown below.

Further, these settings are already done in blank project. Hence, it will work only by doing changes given in **[Location to be customized for each project]**section.

1. *Common Settings*
2. *Domain Layer Settings*
3. *Application Layer Settings*
4. *Servlet Container Settings*

Common Settings

1. Add bean definition of logger class (`ExceptionLogger`) which will output exception log.

- **applicationContext.xml**

```
<!-- Exception Code Resolver. -->
<bean id="exceptionCodeResolver"
    class="org.terasoluna.gfw.common.exception.
↪SimpleMappingExceptionCodeResolver"> <!-- (1) -->
    <!-- Setting and Customization by project. -->
    <property name="exceptionMappings"> <!-- (2) -->
        <map>
            <entry key="ResourceNotFoundException" value="e.xx.fw.5001" />
            <entry key="BusinessException" value="e.xx.fw.8001" />
        </map>
    </property>
    <property name="defaultExceptionCode" value="e.xx.fw.9001" /> <!-- (3) -->
</bean>

<!-- Exception Logger. -->
<bean id="exceptionLogger"
    class="org.terasoluna.gfw.common.exception.ExceptionLogger"> <!-- (4) -->
```

```
    <property name="exceptionCodeResolver" ref="exceptionCodeResolver" /> <!--
↪ (5) -->
</bean>
```

Sr. No.	Description
(1)	Add `ExceptionCodeResolver` to bean definition.
(2)	Specify mapping between name of the exception to be handled and applicable “Exception Code (Message ID)”. In the above example, if the “BusinessException” is included in the class name of exception class (or parent class), “w.xx.fw.8001” will be the “Exception code (Message ID)” and if “ResourceNotFoundException” is included in the class name of exception class (or parent class), “w.xx.fw.5001” will be the “Exception code (Message ID)”. --- **Note: About the Exception Code (Message ID)** The exception code is defined here for taking into account the case wherein message ID is not specified in generated “BusinessException”, however, it is recommended that you specify the “Exception Code (Message ID)” at the implementation side which generates the “BusinessException” (this point is explained later). Specification of “Exception Code (Message ID)” for “BusinessException” is an alternative measure in case it is not specified when “BusinessException” occurs. --- **[Location to be customized for each project]**
(3)	Specify default “Exception Code (Message ID)”. In the above example, if “BusinessException” or “ResourceNotFoundException” is not included in the class names of exception class (or parent class), ” e.xx.fw.9001” will be “Exception code (Message ID)”. **[Location to be customized for each project]** --- **Note: Exception Code (Message ID)** Exception code is only to be output to log (It can also be fetched on screen). It is also possible to create an ID which need not be in the format usually used to define IDs in Properties file, but can be identified in the application. For example, MA7001 etc. ---

次のページに続く

Table. 4.10 – 前のページからの続き

Sr. No.	Description
(4)	Add `ExceptionLogger` to bean definition.
(5)	Inject `ExceptionCodeResolver` into the bean definition of `ExceptionLogger`.

2. Add log definition.

- **logback.xml**

  Add log definition for monitoring log.

```
<appender name="MONITORING_LOG_FILE" class="ch.qos.logback.core.rolling.
↪RollingFileAppender"> <!-- (1) -->
    <file>${app.log.dir:-log}/projectName-monitoring.log</file>
    <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
        <fileNamePattern>${app.log.dir:-log}/projectName-monitoring-%d
↪{yyyyMMdd}.log</fileNamePattern>
        <maxHistory>7</maxHistory>
    </rollingPolicy>
    <encoder>
        <charset>UTF-8</charset>
        <pattern><![CDATA[date:%d{yyyy-MM-dd HH:mm:ss}\tX-Track:%X{X-Track}
↪\tlevel:%-5level\tmessage:%msg%n]]></pattern>
    </encoder>
</appender>

<logger name="org.terasoluna.gfw.common.exception.ExceptionLogger.Monitoring"␣
↪additivity="false"> <!-- (2) -->
    <level value="error" /> <!-- (3) -->
    <appender-ref ref="MONITORING_LOG_FILE" /> <!-- (4) -->
</logger>
```

Sr. No.	Description
(1)	Specify appender definition used for outputting monitoring log. In the above example, an appender to be output to a file has been specified, however the appender used should be consider as per the system requirements. **[Location to be customized for each project]**
(2)	Specify logger definition for monitoring log. When creating ExceptionLogger, if any logger name is not specified, the above settings can be used as is. **Warning: About additivity setting value** Specify `false`. If `true` is specified, the same log will be output by upper level logger (for example, root).
(3)	Specify output level. In ExceptionLogger, 3 types of logs of info, warn, error are output; however, the level specified should be as per the system requirements. The guideline recommends Error level. **[Location to be customized for each project]**
(4)	Specify the appender which will act as output destination. **[Location to be customized for each project]**

Add log definition for application log.

```
<appender name="APPLICATION_LOG_FILE" class="ch.qos.logback.core.rolling.
↪RollingFileAppender"> <!-- (1) -->
    <file>${app.log.dir:-log}/projectName-application.log</file>
    <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
        <fileNamePattern>${app.log.dir:-log}/projectName-application-%d
↪{yyyyMMdd}.log</fileNamePattern>
        <maxHistory>7</maxHistory>
    </rollingPolicy>
    <encoder>
        <charset>UTF-8</charset>
        <pattern><![CDATA[date:%d{yyyy-MM-dd HH:mm:ss}\tthread:%thread\tX-
↪Track:%X{X-Track}\tlevel:%-5level\tlogger:%-48logger{48}\tmessage:%msg%n]]>
↪</pattern>
```

```
    </encoder>
</appender>

<logger name="org.terasoluna.gfw.common.exception.ExceptionLogger"> <!-- (2) -
↪->
    <level value="info" /> <!-- (3) -->
</logger>

<root level="warn">
    <appender-ref ref="STDOUT" />
    <appender-ref ref="APPLICATION_LOG_FILE" /> <!-- (4) -->
</root>
```

Sr. No.	Description
(1)	Specify appender definition used for outputting application log. In the above example, an appender to be output to a file has been specified, however the appender used should be consider as per the system requirements. **[Location to be customized for each project]**
(2)	Specify logger definition for application log. When creating ExceptionLogger, if any logger name is not specified, the above settings can be used as is. **Note: Appender definition for outputting application log** Rather than defining a separate appender for logging exceptions, it is recommended to use the same appender which is used for logging in application code or framework. By using same output destination, it becomes easier to track the process until an exception occurs.
(3)	Specify the output level. In ExceptionLogger, 3 types of logs of info, warn, error are output; however, the level specified should be as per the system requirements. This guideline recommends info level. **[Location to be customized for each project]**
(4)	Log is transmitted to root since appender is not specified for the logger set in (2). Therefore, specify an appender which will act as output destination. Here, it will be output to "STDOUT" and "APPLICATION_LOG_FILE". **[Location to be customized for each project]**

Domain Layer Settings

When exceptions (BusinessException,ResourceNotFoundException) holding ResultMessages occur, add AOP settings and bean definition of interceptor class (`ResultMessagesLoggingInterceptor`) for log output.

- **xxx-domain.xml**

```
<!-- interceptor bean. -->
<bean id="resultMessagesLoggingInterceptor"
```

```
      class="org.terasoluna.gfw.common.exception.
↪ResultMessagesLoggingInterceptor"> <!-- (1) -->
      <property name="exceptionLogger" ref="exceptionLogger" /> <!-- (2) -->
</bean>

<!-- setting AOP. -->
<aop:config>
    <aop:advisor advice-ref="resultMessagesLoggingInterceptor"
                 pointcut="@within(org.springframework.stereotype.Service)" />
↪ <!-- (3) -->
</aop:config>
```

Sr. No.	Description
(1)	Add bean definition of `ResultMessagesLoggingInterceptor`.
(2)	Inject the logger which outputs exception log. Specify "exceptionLogger" defined in `applicationContext.xml`.
(3)	Apply ResultMessagesLoggingInterceptor for the method of the Service class (with `@Service` annotation).

Application Layer Settings

Add to bean definition, the class (`SystemExceptionResolver`) used for handling the exceptions which are not handled by HandlerExceptionResolver registered automatically when <mvc:annotation-driven> is specified. .

- **spring-mvc.xml**

```
<!-- Setting Exception Handling. -->
<!-- Exception Resolver. -->
<bean class="org.terasoluna.gfw.web.exception.SystemExceptionResolver"> <!--␣
↪(1) -->
    <property name="exceptionCodeResolver" ref="exceptionCodeResolver" /> <!--
↪ (2) -->
    <!-- Setting and Customization by project. -->
    <property name="order" value="3" /> <!-- (3) -->
    <property name="exceptionMappings"> <!-- (4) -->
        <map>
            <entry key="ResourceNotFoundException" value="common/error/
↪resourceNotFoundError" />
            <entry key="BusinessException" value="common/error/businessError"␣
↪/>
```

```
            <entry key="InvalidTransactionTokenException" value="common/error/
↪transactionTokenError" />
            <entry key=".DataAccessException" value="common/error/
↪dataAccessError" />
        </map>
    </property>
    <property name="statusCodes"> <!-- (5) -->
        <map>
            <entry key="common/error/resourceNotFoundError" value="404" />
            <entry key="common/error/businessError" value="409" />
            <entry key="common/error/transactionTokenError" value="409" />
            <entry key="common/error/dataAccessError" value="500" />
        </map>
    </property>
    <property name="excludedExceptions"> <!-- (6) -->
        <array>
            <value>org.springframework.web.util.NestedServletException</value>
        </array>
    </property>
    <property name="defaultErrorView" value="common/error/systemError" /> <!--
↪ (7) -->
    <property name="defaultStatusCode" value="500" /> <!-- (8) -->
</bean>

<!-- Settings View Resolver. -->
<mvc:view-resolvers>
    <mvc:jsp prefix="/WEB-INF/views/" /> <!-- (9) -->
</mvc:view-resolvers>
```

Sr. No.	Description
(1)	Add `SystemExceptionResolver` to bean definition.
(2)	Inject the object that resolves exception code (Message ID). Specify “exceptionCodeResolver” defined in `applicationContext.xml`.
	次のページに続く

Table. 4.11 – 前のページからの続き

Sr. No.	Description
(3)	Specify the order of priority for handling. The value can be “3”. When `<mvc:annotation-driven>` is specified, automatically *registered class* is given higher priority. ヒント: **Method to disable exception handling carried out by DefaultHandlerExceptionResolver** When exception handling is carried out by `DefaultHandlerExceptionResolver`, HTTP response code is set; however since View is not resolved, it needs to be resolved using Error Page element of `web.xml`. When it is required to resolve the View using `HandlerExceptionResolver` and not in `web.xml`, then priority order of `SystemExceptionResolver` should be set to “1”. By doing this, the handling process can be executed before `DefaultHandlerExceptionResolver`. For mapping of HTTP response codes when handling is done by `DefaultHandlerExceptionResolver`, refer to *HTTP response code set by DefaultHandlerExceptionResolver*.
(4)	Specify the mapping between name of the exception to be handled and View name. In the above settings, if class name of the exception class (or parent class) includes ”.DataAccessException”, “common/error/dataAccessError” will be treated as View name. If the exception class is “ResourceNotFoundException”, “common/error/resourceNotFoundError” will be treated as View name. **[Location to be customized for each project]**
(5)	Specify the mapping between View name and HTTP status code. In the above settings, when View name is “common/error/resourceNotFoundError”, “404(Not Found)” becomes HTTP status code. **[Location to be customized for each project]**

次のページに続く

Table. 4.11 – 前のページからの続き

Sr. No.	Description
(6)	Specify exception class to be excluded from handling. `org.springframework.web.util.NestedServletException` is excluded from the handling in order to notify to servlet container without handling the fatal error by `SystemExceptionResolver`. For reasons of exclusion from handling, refer to *"Regarding handling of fatal errors using @ExceptionHandler and SystemExceptionResolver"*.
(7)	Specify the default View name. In the above settings, if exception class does not include "ResourceNotFoundException", "BusinessException" and "InvalidTransactionTokenException", and if exception class (or parent class) name does not include ".DataAccessException", "common/error/systemError" becomes the destination View name. **[Location to be customized for each project]**
(8)	Specify default value of HTTP status code to be set in response header. It is recommended that you set **"500"(Internal Server Error)**. **Warning: Behavior when nothing is specified** Please note that it will be handled as **"200"(OK)**.

次のページに続く

Table. 4.11 – 前のページからの続き

Sr. No.	Description
(9)	Actual `View` depends on `ViewResolver` settings. In above settings, destination pages will be as given below. • `/WEB-INF/views/common/error/systemError.jsp` • `/WEB-INF/views/common/error/resourceNotFoundError.jsp` • `/WEB-INF/views/common/error/businessError.jsp` • `/WEB-INF/views/common/error/transactionTokenError.jsp` • `/WEB-INF/views/common/error/dataAccessError.jsp` --- **Tip:** `<mvc:view-resolvers>` is an XML element added from Spring Framework 4.1. If `<mvc:view-resolvers>` element is used, it is possible to define `ViewResolver` in simple manner. Example of definition when `<bean>` element is used in a conventional manner is given below. `<bean id="viewResolver" class="org.springframework.web.servlet.view.InternalResourceViewResolv <property name="prefix" value="/WEB-INF/views/" /> <property name="suffix" value=".jsp" /> </bean>` ---

AOP settings and interceptor class (`HandlerExceptionResolverLoggingInterceptor`) in order to output the log of exceptions handled by `HandlerExceptionResolver` should be added to bean definition.

- **spring-mvc.xml**

```
<!-- Setting AOP. -->
<bean id="handlerExceptionResolverLoggingInterceptor"
    class="org.terasoluna.gfw.web.exception.
↪HandlerExceptionResolverLoggingInterceptor"> <!-- (1) -->
    <property name="exceptionLogger" ref="exceptionLogger" /> <!-- (2) -->
</bean>
<aop:config>
    <aop:advisor advice-ref="handlerExceptionResolverLoggingInterceptor"
        pointcut="execution(* org.springframework.web.servlet.
↪HandlerExceptionResolver.resolveException(..))" /> <!-- (3) -->
</aop:config>
```

Sr. No.	Description
(1)	Add `HandlerExceptionResolverLoggingInterceptor` to bean definition.
(2)	Inject the logger object which outputs exception log. Specify the "exceptionLogger" defined in `applicationContext.xml`.
(3)	Apply `HandlerExceptionResolverLoggingInterceptor` to resolveException method of `HandlerExceptionResolver` interface. As per default settings, this class will not output the log for common library provided `org.terasoluna.gfw.common.exception.ResultMessagesNotificationException` class and its subclasses. The reason the exceptions of the sub class of `ResultMessagesNotificationException` are excluded from the log output is because their log output is carried out by `org.terasoluna.gfw.common.exception.ResultMessagesLoggingInterceptor`. If default settings need to be changed, refer to *About HandlerExceptionResolverLoggingInterceptor settings*.

Filter class (`ExceptionLoggingFilter`) used to output log of fatal errors and exceptions that are out of the boundary of Spring MVC should be added to bean definition and `web.xml`.

- **applicationContext.xml**

```
<!-- Filter. -->
<bean id="exceptionLoggingFilter"
    class="org.terasoluna.gfw.web.exception.ExceptionLoggingFilter" > <!--␣
↪(1) -->
    <property name="exceptionLogger" ref="exceptionLogger" /> <!-- (2) -->
</bean>
```

Sr. No.	Description
(1)	Add `ExceptionLoggingFilter` to bean definition.
(2)	Inject the logger object which outputs exception log. Specify the "exceptionLogger" defined in `applicationContext.xml`.

- **web.xml**

```
<filter>
    <filter-name>exceptionLoggingFilter</filter-name> <!-- (1) -->
    <filter-class>org.springframework.web.filter.DelegatingFilterProxy</
↪filter-class> <!-- (2) -->
</filter>
<filter-mapping>
    <filter-name>exceptionLoggingFilter</filter-name> <!-- (3) -->
    <url-pattern>/*</url-pattern> <!-- (4) -->
</filter-mapping>
```

Sr. No.	Description
(1)	Specify the filter name. Match it with the bean name of `ExceptionLoggingFilter` defined in `applicationContext.xml`.
(2)	Specify the filter class. The value should be fixed to `org.springframework.web.filter.DelegatingFilterProxy`.
(3)	Specify the name of the filter to be mapped. The value specified in (1).
(4)	Specify the URL pattern to which the filter must be applied. It is recommended that you use `/*` for outputting log of fatal errors and exceptions that are out of the boundary of Spring MVC.

- Output Log

```
date:2013-09-25 19:51:52    thread:tomcat-http--3   X-Track:
↪f94de92148f1489b9ceeac3b2f17c969        level:ERROR     logger:o.t.gfw.
↪common.exception.ExceptionLogger         message:[e.xx.fw.9001] An␣
↪exception occurred processing JSP page /WEB-INF/views/exampleJSPException.
↪jsp at line 13
```

Servlet Container Settings

Add error-page definition for Servlet Container in order to handle error response (HttpServletResponse#sendError) received through default exception handling functionality of Spring MVC, fatal errors and the exceptions that are out of the boundary of Spring MVC.

- **web.xml**

  Add definitions in order to handle error response (HttpServletResponse#sendError) received through default exception handling functionality of Spring MVC.

```
<error-page>
    <!-- (1) -->
    <error-code>404</error-code>
    <!-- (2) -->
    <location>/WEB-INF/views/common/error/resourceNotFoundError.jsp</location>
</error-page>
```

Sr. No.	Description
(1)	Specify the **HTTP Response Code** to be handled. **[Location to be customized for each project]** For HTTP response code for which response is sent using the default exception handling function of Spring MVC, refer to *HTTP response code set by DefaultHandlerExceptionResolver*.
(2)	Specify the file name. It should be specified with the path from Web application root. In the above settings, "${WebAppRoot}/WEB-INF/views/common/error/resourceNotFoundError.jsp" will be the View file. **[Location to be customized for each project]**

Add definitions in order to handle fatal errors and exceptions that are out of the boundary of Spring MVC.

```
<error-page>
    <!-- (3) -->
    <location>/WEB-INF/views/common/error/unhandledSystemError.html</location>
</error-page>
```

Sr. No.	Description
(3)	Specify the file name. Specify with a path from web application root. In the above settings, "${WebAppRoot}/WEB-INF/views/common/error/unhandledSystemError.html" will be the View file. **[Location to be customized for each project]**

Note: About the path specified in location tag

If a fatal error occurs, there is high possibility of getting another error if the path of dynamic contents is specified.; hence in location tag, **it is recommended that you specify a path of static contents such as HTML** and not dynamic contents such as JSP.

Note: If untraceable error occurs during development

If an unexpected error response (HttpServletResponse#sendError) occurs after carrying out the above settings, there may be cases wherein it cannot be determined what kind of error response occurred.

Error screen specified in location tag is displayed; however if the cause of error cannot be identified from logs, it

is possible to verify the error response (HTTP response code) on screen by commenting out the above settings.

When it is necessary to individually handle the exceptions that are out of the boundary of Spring MVC, definition of each exception should be added.

```
<error-page>
    <!-- (4) -->
    <exception-type>java.io.IOException</exception-type>
    <!-- (5) -->
    <location>/WEB-INF/views/common/error/systemError.jsp</location>
</error-page>
```

Sr. No.	Description
(4)	Specify the **Exception Class Name (FQCN)** to be handled.
(5)	Specify the file name. Specify it using a path from web application root. In above case, "${WebAppRoot}/WEB-INF/views/common/error/systemError.jsp" will be the View file. **[Location to be customized for each project]**

Coding Points (Service)

The coding points in Service when handling the exceptions are given below.

1. *Generating Business Exception*
2. *Generating System Exception*
3. *Catch the exception to continue the execution*

Generating Business Exception

The method of generating Business Exception is given below.

Note: Notes related to the method of generating business exception

- It is recommended that you generate the business exception by detecting violation of business rules in logic.
- When it is required by the API specification of underlying base framework or existing layer of application, that violation of business rule be notified through an exception, then it is OK to catch the exception and throw it as business exception.

Use of an exception to control the processing flow lowers the readability of overall logic and thus may reduce maintainability.

Generate business exceptions by detecting violation of business rules in logic.

Warning:

- It is assumed that business exception is generated in Service by default. In *AOP settings*, log of business exception occurred in class with `@Service` annotation, is being output. Business exceptions should not be logged in Controller etc. This rule can be changed if needed in the project.

- xxxService.java

```
...
@Service
public class ExampleExceptionServiceImpl implements ExampleExceptionService {
    @Override
    public String throwBusinessException(String test) {
        ...
        // int stockQuantity = 5;
        // int orderQuantity = 6;

        if (stockQuantity < orderQuantity) {                    // (1)
            ResultMessages messages = ResultMessages.error(); // (2)
            messages.add("e.ad.od.5001", stockQuantity);      // (3)
            throw new BusinessException(messages);            // (4)
        }
        ...
```

Sr. No.	Description
(1)	Check whether there is any violation of a business rule.
(2)	If there is a violation, generate ResultMessages. In the above example, ResultMessages of error level are being generated. For the details on method of generating ResultMessages, refer to *Message Management*.
(3)	Add ResultMessage to ResultMessages. Specify message ID as 1st argument (mandatory) and value to be inserted in message as 2nd argument (optional). The value to be inserted in message is a variable-length parameter; hence multiple values can be specified.
(4)	Specify ResultMessages and generate BusinessException.

Tip: For the purpose of explanation, `xxxService.java` logic is written in steps (2)-(4) as shown above, however it can also be written in a single step.

```
throw new BusinessException(ResultMessages.error().add(
    "e.ad.od.5001", stockQuantity));
```

- xxx.properties

  Add the below settings to the properties file.

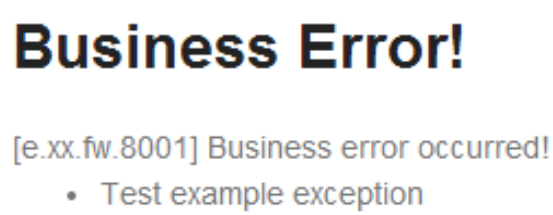
```
e.ad.od.5001 = Order number is higher than the stock quantity={0}. Change the␣
↪order number.
```

An application log as shown below is output.

```
date:2013-09-17 22:25:55    thread:tomcat-http--8   X-Track:
↪6cfb0b378c124b918e40ac0c32a1fac7        level:WARN      logger:o.t.gfw.
↪common.exception.ExceptionLogger         message:[e.xx.fw.8001]␣
↪ResultMessages [type=error, list=[ResultMessage [code=e.ad.od.5001,␣
↪args=[5], text=null]]]
org.terasoluna.gfw.common.exception.BusinessException: ResultMessages␣
↪[type=error, list=[ResultMessage [code=e.ad.od.5001, args=[5], text=null]]]
```

```
// stackTrace omitted
...

date:2013-09-17 22:25:55    thread:tomcat-http--8   X-Track:
 ↪6cfb0b378c124b918e40ac0c32a1fac7        level:DEBUG     logger:o.t.gfw.web.
 ↪exception.SystemExceptionResolver    message:Resolving exception from␣
 ↪handler [public java.lang.String org.terasoluna.exception.app.example.
 ↪ExampleExceptionController.home(java.util.Locale,org.springframework.ui.
 ↪Model)]: org.terasoluna.gfw.common.exception.BusinessException:
 ResultMessages [type=error, list=[ResultMessage [code=e.ad.od.5001, args=[5],
 text=null]]]
date:2013-09-17 22:25:55    thread:tomcat-http--8   X-Track:
 ↪6cfb0b378c124b918e40ac0c32a1fac7        level:DEBUG     logger:o.t.gfw.web.
 ↪exception.SystemExceptionResolver    message:Resolving to view 'common/
 ↪error/businessError' for exception of type [org.terasoluna.gfw.common.
 ↪exception.BusinessException], based on exception mapping [BusinessException]
date:2013-09-17 22:25:55    thread:tomcat-http--8   X-Track:
 ↪6cfb0b378c124b918e40ac0c32a1fac7        level:DEBUG     logger:o.t.gfw.web.
 ↪exception.SystemExceptionResolver    message:Applying HTTP status code 409
date:2013-09-17 22:25:55    thread:tomcat-http--8   X-Track:
 ↪6cfb0b378c124b918e40ac0c32a1fac7        level:DEBUG     logger:o.t.gfw.web.
 ↪exception.SystemExceptionResolver    message:Exposing Exception as model␣
 ↪attribute 'exception'
```

Displayed screen



> **Warning:** It is recommended that you handle business exception in Controller and display a message on each business screen. The above example illustrates a screen which is displayed when the exception is not handled in Controller.

Catch an exception to generate a business exception

```
try {
    order(orderQuantity, itemId );
} catch (StockNotEnoughException e) {                    // (1)
    throw new BusinessException(ResultMessages.error().add(
            "e.ad.od.5001", e.getStockQuantity()), e); // (2)
}
```

Sr. No.	Description
(1)	Catch the exception that occurs when a business rule is violated.
(2)	Specify ResultMessages and **Cause Exception (e)** to generate BusinessException.

Generating System Exception

The method of generating SystemException is given below.

Generate a system exception by detecting a system error in logic.

```
if (itemEntity == null) {                                    // (1)
    throw new SystemException("e.ad.od.9012",
        "not found item entity. item code [" + itemId + "]."); // (2)
}
```

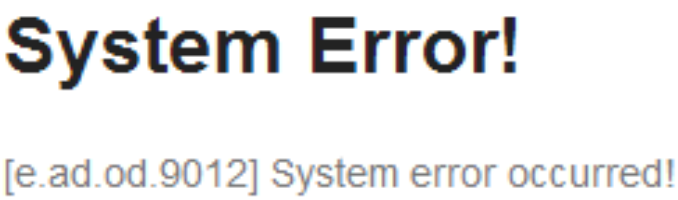
Sr. No.	Description
(1)	Check whether the system is in normal state. In this example, it is checked whether the requested product code (itemId) exists in the product master (Item Master). If it does not exist in the product master, it would be considered that a resource which should have been available in the system, does not exist and hence it is treated as system error.
(2)	If the system is in an abnormal state, specify the exception code (message ID) as 1st argument. Specify exception message as 2nd argument to generate SystemException. In the above example, the value of variable “itemId” is inserted in message text.

Application log as shown below is output.

```
date:2013-09-19 21:03:06      thread:tomcat-http--3   X-Track:
↪c19eec546b054d54a13658f94292b24f        level:DEBUG     logger:o.t.gfw.web.
↪exception.SystemExceptionResolver    message:Resolving exception from␣
↪handler [public java.lang.String org.terasoluna.exception.app.example.
↪ExampleExceptionController.home(java.util.Locale,org.springframework.ui.
↪Model)]: org.terasoluna.gfw.common.exception.SystemException: not found␣
↪item entity. item code [10-123456].
date:2013-09-19 21:03:06      thread:tomcat-http--3   X-Track:
↪c19eec546b054d54a13658f94292b24f        level:DEBUG     logger:o.t.gfw.web.
↪exception.SystemExceptionResolver    message:Resolving to default view
↪'common/error/systemError' for exception of type [org.terasoluna.gfw.common.
↪exception.SystemException]
```

```
date:2013-09-19 21:03:06      thread:tomcat-http--3   X-Track:
↪c19eec546b054d54a13658f94292b24f        level:DEBUG     logger:o.t.gfw.web.
↪exception.SystemExceptionResolver    message:Applying HTTP status code 500
date:2013-09-19 21:03:06      thread:tomcat-http--3   X-Track:
↪c19eec546b054d54a13658f94292b24f        level:DEBUG     logger:o.t.gfw.web.
↪exception.SystemExceptionResolver    message:Exposing Exception as model␣
↪attribute 'exception'
date:2013-09-19 21:03:06      thread:tomcat-http--3   X-Track:
↪c19eec546b054d54a13658f94292b24f        level:ERROR     logger:o.t.gfw.
↪common.exception.ExceptionLogger         message:[e.ad.od.9012] not found␣
↪item entity. item code [10-123456].
org.terasoluna.gfw.common.exception.SystemException: not found item entity.␣
↪item code [10-123456].
      at org.terasoluna.exception.domain.service.ExampleExceptionServiceImpl.
↪throwSystemException(ExampleExceptionServiceImpl.java:14) ~
↪[ExampleExceptionServiceImpl.class:na]
...
// stackTrace omitted
```

Displayed screen



Note: It is recommended to have a common system error screen rather than creating multiple screens for system errors.

The screen mentioned in this guideline displays a (business-wise) message ID for system errors and has a fixed message. This is because there is no need to inform the details of error to the operator and it is sufficient to only convey that the system error has occurred. Therefore, in order to enhance the response for inquiry against system errors, the Message ID which acts as a key for the message text is displayed on the screen, in order to make the analysis easier for the development side. Displayed error screens should be designed in accordance with the UI standards of each project.

Catch an exception to generate system exception.

```
try {
    return new File(preUploadDir.getFile(), key);
} catch (FileNotFoundException e) { // (1)
    throw new SystemException("e.ad.od.9007",
        "not found upload file. file is [" + preUploadDir.getDescription() +
↪"]."
```

```
        e); // (2)
}
```

Sr. No.	Description
(1)	Catch the checked exception classified as system error.
(2)	Specify the exception code (message ID), message, **Cause Exception (e)** to generate SystemException.

Catch the exception to continue the execution

When it is necessary to catch the exception to continue the execution, the exception should be logged before continuing the execution.

When fetching customer interaction history from external system fails, the process of fetching information other than customer history can still be continued. This is illustrated in the following example.

In this example, although fetching of customer history fails, business process does not have to be stopped and hence the execution continues.

```
@Inject
ExceptionLogger exceptionLogger; // (1)

// ...
```

```
InteractionHistory interactionHistory = null;
try {
    interactionHistory = restTemplete.getForObject(uri, InteractionHistory.
↪class, customerId);
} catch (RestClientException e) { // (2)
    exceptionLogger.log(e); // (3)
}

// (4)
Customer customer = customerRepository.findOne(customerId);

// ...
```

Sr. No.	Description
(1)	Inject the object of `org.terasoluna.gfw.common.exception.ExceptionLogger` provided by the common library for log output.
(2)	Catch the exception to be handled.
(3)	Output the handled exception to log. In the example, log method is being called; however if the output level is known in advance and if there is no possibility of any change in output level, it is ok to call info, warn, error methods directly.
(4)	Continue the execution just by outputting the log in (3).

An application log as shown below is output.

```
date:2013-09-19 21:31:47      thread:tomcat-http--3   X-Track:
↪df5271ece2304b12a2c59ff494806397        level:ERROR     logger:o.t.gfw.
↪common.exception.ExceptionLogger         message:[e.xx.fw.9001] Test␣
↪example exception
org.springframework.web.client.RestClientException: Test example exception
...
// stackTrace omitted
```

Warning: When log() is used in `exceptionLogger`, since it will be output at error level; by default, it will be output in monitoring log also.

```
date:2013-09-19 21:31:47  X-Track:df5271ece2304b12a2c59ff494806397      ␣
↪level:ERROR     message:[e.xx.fw.9001] Test example exception
```

As shown in following example, if there is no problem in continuing the execution, and if monitoring log is being monitored through application monitoring, it should be set to a level such that it will not get monitored at log output level or defined such that it does not get monitored from the log content (log message).

```
} catch (RestClientException e) {
    exceptionLogger.info(e);
```

```
}
```

As per default settings, monitoring log other than error level will not be output. It is output in application log as follows:

```
date:2013-09-19 22:17:53    thread:tomcat-http--3   X-Track:
↪999725b111b4445b8d10b4ea44639c61        level:INFO      logger:o.t.gfw.
↪common.exception.ExceptionLogger         message:[e.xx.fw.9001] Test␣
↪example exception
org.springframework.web.client.RestClientException: Test example exception
```

Coding Points (Controller)

The coding points in Controller while handling the exceptions are given below.

1. *Method to handle exceptions at request level*
2. *Method to handle exception at use case level*

Method to handle exceptions at request level

Handle the exception at request level and set the message related information to Model.
Then, by calling the method for displaying the View, generate the model required by the View and determine the View name.

```
@RequestMapping(value = "change", method = RequestMethod.POST)
public String change(@Validated UserForm userForm,
                     BindingResult result,
                     RedirectAttributes redirectAttributes,
                     Model model) {          // (1)

    // omitted

    User user = userHelper.convertToUser(userForm);
    try {
        userService.change(user);
    } catch (BusinessException e) {                                    // (2)
        model.addAttribute(e.getResultMessages());                     // (3)
        return viewChangeForm(user.getUserId(), model);                // (4)
    }

    // omitted
```

```
}
```

Sr. No.	Description
(1)	As of the parameters of the method, define Model in argument as an object which will be used to link the error information with View.
(2)	Exceptions which need to be handled should be caught in application code.
(3)	Add ResultMessages object to Model.
(4)	Call the method for displaying the View at the time of error. Fetch the model and View name required for View display and then return the View name to be displayed.

Method to handle exception at use case level

Handle the exception at use case level and generate ModelMap (ExtendedModelMap) wherein message related information etc. is stored.

Then, by calling the method for displaying the View, generate the model required by the View and determine the View name.

```
@ExceptionHandler(BusinessException.class) // (1)
@ResponseStatus(HttpStatus.CONFLICT) // (2)
public ModelAndView handleBusinessException(BusinessException e) {
    ExtendedModelMap modelMap = new ExtendedModelMap();                    // (3)
    modelMap.addAttribute(e.getResultMessages());                          // (4)
    String viewName = top(modelMap);                                       // (5)
    return new ModelAndView(viewName, modelMap);                           // (6)
}
```

Sr. No.	Description
(1)	The exception class which need to be handled should be specified in the value attribute of `@ExceptionHandler` annotation. You can also specify multiple exceptions which are in the scope of handling.
(2)	Specify the HTTP status code to be returned to value attribute of `@ResponseStatus` annotation. In the example, "409: Conflict" is specified.
(3)	Generate ExtendedModelMap as an object to link the error information and model information with View.
(4)	Add ResultMessages object to ExtendedModelMap.
(5)	Call the method to display the View at the time of error and fetch model and View name necessary for View display.
(6)	Generate ModelAndView wherein View name and Model acquired in steps (3)-(5) are stored and then return the same.

Warning: When `java.lang.Exception` and `javax.servlet.ServletException` are to be handled by a method which assigns `@ExceptionHandler`, the fatal errors cannot be notified to servlet container since `NestedServletException`which wraps the fatal error get handled unintentionally. For details, refer to *"Regarding handling of fatal errors using @ExceptionHandler and SystemExceptionResolver"*.

In such a case, in order to notify fatal error to the servlet container, :ref: NestedServletException is excluded from handling by SystemExceptionResolver<exception-handling-how-to-use-application-configuration-app-label> and NestedServletException is handled by the method which assigns @ExceptionHandler and then is thrown again. Implementation example is as below.

```
@ExceptionHandler(NestedServletException.class) // (1)
public void handleNestedServletException(NestedServletException e)␣
↪throws NestedServletException {
    throw e; // (2)
}
```

Sr. No.	Description
(1)	Assign `@ExceptionHandler` annotation and specify `NestedServletException.class`.
(2)	Throw `NestedServletException` thus handled again.

Regarding capturing Exception and ServletException by multiple controllers

When it is necessary to describe `@ExceptionHandler` which throws `NestedServletException` again by multiple controllers, use of `@ControllerAdvice` should be considered. For details of `@ControllerAdvice`, refer to *Implementing "@ControllerAdvice"*.

Coding points (JSP)

The coding points in JSP while handling the exceptions are given below.

1. *Method to display messages on screen using MessagesPanelTag*
2. *Method to display system exception code on screen*

Tip: When Internet Explorer is a support browser, implementation should be such that size of the HTML responded as error screen should be 513 bytes or more.

This is because in case of Internet Explorer, if the following three conditions such as

- Response status code is Error (4xx and 5xx)
- Response HTML is 512 bytes or less
- Browser setting "Show Friendly HTTP Error Messages" is valid

are satisfied, the mechanism is such that friendly messages created by Internet Explorer will be displayed.

Method to display messages on screen using MessagesPanelTag

The example below illustrates implementation at the time of outputting ResultMessages to an arbitrary location.

```
<t:messagesPanel /> <!-- (1) -->
```

Sr. No.	Description
(1)	<t:messagesPanel> tag should be specified at a location where the message is to be output. For details on usage of <t:messagesPanel> tag, refer to *Message Management*.

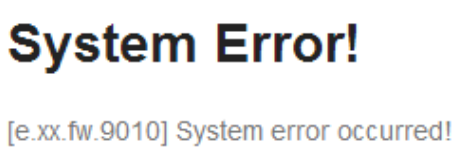
Method to display system exception code on screen

The example below illustrates implementation at the time of displaying exception code (message ID) and fixed message at an arbitrary location.

```
<p>
    <c:if test="${!empty exceptionCode}">  <!-- (1) -->
        [${f:h(exceptionCode)}]            <!-- (2) -->
    </c:if>
    <spring:message code="e.cm.fw.9999" /> <!-- (3) -->
</p>
```

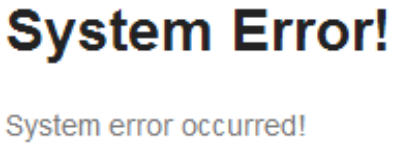
Sr. No.	Description
(1)	Check whether exception code (message ID) exists. Perform existence check when the exception code (message ID) is enclosed with symbols as shown in the above example.
(2)	Output exception code (message ID).
(3)	Output fixed message acquired from message definition.

- Output Screen (With exceptionCode)



- Output Screen (Without exceptionCode)

Note: Messages to be output at the time of system exception



- When a system exception occurs, it is recommended that you display a message which would only convey that a system exception has occurred, without outputting a detailed message from which cause of error can be identified or guessed.
- On displaying a detailed message from which cause of error can be identified or guessed, the vulnerabilities of system are likely to get exposed.

Note: Exception code (message ID)

- When a system exception occurs, it is desirable to output an exception code (message ID) instead of a detailed message.
- By outputting the exception code (message ID), it is possible for development team to respond quickly to the inquiries from system user.
- Only a system administrator can identify the cause of error from exception code (message ID); hence the risk of exposing the vulnerabilities of system is lowered.

4.2.4 How to use (Ajax)

For the exception handling of Ajax, refer to *Ajax*.

4.2.5 Appendix

1. *Exception handling classes provided by the common library*
2. *About SystemExceptionResolver settings*

3. *HTTP response code set by DefaultHandlerExceptionResolver*

Exception handling classes provided by the common library

In addition to the classes provided by Spring MVC, the classes for carrying out exception handling are being provided by the common library.

The roles of classes are as follows:

Table.4.13: **Table - Classes under org.terasoluna.gfw.common.exception package**

Sr. No.	Class	Role
(1)	ExceptionCode Resolver	An interface for resolving exception code (message ID) of exception class. An exception code is used for identifying the exception type and is expected to be output to system error screen and log. It is referenced from `ExceptionLogger`, `SystemExceptionResolver`.

次のページに続く

Table. 4.13 – 前のページからの続き

Sr. No.	Class	Role
(2)	SimpleMapping ExceptionCode Resolver	Implementation class of `ExceptionCodeResolver` contains the mapping of exception class name and exception code, through which the exception code is resolved. The name of exception class need not be FQCN, it can be a part of FQCN or parent class name. **Warning:** • Note that when a part of FQCN is specified, it may be matched with classes which were not assumed. • Note that when the name of parent class is specified, all the child classes will also be matched.
(3)	enums. ExceptionLevel	enum indicating exception level corresponding to exception class. INFO, WARN, ERROR are defined.
(4)	ExceptionLevel Resolver	Interface for resolving exception level (log level) of exception class. Exception level is a code for identifying the level of exception and is used to switch the output level of log. It is referenced from `ExceptionLogger`.

次のページに続く

Table. 4.13 – 前のページからの続き

Sr. No.	Class	Role
(5)	DefaultExceptionLevelResolver	This is an implementation class of `ExceptionLevelResolver` which resolves the exception level by first character of exception code. If the first character (case insensitive) is, 1. “i”, treat it as `ExceptionLevel.INFO` 2. “w”, treat it as `ExceptionLevel.WARN` 3. “e”, treat it as `ExceptionLevel.ERROR` 4. a character other than the above, treat as `ExceptionLevel.ERROR` level. This class is implemented in accordance with the rules of the message ID described in *Message* guidelines.
(6)	ExceptionLogger	A class to output log of exceptions. Monitoring log (only messages) and application log (both messages and stack trace) can be output. This class is used from Filter or Interceptor classes provided by the framework. When the execution is to be continued by handling the exceptions in application code, log should be output by using this class.
(7)	ResultMessagesLoggingInterceptor	This is an interceptor class for logging an occurrence of exception which stores `ResultMessages` (sub exception of `ResultMessagesNotificationException`). Output all the logs at WARN level. It is assumed that this Interceptor will be applied to the methods of class with `@Service` annotation. Log is output using `ExceptionLogger`.

次のページに続く

Table. 4.13 – 前のページからの続き

Sr. No.	Class	Role
(8)	BusinessException	This is an exception class to notify violation of business rules. This exception should be generated in the logic of domain layer. It inherits `java.lang.RuntimeException`; hence the transactions are rolled back as per default behavior. If you want to commit the transactions, this exception class needs to be specified in noRollbackFor or noRollbackForClassName of `@Transactional` annotation.
(9)	Resource NotFoundException	This is an exception class to notify that the specified resource (data) does not exist in the system. This exception should mainly be generated in the logic of domain layer. It inherits `java.lang.RuntimeException`; hence the transactions are rolled back as per default behavior.
(10)	ResultMessages Notification Exception	This is an abstract exception class to notify that the exception holds the result messages (`ResultMessages`), and `BusinessException` and `ResourceNotFoundException` inherit it in the common library. It inherits `java.lang.RuntimeException`; hence the transactions are rolled back as per default behavior. If this exception class is inherited, log of warn level will be output by `ResultMessagesLoggingInterceptor`.
(11)	SystemException	This is an exception class to notify a system or application error. This exception should be generated in the logic of application layer or domain layer. It inherits `java.lang.RuntimeException`; hence the transactions are rolled back as per default behavior.

次のページに続く

Table. 4.13 – 前のページからの続き

Sr. No.	Class	Role
(12)	ExceptionCodeProvider	This is an interface indicating that it has a role to hold the exception code; it is implemented by `SystemException` in the common library. If an exception class that implements this interface is created, it can be used with the exception handling mechanism of ExceptionCodeResolver, to use the exception code of this exception class as it is.

Table.4.14 **Table - Classes under org.terasoluna.gfw.web.exception package**

Sr. No.	Class	Role
(13)	SystemException Resolver	This is a class for handling the exceptions that will not be handled by `HandlerExceptionResolver`, which is registered automatically when `<mvc:annotation-driven>` is specified. It inherits `SimpleMappingExceptionResolver` provided by Spring MVC and adds the functionality of referencing ResultMessages of exception code from View.
(14)	HandlerException ResolverLogging Interceptor	This is an Interceptor class to output the log of exceptions handled by `HandlerExceptionResolver`. In this Interceptor class, the output level of log is switched based on the classification of HTTP response code resolved by `HandlerExceptionResolver`. 1. When it is “100-399”, log is output at INFO level. 2. When it is “400-499”, log is output at WARN level. 3. When it is “500-”, log is output at ERROR level. 4. When it is “-99”, log is not output. By using this Interceptor, it is possible to output the log of all exceptions which are within the boundary of Spring MVC. Log is output using `ExceptionLogger`.
(15)	ExceptionLogging Filter	This is a Filter class to output the log of fatal errors and exceptions that are out of the boundary of Spring MVC. All logs are output at ERROR level. When this Filter is used, fatal errors and all exceptions that are out of the boundary of Spring MVC can be output to log. Log is output using `ExceptionLogger`.

About SystemExceptionResolver settings

This section describes the settings which are not explained above. The settings should be performed depending on the requirements.

Table.4.15: **List of settings not explained above**

Sr. No.	Field Name	Property Name	Description	Default Value
(1)	Attribute name of result message	resultMessagesAttribute	Specify the attribute name (String) used for setting message of businessException to Model. This attribute name is used for accessing result message in View(JSP).	resultMessages
(2)	Attribute name of exception code (message ID)	exceptionCode Attribute	Specify the attribute name (String) used for setting the exception code (message ID) to HttpServletRequest. This attribute name is used for accessing the exception code (message ID) in View(JSP).	exceptionCode
(3)	Header name of exception code (message ID)	exceptionCode Header	Specify the header name (String) used for setting the exception code (message ID) to response header of HttpServletResponse.	X-Exception-Code
(4)	Attribute name of exception object	exceptionAttribute	Specify the attribute name (String) used for setting the handled exception object to Model. This attribute name is used for accessing the exception object in View(JSP).	exception
(5)	List of handler (Controller) objects to be used as this ExceptionResolver	mappedHandlers	Specify the object list (Set) of handlers that use this ExceptionResolver. Only the exceptions in the specified handler objects will be handled. **This setting should not be specified.**	Not specified **When specified, the operation is not guaranteed.**

次のページに続く

Table. 4.15 – 前のページからの続き

Sr. No.	Field Name	Property Name	Description	Default Value
(6)	List of handler (Controller) classes that use this Exception-Resolver	mappedHandlerClasses	Specify the class list (Class[]) of handlers that use this ExceptionResolver. Only the exceptions that occur in the specified handler classes are handled. **This setting should not be specified**	Not specified **When specified, the operation is not guaranteed.**
(7)	Cache control flag of HTTP response	preventResponseCaching	Set the flag (true: Yes, false: No) for cache control at the time of HTTP response. If true: Yes is specified, HTTP response header to disable cache is added.	false: No

(1)-(3) are the settings of `org.terasoluna.gfw.web.exception.SystemExceptionResolver`.
(4) is the setting of
`org.springframework.web.servlet.handler.SimpleMappingExceptionResolver`.
(5)-(7) are the settings of
`org.springframework.web.servlet.handler.AbstractHandlerExceptionResolver`.

Attribute name of result message

If the message set by handling the exception using SystemExceptionResolver and the message set by handling the exception in application code, both are to be output in separate messagesPanel tags in View(JSP), then specify an attribute name exclusive to SystemExceptionResolver.

The example below illustrates settings and implementation when changing the default value to "resultMessagesForExceptionResolver".

- **spring-mvc.xml**

```
<bean class="org.terasoluna.gfw.web.exception.SystemExceptionResolver">

    <!-- omitted -->

    <property name="resultMessagesAttribute" value=
↪"resultMessagesForExceptionResolver" /> <!-- (1) -->

    <!-- omitted -->
</bean>
```

- **jsp**

```
<t:messagesPanel messagesAttributeName="resultMessagesForExceptionResolver"/>
↪<!-- (2) -->
```

Sr. No.	Description
(1)	Specify "resultMessagesForExceptionResolver" in result message attribute name (resultMessagesAttribute).
(2)	Specify attribute name that was set in SystemExceptionResolver, in message attribute name (messagesAttributeName).

Attribute name of exception code (message ID)

When default attribute name is being used in the application code, a different value should be set in order to avoid duplication. When there is no duplication, there is no need to change the default value.

The example below illustrates settings and implementation when changing the default value to "exceptionCodeForExceptionResolver".

- **spring-mvc.xml**

```
<bean class="org.terasoluna.gfw.web.exception.SystemExceptionResolver">

    <!-- omitted -->

    <property name="exceptionCodeAttribute" value=
↪"exceptionCodeForExceptionResolver" /> <!-- (1) -->

    <!-- omitted -->
</bean>
```

- **jsp**

```
<p>
    <c:if test="${!empty exceptionCodeForExceptionResolver}">  <!-- (2) -->
        [${f:h(exceptionCodeForExceptionResolver)}]            <!-- (3) -->
    </c:if>
    <spring:message code="e.cm.fw.9999" />
</p>
```

Sr. No.	Description
(1)	Specify "exceptionCodeForExceptionResolver" in attribute name (exceptionCodeAttribute) of exception code (message ID).
(2)	Specify the value (exceptionCodeForExceptionResolver) set in SystemExceptionResolver as a variable name (for Empty check) to be tested.
(3)	Specify the value (exceptionCodeForExceptionResolver) set in SystemExceptionResolver as a variable name to be output.

Header name of exception code (message ID)

When default header name is being used, set a different value in order to avoid duplication. When there is no duplication, there is no need to change the default value.

The example below illustrates settings and implementation when changing the default value to "X-Exception-Code-ForExceptionResolver".

- **spring-mvc.xml**

```
<bean class="org.terasoluna.gfw.web.exception.SystemExceptionResolver">

    <!-- omitted -->

    <property name="exceptionCodeHeader" value="X-Exception-Code-
↪ForExceptionResolver" /> <!-- (1) -->

    <!-- omitted -->
</bean>
```

Sr. No.	Description
(1)	Specify "X-Exception-Code-ForExceptionResolver" in the header name (exceptionCodeHeader) of the exception code (message ID).

Attribute name of exception object

When default attribute name is being used in the application code, set a different value in order to avoid duplication. When there is no duplication, there is no need to change the default value.

The example below illustrates settings and implementation when changing the default value to "exceptionForExceptionResolver".

- **spring-mvc.xml**

```
<bean class="org.terasoluna.gfw.web.exception.SystemExceptionResolver">

    <!-- omitted -->

    <property name="exceptionAttribute" value="exceptionForExceptionResolver"␣
↪/> <!-- (1) -->

    <!-- omitted -->
</bean>
```

- **jsp**

```
<p>[Exception Message]</p>
<p>${f:h(exceptionForExceptionResolver.message)}</p> <!-- (2) -->
```

Sr. No.	Description
(1)	Specify “exceptionForExceptionResolver” in attribute name (exceptionAttribute) of exception object.
(2)	Specify the value (exceptionForExceptionResolver) set in SystemExceptionResolver as a variable name for fetching message from exception object.

Cache control flag of HTTP response

When header for cache control is to be added to HTTP response, specify true: Yes.

- **spring-mvc.xml**

```
<bean class="org.terasoluna.gfw.web.exception.SystemExceptionResolver">

    <!-- omitted -->

    <property name="preventResponseCaching" value="true" /> <!-- (1) -->

    <!-- omitted -->
</bean>
```

Sr. No.	Description
(1)	Set the cache control flag (preventResponseCaching) of HTTP response to true: Yes.

Note: HTTP response header when Yes is specified

If cache control flag of HTTP response is set to Yes, the following HTTP response header is output.

Cache-Control:no-store

Adding a header for cache control using `SystemExceptionResolver` is an option to suppress display of unintended error screen due to browser cache, however a header for cache control can

also be added for security by using a Spring Security function. For Spring Security function, refer A:ref:*SpringSecurityLinkageWithBrowser*.

About HandlerExceptionResolverLoggingInterceptor settings

This section describes the settings which are not explained above. The settings should be performed depending on the requirements.

Table.4.16 **List of settings not described above**

Sr. No.	Field Name	Property Name	Description	Default Value
(1)	List of exception classes to be excluded from scope of logging	ignoreExceptions	Amongst the exceptions handled by `HandlerExceptionResolver`, the exception classes which are not to be logged should be specified in list format. When an exception of the specified exception class and sub class occurs, the same is not logged in this class. Only the exceptions which are logged at a different location (using different mechanism) should be specified here.	`ResultMessagesNoti` Exceptions of `ResultMessagesNoti` and its sub classes are logged by `ResultMessagesLogg` hence, as default settings, they are being excluded.

List of exception classes to be excluded from scope of logging

The settings when exception classes provided in the project are to be excluded from the scope of logging, are as follows:

- **spring-mvc.xml**

```
<bean id="handlerExceptionResolverLoggingInterceptor"
    class="org.terasoluna.gfw.web.exception.
↪HandlerExceptionResolverLoggingInterceptor">
    <property name="exceptionLogger" ref="exceptionLogger" />
    <property name="ignoreExceptions">
        <set>
            <!-- (1) -->
            <value>org.terasoluna.gfw.common.exception.
↪ResultMessagesNotificationException</value>
```

```
            <!-- (2) -->
            <value>com.example.common.XxxException</value>
        </set>
    </property>
</bean>
```

Sr. No.	Description
(1)	`ResultMessagesNotificationException` specified in the default settings of common library, should be specified under exception to be excluded.
(2)	Specify the exception classes created in the project.

The settings when all the exception classes are to be logged are as follows:

- **spring-mvc.xml**

```
<bean id="handlerExceptionResolverLoggingInterceptor"
    class="org.terasoluna.gfw.web.exception.
↪HandlerExceptionResolverLoggingInterceptor">
    <property name="exceptionLogger" ref="exceptionLogger" />
    <!-- (3) -->
    <property name="ignoreExceptions"><null /></property>
</bean>
```

Sr. No.	Description
(3)	Specify `null` in ignoreExceptions properties. If `null` is specified, all the exception classes will become target for logging.

HTTP response code set by DefaultHandlerExceptionResolver

The mapping between framework exceptions handled using DefaultHandlerExceptionResolver and HTTP status code is shown below.

Sr. No.	Handled framework exceptions	HTTP Status Code
(1)	org.springframework.web.servlet.mvc.multiaction.NoSuchRequestHandlingMethodException	404
(2)	org.springframework.web.HttpRequestMethodNotSupportedException	405
(3)	org.springframework.web.HttpMediaTypeNotSupportedException	415
(4)	org.springframework.web.HttpMediaTypeNotAcceptableException	406
(5)	org.springframework.web.bind.MissingServletRequestParameterException	400
(6)	org.springframework.web.bind.ServletRequestBindingException	400
(7)	org.springframework.beans.ConversionNotSupportedException	500
(8)	org.springframework.beans.TypeMismatchException	400
(9)	org.springframework.http.converter.HttpMessageNotReadableException	400
(10)	org.springframework.http.converter.HttpMessageNotWritableException	500
(11).	org.springframework.web.bind.MethodArgumentNotValidException	400

次のページに続く

Table. 4.17 – 前のページからの続き

Sr. No.	Handled framework exceptions	HTTP Status Code
(12)	org.springframework.web.multipart.support.MissingServletRequestPartException	400
(13)	org.springframework.validation.BindException	400
(14)	org.springframework.web.servlet.NoHandlerFoundException	404
(15)	org.springframework.web.context.request.async.AsyncRequestTimeoutException	503

4.3 Session Management

4.3.1 Overview

This chapter explains Session Management in a Web application.

In Web applications, data is exchanged between the client and the server, using HTTP.

HTTP itself does not have a feature to physically maintain a session. Instead, it provides a mechanism wherein, a logical session is maintained by linking a session identifier value (session ID), between the client and the server.

Cookie or request parameter is used to link the session ID between client and server.

The figure below illustrates establishment of logical session.

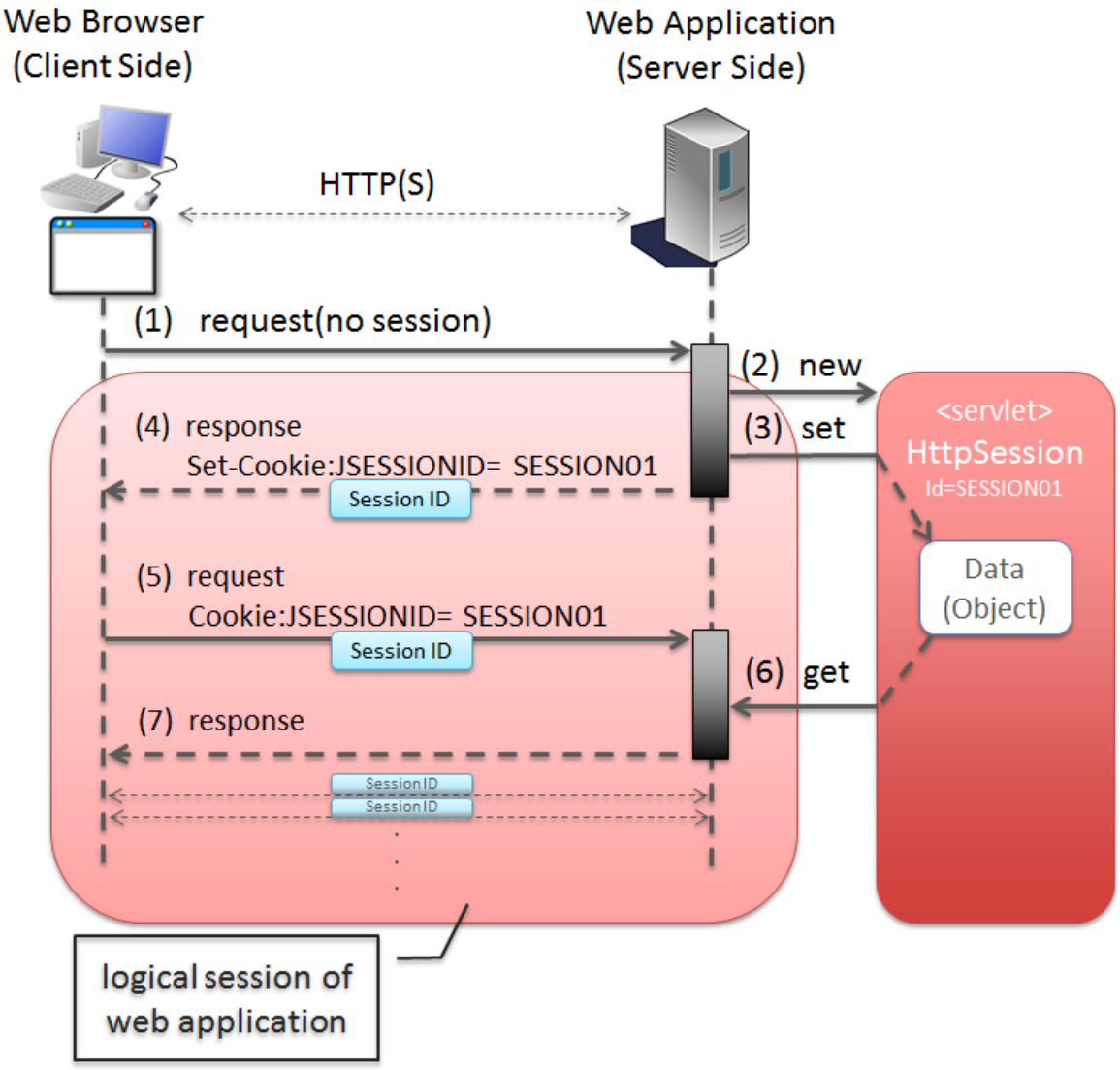


Figure.4.13 **Picture - Establishment of logical session**

Sr. No.	Description
(1)	Web browser (Client) accesses the Web application (Server) when session is not established.
(2)	Web application creates `HttpSession` object for storing the session with Web browser. Session ID is issued at the time of `HttpSession` object creation.
(3)	Web application stores the data sent by the Web browser in `HttpSession` object.

次のページに続く

Table. 4.18 – 前のページからの続き

Sr. No.	Description
(4)	Web application sends a response to the Web browser. By setting “JSESSIONID = Issued session ID” in the “Set-Cookie” header of response, session ID is linked to the Web Browser. Linked session ID is stored in Cookie.
(5)	Web browser links the session ID to Web application by setting “JSESSIONID = Issued session ID” in the “Cookie” header of request. Application server on which the Web application is deployed, fetches `HttpSession` object corresponding to the session ID linked from the Web browser and maps it with the request.
(6)	Web application fetches the data stored in the request mentioned in (1), from the `HttpSession` object associated with the request. **Same data can be accessed across requests.**
(7)	Web application returns the response to Web browser.

Note: About the parameter name to link session ID

In JavaEE Serlvet specifications, the default parameter name to link a session ID is JSESSIONID.

Session lifecycle

Instead of implementing as Controller process, session lifecycle control (create, discard, timeout detection) is implemented by using the processes provided by framework or common library.

Note: `"Session"` mentioned in the following explanation refers to `javax.servlet.http.HttpSession` object provided by Servlet API. `HttpSession` object is the Java object representing the logical session described above.

Generating a session

When Web application is created by the method recommended in this guideline, session can be generated by any one of the following processes.

Sr. No.	Description
1.	Authentication/authorization process provided by Spring Security. Timing and status of session generation can be specified by Spring Security settings. For details on session management in Spring Security, refer to *How to use*.
2.	CSRF token check process provided by Spring Security When a session is already established, new session is not generated. For details on CSRF token check, refer to *CSRF Countermeasures*.
3.	Transaction token check process provided by common library. When a session is already established, new session is not generated. For details on Transaction token check, refer to *Double Submit Protection*.

次のページに続く

Table. 4.19 – 前のページからの続き

Sr. No.	Description
4.	Process for passing a model (form object, domain object etc.) to redirect destination request, by using addFlashAttribute method of `RedirectAttributes` interface. When a session is already established, new session is not generated. For details on `RedirectAttributes` and Flash scope, refer to *Passing data while redirecting request*.
5.	Process for storing a model (form object, domain object etc.) in a session, using `@SessionAttributes` annotation. The specified model (form object, domain object etc.) is stored in session. When a session is already established, new session is not generated. For details on how to use `@SessionAttributes` annotation, refer to *Using @SessionAttributes annotation*.
6.	Process that uses session-scoped bean in Spring Framework. When a session is already established, new session is not generated. For details on how to use session-scoped bean, refer to *Using session-scoped bean of Spring Framework*.

Note: In Sr. Nos. 4, 5, 6 mentioned above, whether the session is used or not is specified by Controller implementation whereas, session generation timing is controlled by the framework. In other words, `HttpSession` API need not be used directly as Controller process.

Storing attributes in a session

When Web application is created by the methods recommended in this guideline, attributes (objects) are stored in session by any one of the following processes.

Sr. No.	Description
1.	Authentication process provided by Spring Security. Authenticated user information is stored in the session. For details on Spring Security authentication process, refer to *Authentication*.
2.	CSRF token check process provided by Spring Security. Issued token value is stored in session. For details on CSRF token check, refer to *CSRF Countermeasures*.
3.	Transaction token check process provided by common library. Issued token value is stored in session. For details on Transaction token check, refer to *Double Submit Protection*.
4.	Process for passing a model (form object, domain object etc.) to redirect destination request, by using addFlashAttribute method of `RedirectAttributes` interface. Object specified in the addFlashAttribute method of `RedirectAttributes` interface, is stored in an area called Flash scope in the session. For details on `RedirectAttributes` and Flash scope, refer to *Passing data while redirecting request*.

次のページに続く

Table. 4.20 – 前のページからの続き

Sr. No.	Description
5.	Process for storing a model (form object, domain object etc.) in session, using `@SessionAttributes` annotation. The specified model (form object, domain object etc.) is stored in session. For the details on how to use `@SessionAttributes` annotation, refer to, *Using @SessionAttributes annotation*.
6.	Process that uses session-scoped bean in Spring Framework. Session-scoped bean is stored in session. For details on how to use session-scoped bean, refer to *Using session-scoped bean of Spring Framework*.

Note: Timing to store the object in session is controlled by the framework. Hence, setAttribute method of `HttpSession` object is not called as Controller process.

Deleting attributes from a session

When Web application is created by the methods recommended in this guideline, attributes (objects) are deleted from a session by any one of the following processes.

Sr. No.	Description
1.	Logout process provided by Spring Security. Authenticated user information is deleted from the session. For details on Spring Security logout process, refer to *Authentication*.
2.	Transaction token check process provided by common library. When the value of issued token exceeds the upper limit allocated to NameSpace, token value that is not in use, is deleted from the session. For details on Transaction token check, refer to *Double Submit Protection*.
3.	Redirect process after the object is stored in Flash scope. Object specified in the addFlashAttribute method of `RedirectAttributes` interface, is deleted from the Flash scope area of session.
4.	Framework process, after the setComplete method of `SessionStatus` object is called as Controller process. Object specified by `@SessionAttributes` annotation is deleted from the session.

Note: Timing to delete the object is controlled by the framework. Hence, removeAttribute method of `HttpSession` object is not called as Controller process.

Discarding a session

When Web application is created by the methods recommended in this guideline, session is discarded using any one of the following processes.

Sr. No.	Description
1.	Logout process provided by Spring Security. For details on Spring Security logout process, refer to *Authentication*.
2.	Process for detecting session timeout of application server.

The scenario in which a session is discarded explicitly is illustrated below.

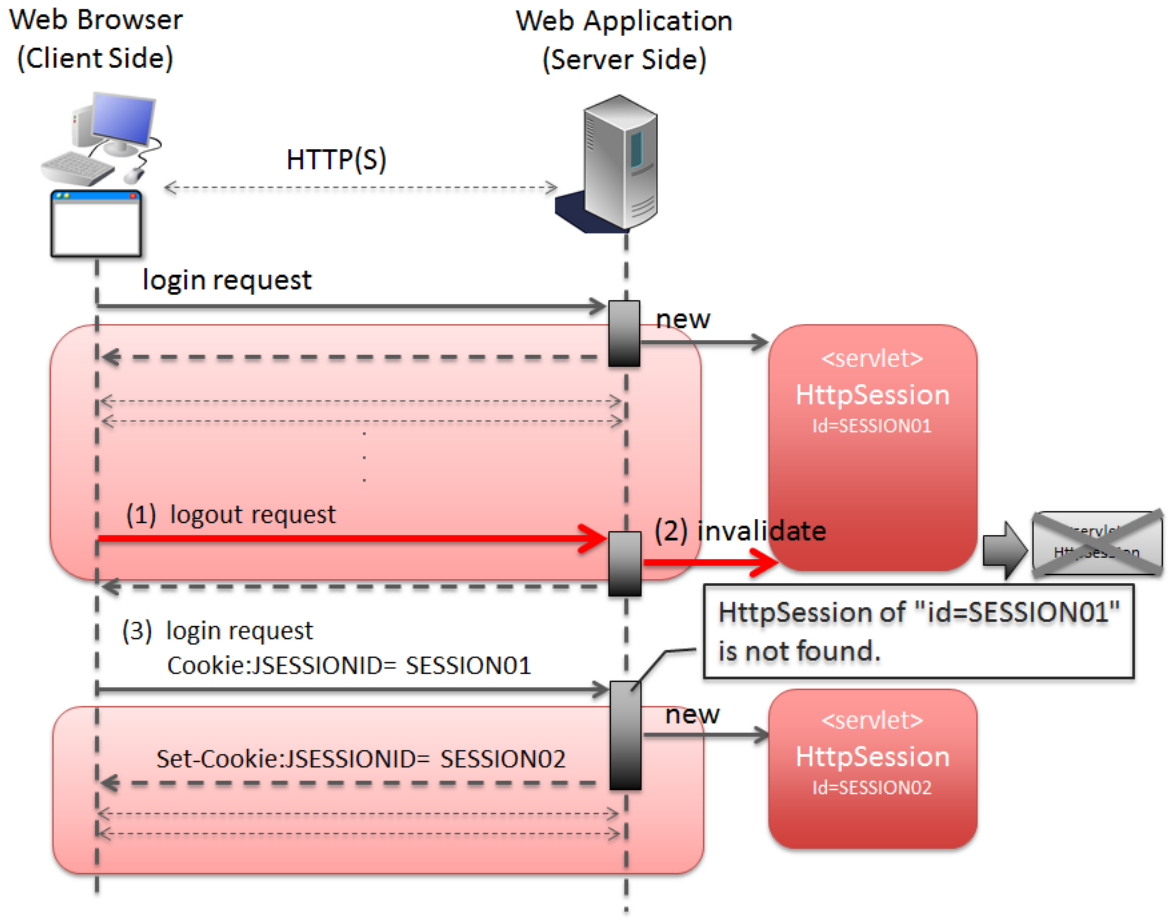


Figure.4.14 **Picture - Invalidate session by processing of Web Application**

Sr. No.	Description
(1)	Access the process that discards session from Web browser. When using Spring Security, the logout process provided by it is used to discard the session. For details on logout process of Spring Security, refer to *Authentication*.
(2)	Web application discards `HttpSession` object corresponding to the session ID linked from Web browser. At this point, `HttpSession` object with the ID, `"SESSION01"`, disappears from server side.
(3)	When the discarded session is accessed from Web browser using the respective session ID, a new session is generated, as `HttpSession` object corresponding to the requested session ID does not exist. In the above example, a session with ID `"SESSION02"` is being generated.

The scenario in which a session is automatically discarded due to timeout is illustrated below.

Sr. No.	Description
(1)	When an established session is not accessed for a particular period, application server detects session timeout.
(2)	Application server discards the session for which session timeout is detected.
(3)	When the session is accessed from Web browser after session timeout occurs, session timeout error is returned to Web browser since `HttpSession` object corresponding to the session ID requested from the Web browser, does not exist.

Note: Designing session timeout

When data needs to be stored in the session, the design should include 'session timeout'. It is recommended to set the timeout as short as possible, especially when the data to be stored is large.

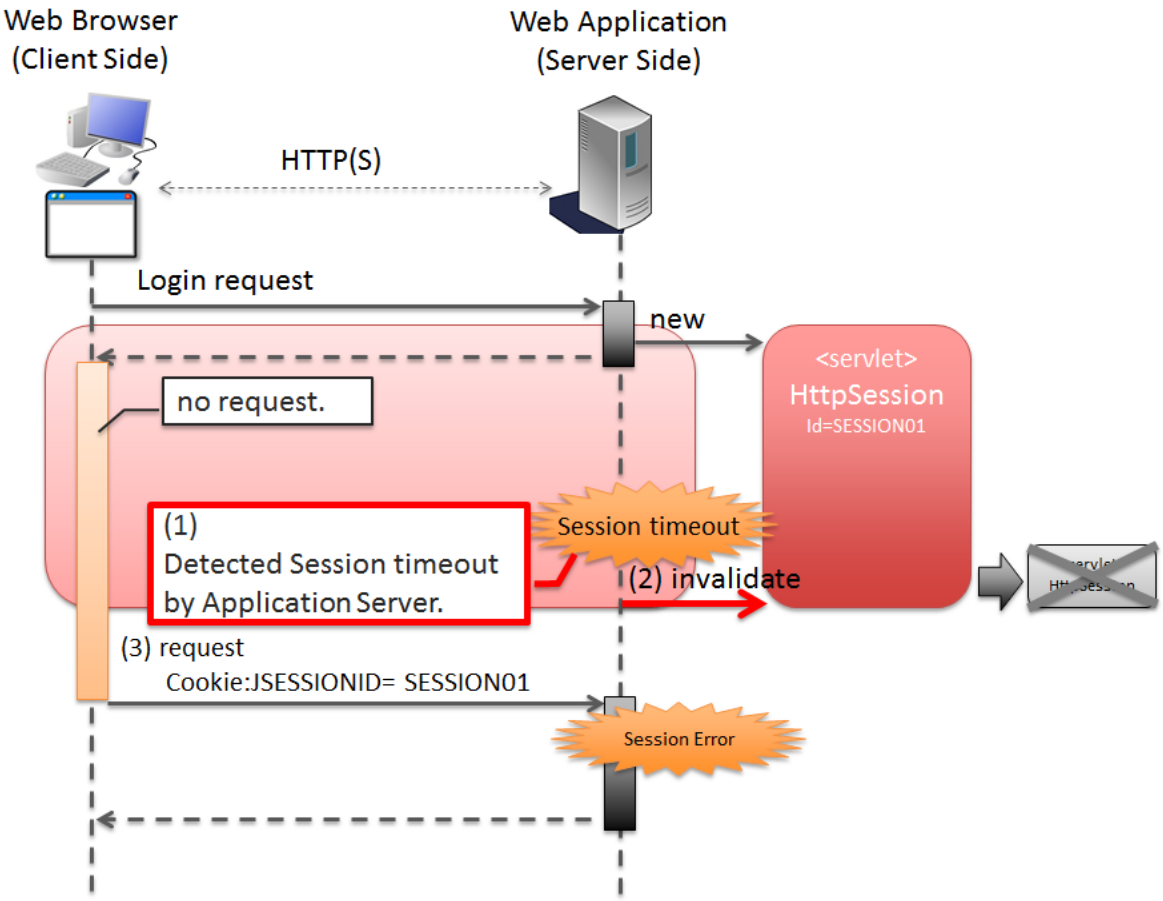


Figure.4.15 **Picture - Invalidate session by Application Server**

Note: Default session timeout period

Default session timeout period differs depending on application server.

- Tomcat: 1800 seconds (30 minutes)
- WebLogic: 3600 seconds (60 minutes)
- WebSphere: 1800 seconds (30 minutes)
- JBoss: 1800 seconds (30 minutes)

Detecting a request after session timeout

When Web application is created by the method recommended in this guideline, a request subsequent to session timeout is detected by any one of the following processes.

Sr. No.	Description
1.	Session timeout check process provided by Spring Security. Session timeout check is performed as per default settings in Spring Security. Therefore, to store data in session, settings to validate the timeout check process of a Spring Security session, are required. For details on timeout check process in Spring Security, refer to *How to use*.
2.	When not using Spring Security, timeout check process needs to be implemented in Servlet Filter or `HandlerInterceptor`of Spring MVC.

The scenario in which session timeout is detected using session check process provided by Spring Security, is illustrated below.

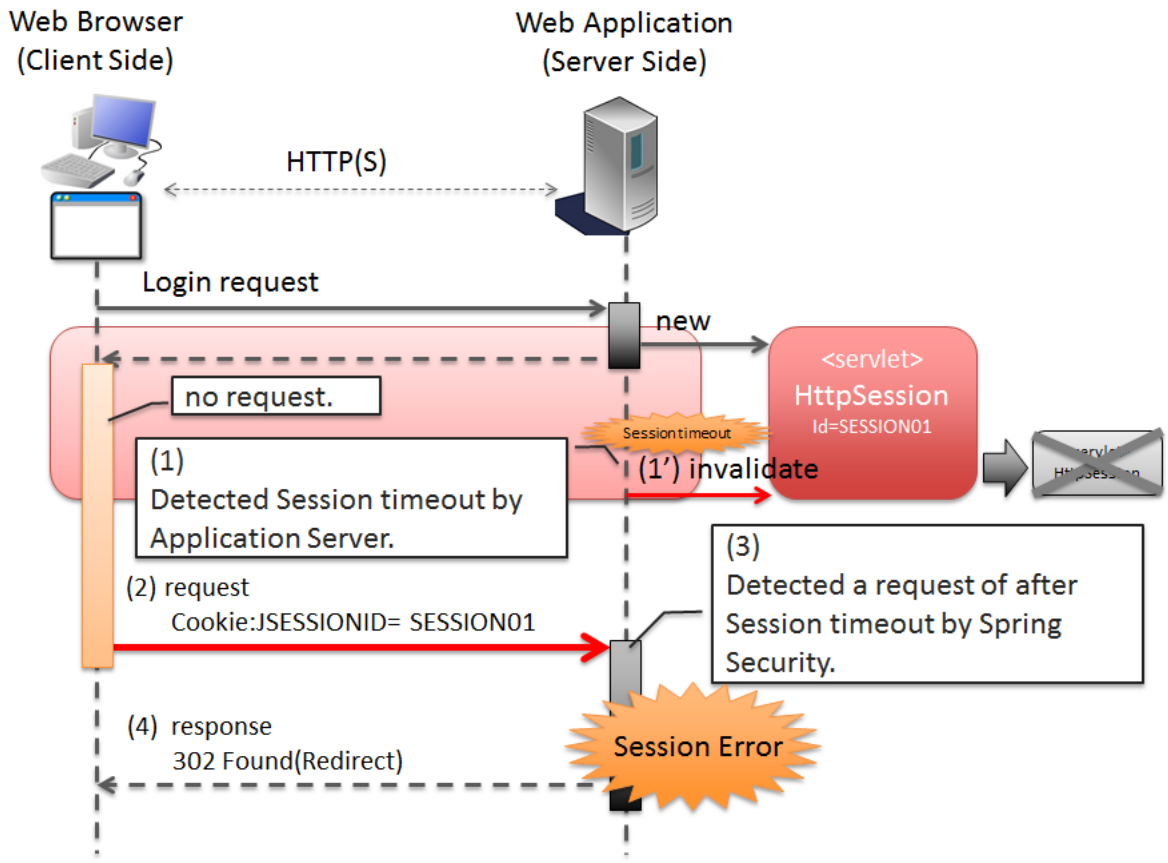


Figure.4.16 **Picture - Detected a request after session timeout by Spring Security**

Sr. No.	Description
(1)	When an established session is not accessed for a particular period, application server detects session timeout and discards the session.
(2)	Web browser accesses the session after the session timeout occurs.
(3)	Spring Security throws session timeout error as, the `HttpSession` object corresponding to the session ID linked from client, does not exist. In default Spring Security implementation, response is sent to the request for redirecting to URL to display the error screen.

Note: Necessity of session timeout check

For the processes having precondition, "Data should be stored in session", session timeout check should always be performed. If this check is not performed, unexpected system errors and operations may occur, as data required by the process cannot be fetched.

About using a session

When data needs to be shared across multiple screens (multiple requests), it can be shared easily by storing this data in session.

However, as against the advantage that the data can be easily shared, storing the data in session also results in application constraints etc. Hence, whether or not to use session should be decided by considering application and system requirements.

Note: Rather than simply storing the data in session, this guideline initially recommends to consider a policy wherein session is not used. Further, if session is used, it recommends storing only the absolutely required data in it.

Note: Storing the data applicable to following conditions has proved better.

- Data that does not provide linkage between use cases, but for such data, the status when it is moved and returned from a different use case needs to be stored.

For example, search condition of a List screen corresponds to this pattern.

When the search condition of a List screen returns from another use case (example: “Change searched data” use case), many a times its status before moving to other use case needs to be stored as a functional requirement.

Search conditions can also be shared in hidden state but this causes excessive dependency between use cases and implementing the application is expected to be complex.

- Data for which linkage between use cases is necessary.

  For example, data stored in the shopping site cart corresponds to this pattern.

  Data stored in the shopping site cart involves use cases of ” Adding the product to the cart”, ” Displaying the cart”, “Changing the status of the cart” and ” Purchasing the products in the cart”. It requires the data linkage of all these use cases.

  However, when scalability needs to be considered, sometimes it is better to store the data in database instead of session.

Advantages and disadvantages of using session

The advantages and disadvantages of using session are as follows:

- **Advantages**
  - Data can be shared across multiple screens (multiple requests) easily when a single process is composed in multiple screens such as a Wizard Screens.
  - By storing the fetched data in session, number of executions of data acquisition can be reduced.
- **Disadvantages**
  - When the screen with same process is opened in multiple browsers or various tabs, data consistency cannot be maintained, as mutual operations interfere with the data stored in session.

    In order to maintain data consistency, control is required so that screen with the same process cannot be opened in multiple browsers or tabs.

    This control can be implemented by using Transaction token check provided by common library, however it results in reduced usability.
  - Session data is usually stored in application server memory; hence memory usage also increases with increase in data stored in the session.

    If unnecessary data that is no longer used in a process is left as it is, it gets excluded from garbage collection process, leading to memory exhaustion. Such data needs to be deleted from session at a stage when it is rendered unnecessary.

    The timing to delete unnecessary data from session needs to be designed separately.
  - Storing process data in session may lower scalability of application server.

Note: Any one of the following methods is required to scale out the application server.

1. Performing session replication and sharing the session information with all application servers.

   When performing session replication, the load on replication process increases in proportion to the data stored in session and number of application servers for replication.

   Therefore, issues owing to scale out such as, possible degradation in response time etc. need to be considered.

2. Distributing all requests of the same session to the same application server, using load balancer.

   When requests are distributed to the same application server and if the server is down, the process cannot be continued by another application server.

   Therefore, it needs to be noted that, this method may not be feasible in applications that demand high availability (service level).

Scale out method should be determined upon considering each of these points.

Advantages and disadvantages of not using session

In order to avoid the disadvantages faced while using session, all the data required for server processing can be implemented by linking as request parameters.
The advantages and disadvantages of not using session are as follows:

- **Advantages**
  - As data is not stored at server side, even if multiple browsers and tabs are used, their operations do not interfere with each other. Therefore, multiple screens of the same process can be started without impairing the usability.
  - As data is not stored at server side, continuous utilization of memory can be controlled.
  - Number of factors that lower the scalability of application server are reduced.
- **Disadvantages**
  - Data required for server processing needs to be sent as request parameter. As a result, even the items that are not displayed on the screen need to be specified as hidden items.

    Thus, JSP implementation code increases.

    This can be minimized by creating JSP tag library.
  - Amount of data flowing to the network increases, as all the data required for server processing needs to be sent through requests.

  - Data required for screen display needs to be fetched each time. Hence, number of executions of data acquisition increases.

About the data to be stored in session

Following points need to be considered for the data to be stored in session.

- It should be serializable object (object implementing `java.io.Serializable`).
- It should not be a large object that can cause memory exhaustion.

Serializable objects

Data to be stored in the session may be input or output to disk or network under specific conditions.
Therefore the objects need to be serializable.

Cases where data is input/output to disk are as follows:

- When application server is stopped while a session is active, the session and the data stored in it are saved to the disk.

  The saved session and data stored in it, are restored with the start of application server.

  Support status for this data restoring operation differs depending on application server.
- If session storage area is about to overflow or if the session is not accessed for a particular period of time since the last access, there is a possibility of session swap-out.

  The swapped-out session is swapped-in when the session is accessed again.

  Conditions etc. for swap-out differ depending on application server.

Cases where data is input/output to network are as follows:

- To perform replication of session in another application server, data stored in the session is sent to that application server via network.

Amount of data to be stored in session

It is recommended that the data to be stored in session should be as compact as possible.

When large amount of data is stored in session, it fatally degrades the performance. Hence, it is recommended to design such that, large amount of data will not be stored in session.

The main causes of performance degradation are as follows:

- When storing large amount of data in session, application server settings need to be enabled such that session is easily swapped-out so as to prevent memory exhaustion.

Swap-out is a “Heavy” process. Hence, it occurring very frequently may affect the overall application performance.

Support status for swap-out operation or setting method, differs depending on application server.

- When carrying out session replication, serialization and deserialization of an object are performed.

  Serialization and deserialization of an object with large capacity being “heavy” processes, may affect performance, such as response time.

To make the session data compact, storing the data applicable to the following conditions in request scope instead of session scope, should be considered.

- Read-only data that cannot be edited by screen operations.

  If latest data is fetched at a time when it is required, and if the fetched data is displayed in View (JSP) by storing it in request scope, it need not be stored in session.

- Data that can be edited by screen operations. It can be shared only in the screen operations of a use case.

  If data can be shared in all screen transitions as hidden HTML fields, it need not be stored in session.

Points to be considered in case of application server clustering

A normal system is rarely composed of a single application server. Considering requirements such as availability, efficiency etc. it consists of multiple servers.

Therefore, when storing the data in session, any one of the following mechanisms needs to be applied in accordance with system requirements.

1. For systems that require high availability (service level), if one AP server is down, it should be possible to continue the processing on another AP server.

   To be able to continue processing on another AP server when one AP server is down, session information needs to be shared amongst all AP servers. Hence, session replication needs to be carried out by application servers that are configured as a cluster.

   As an alternate method, session information can be shared by setting the session storage location to cache server such as Oracle Coherence or database.

   It would be better to consider setting the session storage location to a cache server like Oracle Coherence or database if, number of AP servers, amount of data stored in session and number of sessions that can be pasted simultaneously are in large quantity.

2. For systems that do not require high availability (service level), processing need not be continued on another server if the AP server is down.

   Therefore, session information need not be shared amongst all AP servers. Thus, it is alright if all the requests in a same session are distributed to the same AP server, using load balancer functionality.

Warning: When the Web application is created by methods recommended in this guideline, either of the above mechanisms needs to be applied to store the following data in session.

- User information authenticated by the Spring Security authentication process.
- Token values issued by Spring Security CSRF token check.
- Token values issued by the transaction token check provided by common library.

About storage location of session

The session storage location can also be set in the In-memory data grids such as Key-Value Store or Oracle Coherence, other than AP server memory.

There is room for consideration when scalability is required.

The implementation method to change the storage location of session differs according to the AP server and the storage location itself. Therefore, it is omitted in this guideline.

4.3.2 How to use

This guideline recommends using any one of the following methods to store data in session.

1. *Using @SessionAttributes annotation*
2. *Using session-scoped bean of Spring Framework*

Warning: `HttpSession` API can be called directly by specifying the `HttpSession` object as an argument of Controller handler method. However, **as a rule, this guideline strongly recommends not to use the HttpSession API directly.**

`HttpSession` API may be used directly only for those processes that cannot be implemented otherwise. However, in most of the business processes, it is not necessary to use the HttpSession API directly. Therefore, set such that `HttpSession` object is not specified as an argument of the Controller handler method.

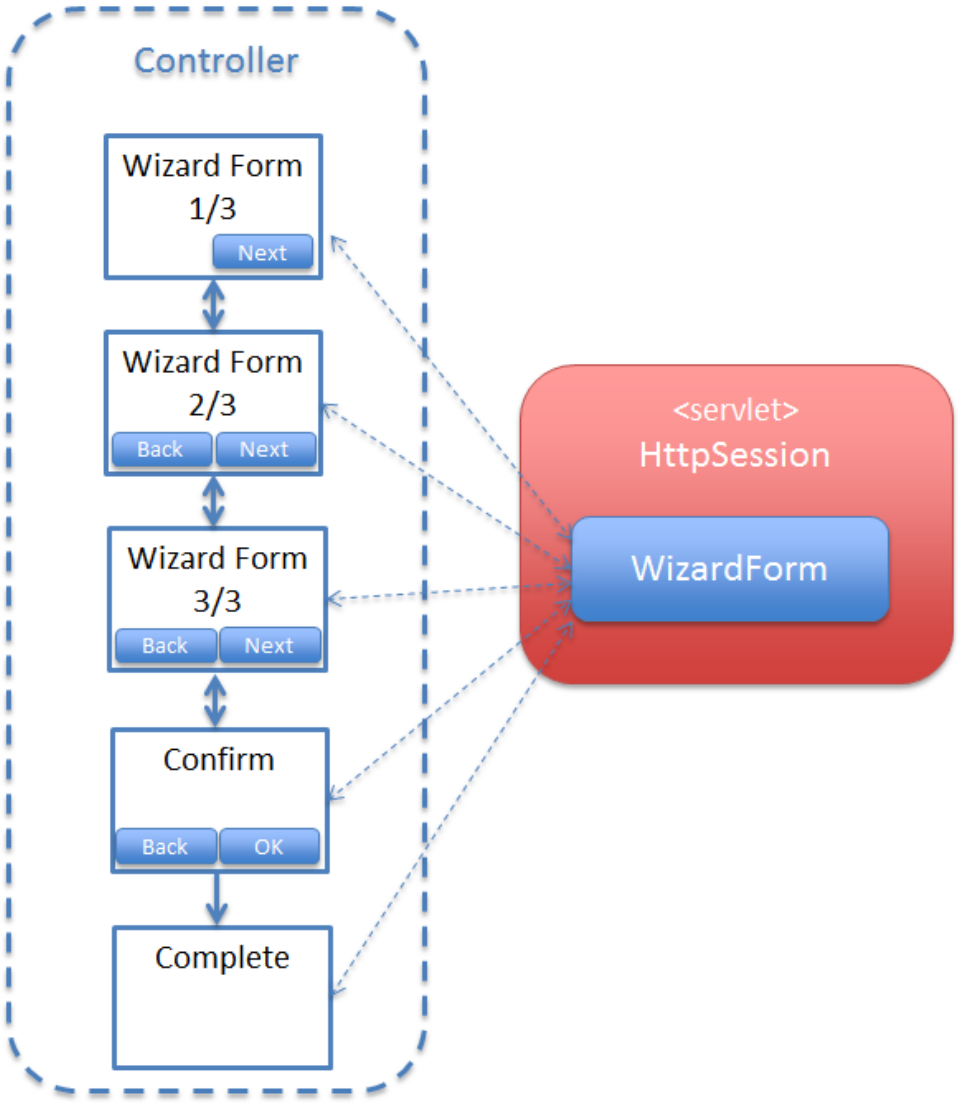
Using `@SessionAttributes` annotation

`@SessionAttributes` annotation is used for sharing data during screen transitions in Controller.

However, cases where each of the screens namely, the input screen, confirmation screen and completion screen consists of one page, it is better to share the data using 'hidden' HTML field rather than using session.

Cases where the input screen consists of multiple pages or when complicated screen transitions are involved, adopting the method, to store form object in session using `@SessionAttributes` annotation, should be considered.

Application designing and implementation can be simplified by storing the form object in session.



Specifying the object to be stored in session

Specify the object to be stored in session by specifying `@SessionAttributes` annotation in "class".

```
@Controller
@RequestMapping("wizard")
@SessionAttributes(types = { WizardForm.class, Entity.class }) // (1)
public class WizardController {
    // ...
}
```

Sr. No.	Description
(1)	In "types" attribute of `@SessionAttributes` annotation, specify the object type to be stored in session. Amongst the objects added to `Model` object using `@ModelAttribute` annotation or addAttribute method of `Model`, the objects matching with the type specified in "types" attribute, are stored in session. In the above example, objects of `WizardForm` class and `Entity` class are stored in session.

Note: Management unit for life cycle

Life cycles of the objects stored in session using `@SessionAttributes` annotation, are managed at Controller level.

When setComplete method of `SessionStatus` object is called, all the objects specified with '' @SessionAttributes '' annotation are deleted from session. Therefore, to store the objects having different life cycles in session, it is necessary to divide the Controller.

Warning: Points to be considered when using @SessionAttributes annotation

As described above, life cycle is managed at Controller level. However, when the object with the same attribute name in multiple Controllers, is stored in session using `@SessionAttributes` annotation, its life cycle is managed across Controllers.

In case of a process wherein, screen can be operated simultaneously by opening another window or tab, it can cause an error as the same object is being accessed. Therefore, to use the class of the same form object in multiple Controllers, first different values (attribute names) should be specified in "value" attribute of `@ModelAttribute`annotation. Then the same values should be specified in "value" attribute of `@SessionAttributes` annotation.

The object to be stored in session can also be specified in attribute name.

How to specify is explained below.

```
@Controller
@RequestMapping("wizard")
@SessionAttributes(value = { "wizardCreateForm" }) // (2)
public class WizardController {

    // ...

    @ModelAttribute(value = "wizardCreateForm")
    public WizardForm setUpWizardForm() {
        return new WizardForm();
    }

    // ...
}
```

Sr. No.	Description
(2)	Specify the attribute name of the object to be stored in session, in "value" attribute of `@SessionAttributes` annotation. Among the objects added to `Model` object using `@ModelAttribute` annotation or addAttribute method of `Model`, the objects matching with the attribute name specified in `value` attribute, are stored in session. In the above example, objects with attribute name `"wizardCreateForm"` are stored in session.

Adding object to session

Object can be added to session using the following two methods.

- Method with `@ModelAttribute` annotation is used to return the object to be added to the session.
- addAttribute method of `Model` object is used to add the object stored in session.

Object added to `Model` object is stored in session as per the type of `@SessionAttributes` annotation and value of "value" attribute. Hence, there is no need to consider the session when implementation is carried out using handler method of Controller.

How to return the object to be stored in session using the method with `@ModelAttribute` annotation, is explained below.

It is recommended to create object using this method when storing form object in session.

```
@ModelAttribute(value = "wizardForm") // (1)
public WizardForm setUpWizardForm() {
    return new WizardForm();
}
```

Sr. No.	Description
(1)	In "value" attribute, specify the attribute name to be stored in `Model` object. In the above example, the object returned is stored in session, with attribute name `"wizardForm"`. When "value" attribute is specified, method with `@ModelAttribute` annotation is no longer called by the subsequent requests after the object is stored in session. Thus, it has an advantage wherein, unnecessary objects are not generated.

Warning: Operation in case "value" attribute of @ModelAttribute annotation is omitted

When "value" attribute is omitted, method with `@ModelAttribute` annotation is called by all requests in order to generate default attribute name. Therefore, it has a disadvantage of unnecessary objects being generated. Hence, **This method should not be used when storing objects in session.**

```
@ModelAttribute // (1)
public WizardForm setUpWizardForm() {
    return new WizardForm();
}
```

Sr. No.	Description
(1)	Use the method with `@ModelAttribute` annotation to generate and return the object to be added in session. In the above example, object returned with attribute name `"wizardForm"` annotation, is stored in session.

How to add object to session using addAttribute method of `Model` object is explained below.

When storing the Domain object to session, this method is used to add the object.

```
@RequestMapping(value = "update/{id}", params = "form1")
public String updateForm1(@PathVariable("id") Integer id, WizardForm form,
        Model model) {
```

```
    Entity loadedEntity = entityService.getEntity(id);
    model.addAttribute(loadedEntity); // (3)
    beanMapper.map(loadedEntity, form);
    return "wizard/form1";
}
```

Sr. No.	Description
(3)	Add the object to be stored in session using addAttribute method of `Model` object. In the above example, the object fetched from domain layer with the attribute name `"entity"` is stored in session.

Fetching the object stored in session

The object stored in session can be received as an argument of Controller handler method.

There is no need to consider the session in Controller handler method, as the object stored session gets stored in `Model` object as per the attribute value of `@SessionAttributes` annotation.

```
@RequestMapping(value = "save", method = RequestMethod.POST)
public String save(@Validated({ Wizard1.class, Wizard2.class,
        Wizard3.class }) WizardForm form,   // (1)
        BindingResult result,
        Entity entity,                      // (2)
        RedirectAttributes redirectAttributes) {
    // ...
    return "redirect:/wizard/save?complete";
}
```

Sr. No.	Description
(1)	Fetch the object stored in `Model` object. In the above example, object stored in session scope with attribute name `"wizardForm"` is passed to argument form. For details on `Wizard1.class` , `Wizard2.class` , `Wizard3.class` specified by `@Validated` annotation, refer to *Example of screen transition implementation using @SessionAttributes in wizard format* of Appendix.
(2)	In the above example, object stored in session scope with attribute name `"entity"`, is passed to argument “entity”.

Note: How to prevent binding of request parameters when receiving object stored in session scope

When an object stored in the session scope is received as an argument of the handler method, there is a possibility that the request parameter is bound to the argument.

In order to prevent the request parameter from being bound, it can be realized by getting the object stored in the session scope from the `Model`object in the handler method without receiving it from the argument of the handler method, but it is not type-safe because it is necessary to specify the attribute name of the object to be acquired as a character string.

On the other hand, in Spring Framework 4.3, the `binding`attribute is added to the `ModelAttribute`annotation, so that it becomes possible to specify whether or not to bind request parameters as arguments. By attaching the `@ ModelAttribute`annotation as an argument and specifying `false`as the `binding`attribute, it is possible to prevent binding of request parameters and objects stored in the scope can be acquired.

In the example below, an object stored in the session scope with an attribute name `entity`is acquired by preventing binding of request parameters.

```
@RequestMapping(value = "save", method = RequestMethod.POST)
public String save(@Validated({ Wizard1.class, Wizard2.class,
        Wizard3.class }) WizardForm form,
        BindingResult result,
        @ModelAttribute(binding = false) Entity entity,
        RedirectAttributes redirectAttributes) {
    // ...
    return "redirect:/wizard/save?complete";
}
```

When the object to be passed to the argument of Controller handler method does not exist in `Model` object, the operation changes depending on whether `@ModelAttribute` annotation is specified or not.

- When `@ModelAttribute` annotation is not specified, a new object is created and passed as argument. The created object is stored in `Model` object and subsequently in session as well.

Note: Redirect operations

Created object is not stored in session if it is redirected to a transition destination. Therefore, when referring to the created object in redirect process, it is necessary to store the object in Flash scope using addFlashAttribute method of `RedirectAttributes`.

- When `@ModelAttribute` annotation is specified, `org.springframework.web.HttpSessionRequiredExcep` occurs.

```
@RequestMapping(value = "save", method = RequestMethod.POST)
public String save(@Validated({ Wizard1.class, Wizard2.class,
        Wizard3.class }) WizardForm form, // (3)
        BindingResult result,
        @ModelAttribute Entity entity, // (4)
        RedirectAttributes redirectAttributes) {
    // ...
    return "redirect:/wizard/save?complete";
}
```

Sr. No.	Description
(3)	Input validation is performed by setting specific verification groups (`Wizard1.class`, `Wizard2.class`, `Wizard3.class`) using `@Validated` annotation. For details on input validation, refer to *Input Validation*.
(4)	When `@ModelAttribute` annotation is specified in argument and when the target object that does not exist in session is called, `HttpSessionRequiredException` occurs. `HttpSessionRequiredException` occurs due to client operations such as browser back, directly accessing specified URL etc. Therefore, the exception needs to be handled as client error.

Following are the settings to handle `HttpSessionRequiredException` as client error.

- spring-mvc.xml

```
<bean class="org.terasoluna.gfw.web.exception.SystemExceptionResolver">
    <property name="exceptionCodeResolver" ref="exceptionCodeResolver" />
    <!-- ... -->
    <property name="exceptionMappings">
        <map>
            <!-- ... -->
            <entry key="HttpSessionRequiredException "
                   value="common/error/operationError" /> <!-- (5) -->
        </map>
    </property>
    <property name="statusCodes">
        <map>
            <!-- ... -->
            <entry key="common/error/operationError" value="400" /> <!-- (6) -
↪->
        </map>
    </property>
    <!-- ... -->
</bean>
```

Sr. No.	Description
(5)	Add the exception handling definition of `HttpSessionRequiredException` to `exceptionMappings` of `SystemExceptionResolver` provided by common library. In the above example, `/WEB-INF/views/common/error/operationError.jsp` is specified as the transition destination at the time of exception.
(6)	In `statusCodes` of `SystemExceptionResolver`, specify the HTTP response code that is generated when `HttpSessionRequiredException` occurs. In the above example, Bad Request (`400`) is specified as the HTTP response code at the time of exception.

- applicationContext.xml

```
<bean id="exceptionCodeResolver"
    class="org.terasoluna.gfw.common.exception.
↪SimpleMappingExceptionCodeResolver">
    <!-- Setting and Customization by project. -->
    <property name="exceptionMappings">
        <map>
            <!-- ... -->
            <entry key="HttpSessionRequiredException" value="w.xx.0003" /> <!-
↪- (7) -->
        </map>
```

```
    </property>
    <property name="defaultExceptionCode" value="e.xx.0001" /> <!-- (8) -->
</bean>
```

Sr. No.	Description
(7)	Add the exception handling definition of `HttpSessionRequiredException` to `exceptionMappings` of `SimpleMappingExceptionCodeResolver` provided by common library. In the above example, `"w.xx.0003"` is specified as the exception code at the time of exception. When this setting is not added, default exception code is output to log.
(8)	Default exception code at the time of exception.

Deleting the object stored in session

To delete the object stored in session using `@SessionAttributes`, call setComplete method of `org.springframework.web.bind.support.SessionStatus` from the handler method of Controller.

On calling setComplete method of `SessionStatus` object, the object specified in attribute value of `@SessionAttributes` annotation, is deleted from session.

Note: Time when the object is deleted from session

By calling setComplete method of `SessionStatus` object, the object specified in attribute value of `@SessionAttributes` annotation is deleted from session. However, the actual time when the object is deleted is different than when setComplete method is called.

setComplete method of `SessionStatus` object changes only the internal flag, whereas, the actual deletion of object is carried out by the framework, after the handler method of Controller is completed.

Note: Referring object from View (JSP)

Object is deleted from session by calling the setComplete method of `SessionStatus` object. However, it can be referred from View (JSP) as the same object remains in `Model` object.

The objects stored in session need to be deleted from the following 3 locations.

- Request to display completion screen. **(Mandatory)**

  Delete unnecessary objects, since the objects stored in session are not accessed once completion screen is displayed.

> **Warning: Reason for deletion**
>
> The objects stored in session are outside the scope of garbage collection process. As a result, if unnecessary objects are not deleted, it leads to memory exhaustion. Moreover, storing unnecessary objects in session results in session swap-out which is a heavy process and may affect the overall application performance.

- Request to stop the consecutive screen operations. **(Mandatory)**

  Events such as "Go Back to Menu" or "Cancel" etc. to stop the consecutive screen operations, do not access the objects stored in session. Therefore, unnecessary objects should be deleted from session.

- Request for initial display of input screen. (Optional)

> **Warning: Reason for deletion**
>
> When the browser or tab is closed during screen operations, information entered during input, remains in the form object stored in session. As a result, this information is displayed on screen if it is not deleted at the time of initial display. However, in cases where it is alright to display this information on screen, it is not mandatory to delete the information using request for initial display.

Example given below illustrates implementation of object deletion by using a request to display completion screen.

```
// (1)
@RequestMapping(value = "save", method = RequestMethod.POST)
public String save(@ModelAttribute @Validated({ Wizard1.class,
        Wizard2.class, Wizard3.class }) WizardForm form,
        BindingResult result, Entity entity,
        RedirectAttributes redirectAttributes) {
    // ...
    return "redirect:/wizard/save?complete"; // (2)
}

// (3)
@RequestMapping(value = "save", params = "complete", method = RequestMethod.
↪GET)
```

```
public String saveComplete(SessionStatus sessionStatus) {
    sessionStatus.setComplete(); // (4)
    return "wizard/complete";
}
```

Sr. No.	Description
(1)	Handler method to perform update process.
(2)	Redirect to request (3), in order to display completion screen.
(3)	Handler method to display completion screen.
(4)	Call setComplete method of `SessionStatus` object and delete the object from session. Display process for View (JSP) is not affected directly as the same object remains in `Model` object.

Example given below illustrates implementation of object deletion by using a request to stop consecutive operations.

```
// (1)
@RequestMapping(value = "save", params = "cancel", method = RequestMethod.
↪POST)
public String saveCancel(SessionStatus sessionStatus) {
    sessionStatus.setComplete(); // (2)
    return "redirect:/wizard/menu"; // (3)
}
```

Sr. No.	Description
(1)	Handler method to stop the consecutive screen operations.
(2)	Call setComplete method of `SessionStatus` object and delete the object from session.
(3)	In the above example, user is redirected to Menu Screen.

Example given below illustrates implementation of object deletion by using a request for initial display of input screen.

```
// (1)
@RequestMapping(value = "create", method = RequestMethod.GET)
public String initializeCreateWizardForm(SessionStatus sessionStatus) {
    sessionStatus.setComplete();              // (2)
    return "redirect:/wizard/create?form1";   // (3)
}

// (4)
@RequestMapping(value = "create", params = "form1")
public String createForm1() {
    return "wizard/form1";
}
```

Sr. No.	Description
(1)	Handler method for initial display of input screen.
(2)	Call setComplete method of `SessionStatus` object.
(3)	Redirect to request (4), that displays input screen. Object is deleted from session by calling the setComplete method of `SessionStatus` object. However, as the same object remains in `Model` object, if View (JSP) is called directly, the information being entered is displayed. Therefore, **It should be redirected to the request for displaying input screen after deleting from session.**
(4)	Handler method to display input screen.

Process implementation using @SessionAttributes

For specific implementation, refer to *Example of screen transition implementation using @SessionAttributes in wizard format* of Appendix.

Using session-scoped bean of Spring Framework

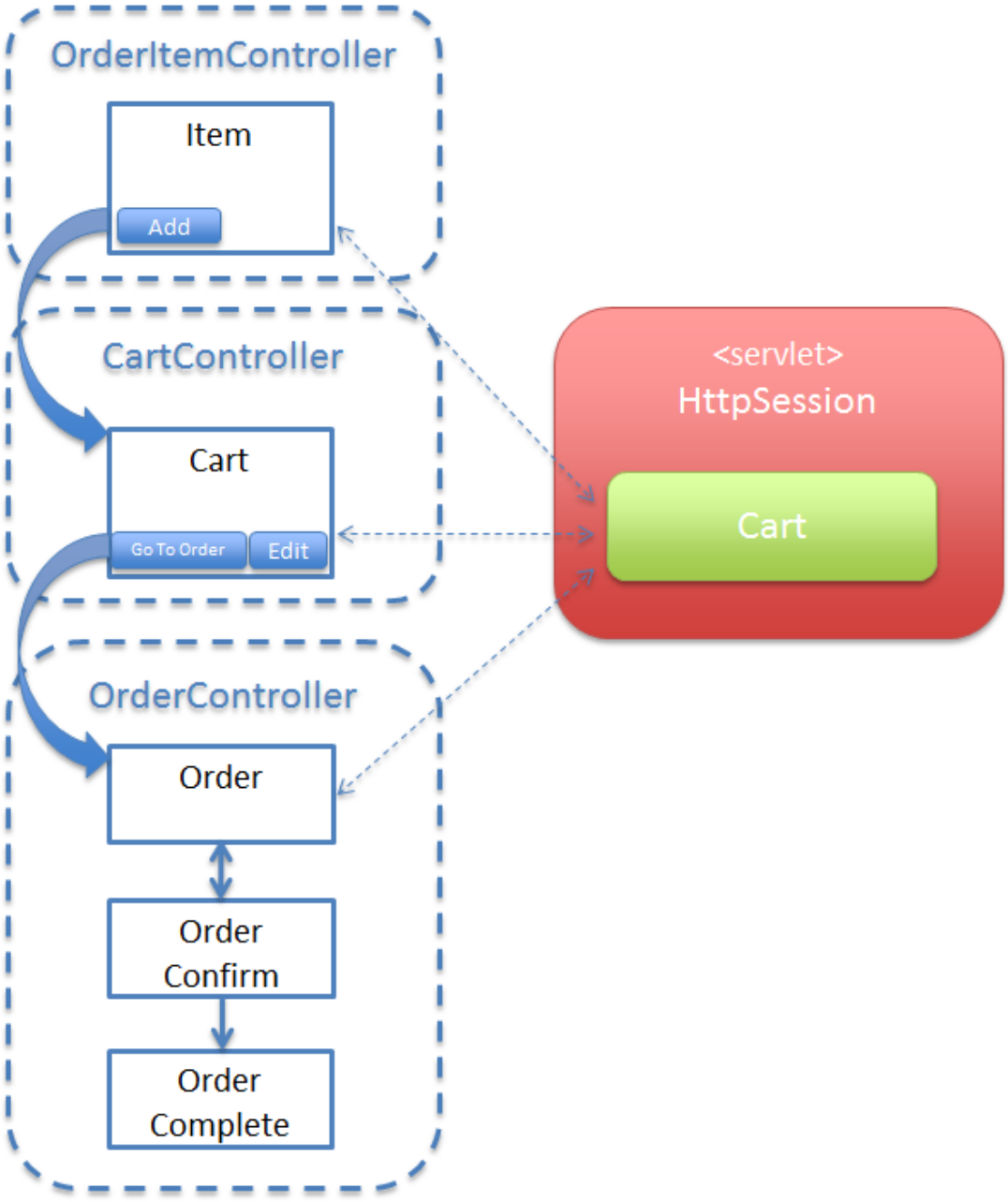
It is recommended to use the session-scoped bean of Spring Framework in order to
share the data in screen transition across multiple Controllers.

Bean definition of session scope

Define session-scoped bean of Spring Framework.

There are 2 methods to define the session-scoped bean.

- Define bean using component-scan.
- Define bean in Bean definition file (XML).



How to use component-scan is shown below.

- Class

```
@Component
@Scope(value = "session", proxyMode = ScopedProxyMode.TARGET_CLASS) // (1)
public class SessionCart implements Serializable {

    private static final long serialVersionUID = 1L;

    private Cart cart;
```

```
    public Cart getCart() {
        if (cart == null) {
            cart = new Cart();
        }
        return cart;
    }

    public void setCart(Cart cart) {
        this.cart = cart;
    }

    public void clearCart() { // (2)
        cart.clearCart();
    }
}
```

Sr. No.	Description
(1)	Set Bean scope to `"session"`. Also, it be enabled the scoped-proxy by specifying a `ScopedProxyMode.TARGET_CLASS` in `proxyMode` attribute.
(2)	Create a method to clear the cart content (delete products from the cart) while order has been finished.

Note: To define Entity class to be handled using JPA as session scope ban, it is recommended to provide a wrapper class instead of defining it directly as session-scoped bean.

If Entity class to be handled using JPA is defined as session-scoped bean, it cannot be handled directly using the API of JPA (throws error if handled directly). Therefore, a process to convert it to Entity object that can be handled in JPA is required.

In the above example, JPA Entity class called `Cart` is wrapped in a wrapper class called `SessionCart` and set as session-scoped bean. With this, the process to convert it to Entity object that can be handled in JPA is no longer required; hence the process to be performed in Controller becomes simple.

Note: Reason for enabling scoped-proxy

scoped-proxy needs to be enabled to inject session-scoped bean in singleton scope Controller.

- spring-mvc.xml

```
<context:component-scan base-package="xxx.yyy.zzz.app" /> // (2)
```

Sr. No.	Description
(2)	Scan components using the <context:component-scan> element. Specify a base package to scan components in the base-package attribute.

How to define it in Bean definition file (XML), is shown below.

- JavaBean

```
<beans:bean id="sessionCart" class="xxx.yyy.zzz.app.SessionCart"
            scope="session"> <!-- (3) -->
    <aop:scoped-proxy /> <!-- (4) -->
</beans:bean>
```

Sr. No.	Description
(3)	Set Bean scope to "session".
(4)	Specify <aop:scoped-proxy /> element and enable scoped-proxy.

Using session-scoped bean

To store and fetch objects in session using session-scoped bean,

inject the session-scoped bean to Controller.

```
@Inject
SessionCart sessionCart; // (1)

@RequestMapping(value = "add")
public String addCart(@Validated ItemForm form, BindingResult result) {
    if (result.hasErrors()) {
        return "item/item";
    }
    CartItem cartItem = beanMapper.map(form, CartItem.class);
```

```
    Cart addedCart = cartService.addCartItem(sessionCart.getCart(), // (2)
            cartItem);
    sessionCart.setCart(addedCart); // (3)
    return "redirect:/cart";
}
```

Sr. No.	Description
(1)	Inject session-scoped bean to Controller.
(2)	On calling the method of session-scoped bean, object stored in session is returned. When there is no object stored in session, newly created object is returned as well as stored in session. In the above example, Service method is called to check inventory etc. before adding to cart.
(3)	In the above example, `Cart` object passed as an argument of addCartItem method of `CartService` and the `Cart` object returned with value, may form a separate instance. Therefore, the returned `Cart` object is set to session-scoped bean.

Note: How to refer session-scoped Bean from View(JSP)

A session-scoped Bean can be referred from JSP even when Bean is not added to `Model` object in Controller by using SpEL(Spring Expression Language) formula.

```
<spring:eval var="cart" expression="@sessionCart.cart" />      <%-- (1) --%>

<%-- omitted --%>

<c:forEach var="item" items="${cart.cartItems}">      <%-- (2) --%>
    <tr>
        <td>${f:h(item.id)}</td>
        <td>${f:h(item.itemCode)}</td>
        <td>${f:h(item.quantity)}</td>
    </tr>
</c:forEach>
```

Sr. No.	Description
(1)	Refer session-scoped Bean.
(2)	Display session-scoped Bean.

Deleting objects stored in session

If you want to remove object from session which is not required any more, reset the Bean field from session scope.

Note: The IoC Container destroys session-scoped beans when a session expires.

Since the IoC Container manages the lifecycle of session-scoped beans, avoid deleting them explicitly.

```
@Controller
@RequestMapping("order")
public class OrderController {

    @Inject
    SessionCart sessionCart;  // (1)

    // ...

    @RequestMapping(method = RequestMethod.POST)
    public String order() {
        // ...
        return "redirect:/order?complete";
    }

    @RequestMapping(params = "complete", method = RequestMethod.GET)
    public String complete(Model model, SessionStatus sessionStatus) {
        sessionCart.clearCart(); // (2)
        return "order/complete";
    }

}
```

Sr. No.	Description
(1)	An injectable session-scoped bean.
(2)	delete all items which have already been ordered by initializing the state of the session-scoped bean.

Process implementation using session-scoped bean

For a more specific implementation, refer to *Example of screen transition across multiple Controllers using session-scoped bean.* of Appendix.

Debug log output of session operations

Class that outputs the operations performed for a session to debug log, is provided by common library.
The log output by this class is effective to check whether session operations are performed as expected.

For details on common library, refer to :ref:*logging_appendix_httpsessioneventlogginglistener*.

Using the JSP implicit object `sessionScope`

To use JSP implicit object `sessionScope` , the session attribute value of page directive needs to be set to `true`.
It is set as `false` in the `include.jsp`, provided by blank project.

`include.jsp` is stored in the `src/main/webapp/WEB-INF/views/common` directory.

- `include.jsp`

```
<%@ page session="true"%>      <%-- (1) --%>

<%-- omitted --%>
```

Sr. No.	Description
(1)	Set the session attribute value of page directive to `true`.

4.3.3 How to extend

Synchronizing requests in same session

It is recommended to synchronize the requests in the same session in order to use `@SessionAttributes` annotation or session-scoped bean.

If the requests are not synchronized, the object stored in session may be accessed at the same time, causing unexpected errors or operations.

For example, incorrect value may be set for the form object with completed input validation.

To prevent this, it is strongly recommended to set synchronizeOnSession of `org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter` to 'true' and synchronize the requests in the same session.

It can be implemented by creating BeanPostProcessor as follows and performing bean definition.

- Component

```
package com.example.app.config;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.BeansException;
import org.springframework.beans.factory.config.BeanPostProcessor;
import org.springframework.web.servlet.mvc.method.annotation.
↪RequestMappingHandlerAdapter;

public class EnableSynchronizeOnSessionPostProcessor
   implements BeanPostProcessor {
    private static final Logger logger = LoggerFactory
            .getLogger(EnableSynchronizeOnSessionPostProcessor.class);

    @Override
    public Object postProcessBeforeInitialization(Object bean, String␣
↪beanName)
        throws BeansException {
        // NO-OP
        return bean;
    }

    @Override
    public Object postProcessAfterInitialization(Object bean, String beanName)
        throws BeansException {
        if (bean instance of RequestMappingHandlerAdapter) {
            RequestMappingHandlerAdapter adapter =
```

```
                (RequestMappingHandlerAdapter) bean;
            logger.info("enable synchronizeOnSession => {}", adapter);
            adapter.setSynchronizeOnSession(true); // (1)
        }
        return bean;
    }

}
```

Sr. No.	Description
(1)	Requests in the same session can be synchronized by specifying `true` as an argument of setSynchronizeOnSession method of `org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandl`

- spring-mvc.xml

```
<bean class="com.example.app.config.EnableSynchronizeOnSessionPostProcessor" /
↪> <!-- (2) -->
```

Sr. No.	Description
(2)	Define bean for `BeanPostProcessor` created in (1).

4.3.4 Appendix

Example of screen transition implementation using `@SessionAttributes` in wizard format

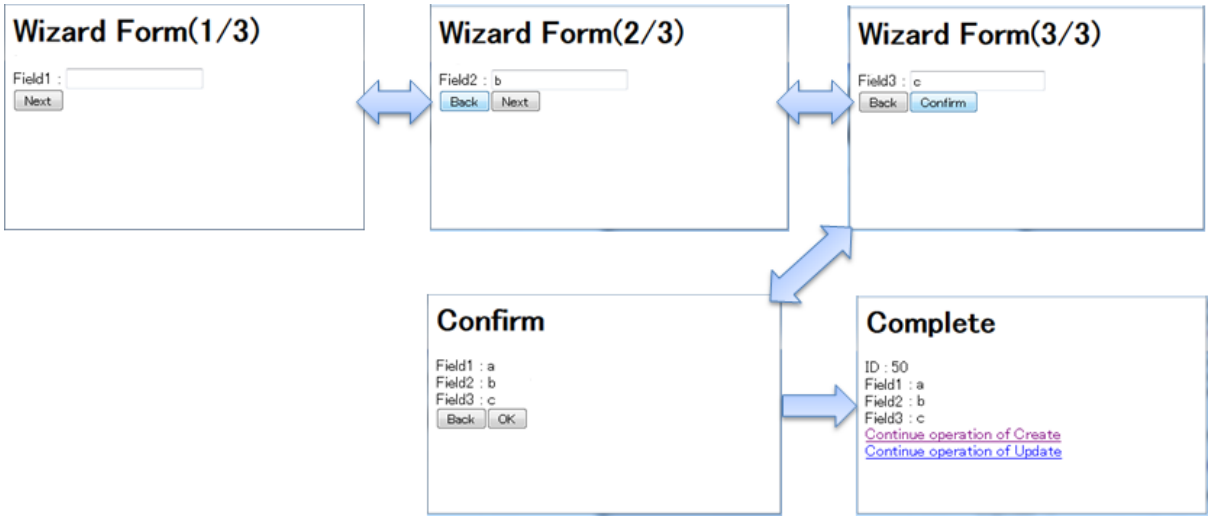
Screen transition implementation using `@SessionAttributes` annotation in wizard format, is explained below.

Process specifications are as follows:

- Provide screen for registering and updating entity.
- Input screen consists of 3 sub-screens wherein one field is entered on each sub-screen.
- Before saving (registering/updating), the entered values can be verified on confirmation screen.
- Input validation is performed at the time of screen transition. User is returned to input screen in case of error.

- Before saving (registering/updating), perform input validation again for all the input values and display the error screen notifying invalid operations.
- If all the input values are appropriately validated, save the input data to the database.

Basic screen transition is as follows:



Example of implementation is as follows:

- Form object

```
public class WizardForm implements Serializable {

    private static final long serialVersionUID = 1L;

    // (1)
    @NotEmpty(groups = { Wizard1.class })
    private String field1;

    // (2)
    @NotEmpty(groups = { Wizard2.class })
    private String field2;

    // (3)
    @NotEmpty(groups = { Wizard3.class })
    private String field3;

    // ...

    // (4)
    public static interface Wizard1 {
    }
```

```
    // (5)
    public static interface Wizard2 {
    }

    // (6)
    public static interface Wizard3 {
    }

}
```

Sr. No.	Description
(1)	Field to be entered on the first page of input screen.
(2)	Field to be entered on the second page of input screen.
(3)	Field to be entered on the third page of input screen.
(4)	Verification group interface to indicate that it is the field entered on the first page of input screen.
(5)	Verification group interface to indicate that it is the field entered on the second page of input screen.
(6)	Verification group interface to indicate that it is the field entered on the third page of input screen.

Note: Verification group

When performing input validation for screen transition, only the fields on corresponding page need to be validated. In Bean Validation, verification rules can be grouped by setting the interface or class representing the verification group. In this implementation, input validation for each screen is performed by providing verification group for each screen.

- Controller

```
@Controller
@RequestMapping("wizard")
@SessionAttributes(types = { WizardForm.class, Entity.class }) // (7)
public class WizardController {

    @Inject
    WizardService wizardService;

    @Inject
    Mapper beanMapper;
```

Sr. No.	Description
(7)	In the above example, form object (`WizardForm.class`) and entity (`Entity.class`) object are stored in session.

```
@ModelAttribute("wizardForm") // (8)
public WizardForm setUpWizardForm() {
    return new WizardForm();
}
```

Sr. No.	Description
(8)	In the above example, form object (`WizardForm`) to be stored in session is generated. To prevent creation of unnecessary objects, "value" attribute of `@ModelAttribute` annotation is specified.

```
// (9)
@RequestMapping(value = "create", method = RequestMethod.GET)
public String initializeCreateWizardForm(SessionStatus sessionStatus) {
    sessionStatus.setComplete();
    return "redirect:/wizard/create?form1";
}

// (10)
@RequestMapping(value = "create", params = "form1")
public String createForm1() {
    return "wizard/form1";
}
```

Sr. No.	Description
(9)	Handler method for initial display of input screen for registration. Objects for which operation is in process, may be stored in session; hence such objects are deleted by this handler method.
(10)	Handler method to display the first page of input screen for registration.

```
// (11)
@RequestMapping(value = "{id}/update", method = RequestMethod.GET)
public String initializeUpdateWizardForm(@PathVariable("id") Integer id,
        RedirectAttributes redirectAttributes, SessionStatus sessionStatus) {
    sessionStatus.setComplete();
    return "redirect:/wizard/{id}/update?form1";
}

// (12)
@RequestMapping(value = "{id}/update", params = "form1")
public String updateForm1(@PathVariable("id") Integer id, WizardForm form,
        Model model) {
    Entity loadedEntity = wizardService.getEntity(id);
    beanMapper.map(loadedEntity, form); // (13)
    model.addAttribute(loadedEntity); // (14)
    return "wizard/form1";
}
```

Sr. No.	Description
(11)	Handler method for initial display of input screen for update.
(12)	Handler method to display input screen for update (first page).
(13)	Set the fetched entity status in form object. In the above example, Bean mapper library called Dozer, is used.
(14)	Add the fetched entity to `Model` object and store in session. In the above example, it is stored in session with attribute name `"entity"`.

```
// (15)
@RequestMapping(value = "save", params = "form2", method = RequestMethod.POST)
public String saveForm2(@Validated(Wizard1.class) WizardForm form, // (16)
        BindingResult result) {
    if (result.hasErrors()) {
        return saveRedoForm1();
    }
    return "wizard/form2";
}

// (17)
@RequestMapping(value = "save", params = "form3", method = RequestMethod.POST)
public String saveForm3(@Validated(Wizard2.class) WizardForm form, // (18)
        BindingResult result) {
    if (result.hasErrors()) {
        return saveRedoForm2();
    }
    return "wizard/form3";
}

// (19)
@RequestMapping(value = "save", params = "confirm", method = RequestMethod.
↪POST)
public String saveConfirm(@Validated(Wizard3.class) WizardForm form, // (20)
        BindingResult result) {
    if (result.hasErrors()) {
        return saveRedoForm3();
    }
    return "wizard/confirm";
}
```

Sr. No.	Description
(15)	Handler method to display second page of input screen.
(16)	Specify the verification group (`Wizard1.class`) on the first page of input screen, in "value" attribute of `@Validated` annotation, so as to perform input validation of only the value entered in first page of input screen.
(17)	Handler method to display the third page of input screen.
(18)	Specify the verification group (`Wizard2.class`) on the second page of input screen, in "value" attribute of `@Validated` annotation, so as to perform input validation of only the value entered in second page of input screen.
(19)	Handler method to display confirmation screen.
(20)	Specify the verification group (`Wizard3.class`) on the third page of input screen, in "value" attribute of `@Validated` annotation, so as to perform input validation of only the value entered in third page of input screen.

```
// (21)
@RequestMapping(value = "save", method = RequestMethod.POST)
public String save(@ModelAttribute @Validated({ Wizard1.class,
        Wizard2.class, Wizard3.class }) WizardForm form, // (22)
        BindingResult result,
        Entity entity, // (23)
        RedirectAttributes redirectAttributes) {
    if (result.hasErrors()) {
        throw new InvalidRequestException(result); // (24)
    }

    beanMapper.map(form, entity);

    entity = wizardService.saveEntity(entity); // (25)

    redirectAttributes.addFlashAttribute(entity); // (26)
```

```
    return "redirect:/wizard/save?complete";
}

// (27)
@RequestMapping(value = "save", params = "complete", method = RequestMethod.
↪GET)
public String saveComplete(SessionStatus sessionStatus) {
    sessionStatus.setComplete();
    return "wizard/complete";
}
```

Sr. No.	Description
(21)	Handler method to execute save process.
(22)	Specify the verification group interface (`Wizard1.class`, `Wizard2.class`, `Wizard3.class`) of each input screen, in the "value" attribute of `@Validated` annotation, to check all the values entered on input screen.
(23)	Fetch the `Entity.class` object to be saved. For registration process, newly created object is fetched and for update process, the object stored in session by process (14), is fetched.
(24)	As error does not occur when screen transition is performed using the buttons provided by the application, `InvalidRequestException` is thrown when an invalid operation is performed. Further, exception class `InvalidRequestException` needs to be created separately, as it is not provided by common library.
(25)	Save the `Entity.class` object reflecting the input value.
(26)	`Entity.class` object saved by handler method of redirect destination is stored in Flash scope so that it can be referred.
(27)	Handler method to display completion screen.

```
    // (28)
    @RequestMapping(value = "save", params = "redoForm1")
    public String saveRedoForm1() {
        return "wizard/form1";
    }

    // (29)
    @RequestMapping(value = "save", params = "redoForm2")
```

```
    public String saveRedoForm2() {
        return "wizard/form2";
    }

    // (30)
    @RequestMapping(value = "save", params = "redoForm3")
    public String saveRedoForm3() {
        return "wizard/form3";
    }

}
```

Sr. No.	Description
(28)	Handler method to re-display first page of input screen.
(29)	Handler method to re-display second page of input screen.
(30)	Handler method to re-display third page of input screen.

- Complete Controller source code

```
@Controller
@RequestMapping("wizard")
@SessionAttributes(types = { WizardForm.class, Entity.class })
// (7)
public class WizardController {

    @Inject
    EntityService wizardService;

    @Inject
    Mapper beanMapper;

    @ModelAttribute("wizardForm")
    // (8)
    public WizardForm setUpWizardForm() {
        return new WizardForm();
    }

    // (9)
    @RequestMapping(value = "create", method = RequestMethod.GET)
    public String initializeCreateWizardForm(SessionStatus sessionStatus) {
        sessionStatus.setComplete();
```

```
        return "redirect:/wizard/create?form1";
    }

    // (10)
    @RequestMapping(value = "create", params = "form1")
    public String createForm1() {
        return "wizard/form1";
    }

    // (11)
    @RequestMapping(value = "{id}/update", method = RequestMethod.GET)
    public String initializeUpdateWizardForm(@PathVariable("id") Integer id,
            RedirectAttributes redirectAttributes, SessionStatus␣
↪sessionStatus) {
        sessionStatus.setComplete();
        return "redirect:/wizard/{id}/update?form1";
    }

    // (12)
    @RequestMapping(value = "{id}/update", params = "form1")
    public String updateForm1(@PathVariable("id") Integer id, WizardForm form,
            Model model) {
        Entity loadedEntity = wizardService.getEntity(id);
        beanMapper.map(loadedEntity, form); // (13)
        model.addAttribute(loadedEntity); // (14)
        return "wizard/form1";
    }

    // (15)
    @RequestMapping(value = "save", params = "form2", method = RequestMethod.
↪POST)
    public String saveForm2(@Validated(Wizard1.class) WizardForm form, // (16)
            BindingResult result) {
        if (result.hasErrors()) {
            return saveRedoForm1();
        }
        return "wizard/form2";
    }

    // (17)
    @RequestMapping(value = "save", params = "form3", method = RequestMethod.
↪POST)
    public String saveForm3(@Validated(Wizard2.class) WizardForm form, // (18)
            BindingResult result) {
        if (result.hasErrors()) {
            return saveRedoForm2();
        }
        return "wizard/form3";
    }
```

```
    // (19)
    @RequestMapping(value = "save", params = "confirm", method =␣
↪RequestMethod.POST)
    public String saveConfirm(@Validated(Wizard3.class) WizardForm form, //␣
↪(20)
            BindingResult result) {
        if (result.hasErrors()) {
            return saveRedoForm3();
        }
        return "wizard/confirm";
    }

    // (21)
    @RequestMapping(value = "save", method = RequestMethod.POST)
    public String save(@ModelAttribute @Validated({ Wizard1.class,
            Wizard2.class, Wizard3.class }) WizardForm form, // (22)
            BindingResult result, Entity entity, // (23)
            RedirectAttributes redirectAttributes) {
        if (result.hasErrors()) {
            throw new InvalidRequestException(result); // (24)
        }

        beanMapper.map(form, entity);

        entity = wizardService.saveEntity(entity); // (25)

        redirectAttributes.addFlashAttribute(entity); // (26)

        return "redirect:/wizard/save?complete";
    }

    // (27)
    @RequestMapping(value = "save", params = "complete", method =␣
↪RequestMethod.GET)
    public String saveComplete(SessionStatus sessionStatus) {
        sessionStatus.setComplete();
        return "wizard/complete";
    }

    // (28)
    @RequestMapping(value = "save", params = "redoForm1")
    public String saveRedoForm1() {
        return "wizard/form1";
    }

    // (29)
    @RequestMapping(value = "save", params = "redoForm2")
    public String saveRedoForm2() {
        return "wizard/form2";
    }
```

```
    // (30)
    @RequestMapping(value = "save", params = "redoForm3")
    public String saveRedoForm3() {
        return "wizard/form3";
    }

}
```

- First page of input screen (JSP)

```
<html>
<head>
<title>Wizard Form(1/3)</title>
</head>
<body>
    <h1>Wizard Form(1/3)</h1>
    <form:form action="${pageContext.request.contextPath}/wizard/save"
        modelAttribute="wizardForm">
        <form:label path="field1">Field1</form:label> :
        <form:input path="field1" />
        <form:errors path="field1" />
        <div>
            <form:button name="form2">Next</form:button>
        </div>
    </form:form>
</body>
</html>
```

- Second page of input screen (JSP)

```
<html>
<head>
<title>Wizard Form(2/3)</title>
</head>
<body>
    <h1>Wizard Form(2/3)</h1>
    <%-- (31) --%>
    <form:form action="${pageContext.request.contextPath}/wizard/save"
        modelAttribute="wizardForm">
        <form:label path="field2">Field2</form:label> :
        <form:input path="field2" />
        <form:errors path="field2" />
        <div>
            <form:button name="redoForm1">Back</form:button>
            <form:button name="form3">Next</form:button>
        </div>
    </form:form>
</body>
</html>
```

Sr. No.	Description
(31)	There is no need to hide the input screen fields of first page since the form object is stored in session.

- Third page of input screen (JSP)

```
<html>
<head>
<title>Wizard Form(3/3)</title>
</head>
<body>
    <h1>Wizard Form(3/3)</h1>
    <%-- (32) --%>
    <form:form action="${pageContext.request.contextPath}/wizard/save"
        modelAttribute="wizardForm">
        <form:label path="field3">Field3</form:label> :
        <form:input path="field3" />
        <form:errors path="field3" />
        <div>
            <form:button name="redoForm2">Back</form:button>
            <form:button name="confirm">Confirm</form:button>
        </div>
    </form:form>
</body>
</html>
```

Sr. No.	Description
(32)	There is no need to hide the input screen fields of first and second page since form object is stored in session.

- Confirmation screen (JSP)

```
<html>
<head>
<title>Confirm</title>
</head>
<body>
    <h1>Confirm</h1>
    <%-- (33) --%>
    <form:form action="${pageContext.request.contextPath}/wizard/save"
        modelAttribute="wizardForm">
        <div>
            Field1 : <c:out value="${wizardForm.field1}" />
```

```
        </div>
        <div>
            Field2 : <c:out value="${wizardForm.field2}" />
        </div>
        <div>
            Field3 : <c:out value="${wizardForm.field3}" />
        </div>
        <div>
            <form:button name="redoForm3">Back</form:button>
            <form:button>OK</form:button>
        </div>
    </form:form>
</body>
</html>
```

Sr. No.	Description
(33)	There is no need to hide input screen fields since form object is stored in session.

- Completion screen (JSP)

```
<html>
<head>
<title>Complete</title>
</head>
<body>
    <h1>Complete</h1>
    <div>
        <div>
            ID : ${f:h(entity.id)}
        </div>
        <div>
            Field1 : ${f:h(entity.field1)}
        </div>
        <div>
            Field2 : ${f:h(entity.field2)}
        </div>
        <div>
            Field3 : ${f:h(entity.field3)}
        </div>
    </div>
    <div>
        <a href="${pageContext.request.contextPath}/wizard/create">
            Continue operation of Create
        </a>
    </div>
    <div>
        <a href="${pageContext.request.contextPath}/wizard/${entity.id}/update
↪">
```

```
            Continue operation of Update
        </a>
    </div>
</body>
</html>
```

- spring-mvc.xml

```
<bean class="org.terasoluna.gfw.web.exception.SystemExceptionResolver">
    <property name="exceptionCodeResolver" ref="exceptionCodeResolver" />
    <!-- ... -->
    <property name="exceptionMappings">
        <map>
            <!-- ... -->
            <entry key="InvalidRequestException"
                   value="common/error/operationError" /> <!-- (34) -->
        </map>
    </property>
    <property name="statusCodes">
        <map>
            <!-- ... -->
            <entry key="common/error/operationError" value="400" /> <!-- (35)␣
↪-->
        </map>
    </property>
    <!-- ... -->
</bean>
```

Sr. No.	Description
(34)	Add exception handling definition of `InvalidRequestException` that notifies detection of invalid request when executing save process, in `exceptionMappings` of `SystemExceptionResolver`, provided by common library. In the above example, `/WEB-INF/views/common/error/operationError.jsp` is specified as the transition destination when exception occurs.
(35)	Specify the HTTP response code when `HttpSessionRequiredException` occurs in `statusCodes` of `SystemExceptionResolver`. In the above example, Bad Request (`400`) is specified as HTTP response code when exception occurs.

- applicationContext.xml

```
<bean id="exceptionCodeResolver"
    class="org.terasoluna.gfw.common.exception.
↪SimpleMappingExceptionCodeResolver">
    <!-- Setting and Customization by project. -->
    <property name="exceptionMappings">
        <map>
            <!-- ... -->
            <entry key="InvalidRequestException" value="w.xx.0004" /> <!--␣
↪(36) -->
        </map>
    </property>
    <property name="defaultExceptionCode" value="e.xx.0001" /> <!-- (37) -->
</bean>
```

Sr. No.	Description
(36)	Add exception handling definition of `InvalidRequestException` to `exceptionMappings` of `SimpleMappingExceptionCodeResolver` provided by common library. In the above example, `"w.xx.0004"` is specified as the exception code when an exception occurs. When this setting is not added, default exception code is output to log.
(37)	Default exception code in case of an exception.

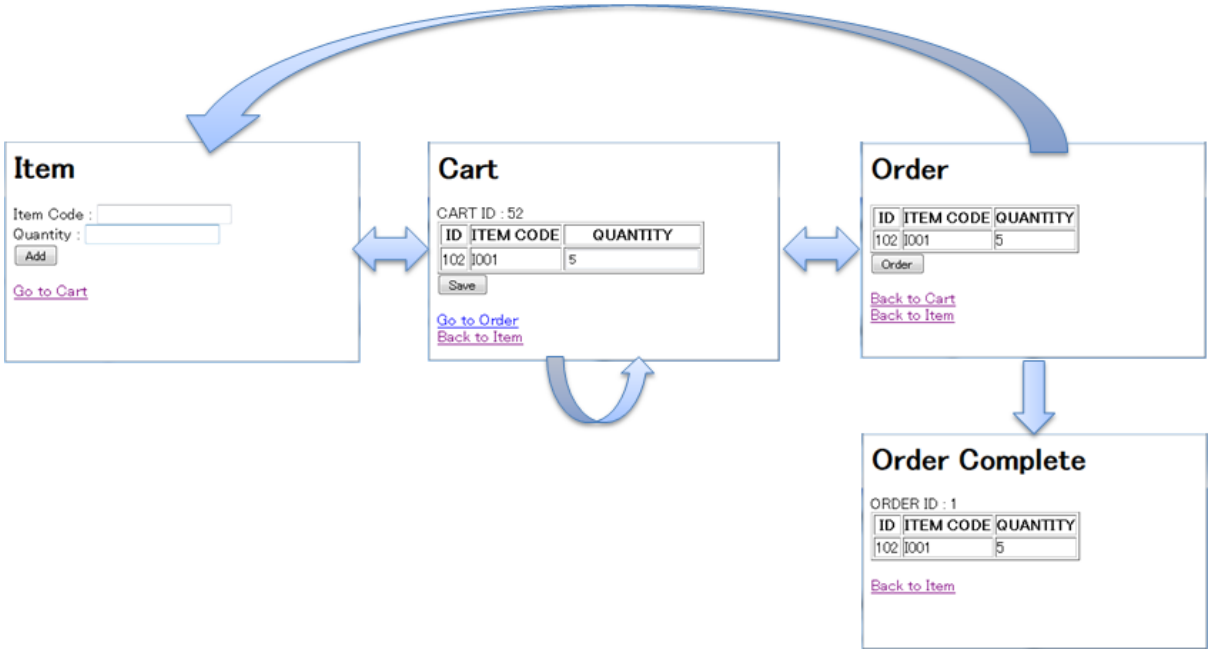
Example of screen transition across multiple Controllers using session-scoped bean.

Implementation using session-scoped bean is explained with example of process of performing screen transitions across multiple Controllers.

Process specifications are as follows:

- Provide a process to add products to cart.
- Provide a process to change the quantity of products added to the cart.
- Provide a process to order the products stored in cart.
- Above 3 processes are provided as independent functions and should be set in separate Controllers (ItemController, CartController, OrderController).
- Store the cart in session as it is shared by the above 3 processes.
- Display the cart screen when products are added to cart.

Screen transition should be as follows:



Implementation is as follows:

- JavaBean to be defined as session-scoped bean.

```
@Component
@Scope(value = "session", proxyMode = ScopedProxyMode.TARGET_CLASS)
public class SessionCart implements Serializable {

    private static final long serialVersionUID = 1L;

    private Cart cart; // (1)

    public Cart getCart() {
        if (cart == null) {
            cart = new Cart();
        }
        return cart;
    }

    public void setCart(Cart cart) {
        this.cart = cart;
    }

    public void clearCart() { // (2)
        cart.clearCart();
    }
}
```

Sr. No.	Description
(1)	Wrap the Entity (Domain object) called, `Cart`.
(2)	Make cart empty by removing all product objects from `cart`which are added in the cart.

- ItemController

```
@Controller
@RequestMapping("item")
public class ItemController {

    @Inject
    SessionCart sessionCart;

    @Inject
    CartService cartService;

    @Inject
    Mapper beanMapper;

    @ModelAttribute
    public ItemForm setUpItemForm() {
        return new ItemForm();
    }

    // (3)
    @RequestMapping
    public String view(Model model) {
        return "item/item";
    }

    // (4)
    @RequestMapping(value = "add")
    public String addCart(@Validated ItemForm form, BindingResult result) {
        if (result.hasErrors()) {
            return "item/item";
        }
        CartItem cartItem = beanMapper.map(form, CartItem.class);
        Cart cart = cartService.addCartItem(sessionCart.getCart(), // (5)
                cartItem);
        sessionCart.setCart(cart); // (6)
        return "redirect:/cart"; // (7)
    }
}
```

Sr. No.	Description
(3)	Handler method to display product screen.
(4)	Handler method to add specified products to cart.
(5)	Pass `Cart` object stored in the session in Service method.
(6)	Reflect `Cart` object returned by Service method to session-scoped bean. By reflecting in session-scoped Bean, `Cart` object is stored in the session.
(7)	Redirect to the request to display cart screen after adding products to cart. **When transiting to a separate Controller screen, instead of calling View (JSP) directly, it is recommended to redirect to the request for displaying the screen.**

- CartController

```
@Controller
@RequestMapping("cart")
public class CartController {

    @Inject
    SessionCart sessionCart;

    @Inject
    CartService cartService;

    @Inject
    Mapper beanMapper;

    @ModelAttribute
    public CartForm setUpCartForm() {
        return new CartForm();
    }

    // (8)
    @RequestMapping
    public String cart(CartForm form) {
        beanMapper.map(sessionCart.getCart(), form);
```

```
        return "cart/cart";
    }

    // (9)
    @RequestMapping(params = "edit", method = RequestMethod.POST)
    public String edit(@Validated CartForm form, BindingResult result,
            Model model) {
        if (result.hasErrors()) {
            return "cart/cart";
        }

        Cart cart = sessionCart.getCart();
        Iterator<CartItemForm> itemForm = form.getCartItems().iterator();
        for (CartItem item : cart.getCartItems()) {
            beanMapper.map(itemForm.next(), item);
        }

        cart = cartService.saveCart(cart);
        sessionCart.setCart(cart); // (10)

        return "redirect:/cart"; // (11)
    }


}
```

Sr. No.	Description
(8)	Handler method to display cart screen (quantity change screen).
(9)	Handler method to change quantity.
(10)	Reflect `Cart` object returned by Service method, to session-scoped bean. By reflecting in session-scoped bean, it is reflected in the session.
(11)	Redirect to the request to display cart screen (quantity change screen) after changing quantity. **In case of Update process, instead of calling View (JSP) directly, it is recommended to redirect to the request for displaying the screen.**

- OrderController

```
@Controller
@RequestMapping("order")
@SessionAttributes("scopedTarget.sessionCart")
public class OrderController {

    @Inject
    SessionCart sessionCart;

    @ModelAttribute
    public OrderForm setUpOrderForm() {
        return new OrderForm();
    }

    // (12)
    @RequestMapping
    public String view() {
        return "order/order";
    }

    // (13)
    @RequestMapping(method = RequestMethod.POST)
    public String order() {
        // ...
        return "redirect:/order?complete";
    }

    // (14)
    @RequestMapping(params = "complete", method = RequestMethod.GET)
    public String complete(Model model, SessionStatus sessionStatus) {
        sessionCart.clearCart();
        return "order/complete";
    }

}
```

Sr. No.	Description
(12)	Handler method to display Order screen.
(13)	Handler method to place an Order.
(14)	Handler method to display Order Completion screen.

- Product screen (JSP)

```
<html>
<head>
<title>Item</title>
</head>
<body>
    <h1>Item</h1>
    <form:form action="${pageContext.request.contextPath}/item/add"
        modelAttribute="itemForm">
        <form:label path="itemCode">Item Code</form:label> :
        <form:input path="itemCode" />
        <form:errors path="quantity" />
        <br>
        <form:label path="quantity">Quantity</form:label> :
        <form:input path="quantity" />
        <form:errors path="quantity" />
        <div>
            <%-- (15) --%>
            <form:button>Add</form:button>
        </div>
    </form:form>
    <div>
        <a href="${pageContext.request.contextPath}/cart">Go to Cart</a>
    </div>
</body>
</html>
```

Sr. No.	Description
(15)	Button to add a product.

- Cart screen (JSP)

```
<html>
<head>
<title>Cart</title>
</head>
<body>
    <%-- (16) --%>
    <spring:eval var="cart" experssion="@sessionCart.cart" />
    <h1>Cart</h1>
    <c:choose>
        <c:when test="${ empty cart.cartItems }">
            <div>Cart is empty.</div>
        </c:when>
        <c:otherwise>
            CART ID :
            ${f:h(cart.id)}
            <form:form modelAttribute="cartForm">
```

```
                <table border="1">
                    <thead>
                        <tr>
                            <th>ID</th>
                            <th>ITEM CODE</th>
                            <th>QUANTITY</th>
                        </tr>
                    </thead>
                    <tbody>
                        <c:forEach var="item"
                            items="${cart.cartItems}"
                            varStatus="rowStatus">
                            <tr>
                                <td>${f:h(item.id)}</td>
                                <td>${f:h(item.itemCode)}</td>
                                <td>
                                    <form:input
                                        path="cartItems[${rowStatus.index}].
↪quantity" />
                                    <form:errors
                                        path="cartItems[${rowStatus.index}].
↪quantity" />
                                </td>
                            </tr>
                        </c:forEach>
                    </tbody>
                </table>
                <%-- (17) --%>
                <form:button name="edit">Save</form:button>
            </form:form>
        </c:otherwise>
    </c:choose>
    <c:if test="${ not empty cart.cartItems }">
        <div>
            <%-- (18) --%>
            <a href="${pageContext.request.contextPath}/order">Go to Order</a>
        </div>
    </c:if>
    <div>
        <a href="${pageContext.request.contextPath}/item">Back to Item</a>
    </div>
</body>
</html>
```

Sr. No.	Description
(16)	Refer session-scoped Bean using SpEL formula.
(17)	Button to update quantity.
(18)	Link to display Order screen.

- Order screen (JSP)

```
<html>
<head>
<title>Order</title>
</head>
<body>
    <spring:eval var="cart" experssion="@sessionCart.cart" />
    <h1>Order</h1>
    <table border="1">
        <thead>
            <tr>
                <th>ID</th>
                <th>ITEM CODE</th>
                <th>QUANTITY</th>
            </tr>
        </thead>
        <tbody>
            <c:forEach var="item" items="${cart.cartItems}"
                varStatus="rowStatus">
                <tr>
                    <td>${f:h(item.id)}</td>
                    <td>${f:h(item.itemCode)}</td>
                    <td>${f:h(item.quantity)}</td>
                </tr>
            </c:forEach>
        </tbody>
    </table>
    <form:form modelAttribute="orderForm">
        <%-- (19) --%>
        <form:button>Order</form:button>
    </form:form>
    <div>
        <a href="${pageContext.request.contextPath}/cart">Back to Cart</a>
    </div>
    <div>
        <a href="${pageContext.request.contextPath}/item">Back to Item</a>
```

```
    </div>
</body>
</html>
```

Sr. No.	Description
(19)	Button to place an order.

- Order Completion screen (JSP)

```
<html>
<head>
<title>Order Complete</title>
</head>
<body>
    <h1>Order Complete</h1>
    ORDER ID :
    ${f:h(order.id)}
    <table border="1">
        <thead>
            <tr>
                <th>ID</th>
                <th>ITEM CODE</th>
                <th>QUANTITY</th>
            </tr>
        </thead>
        <tbody>
            <c:forEach var="item" items="${order.orderItems}"
                varStatus="rowStatus">
                <tr>
                    <td>${f:h(item.id)}</td>
                    <td>${f:h(item.itemCode)}</td>
                    <td>${f:h(item.quantity)}</td>
                </tr>
            </c:forEach>
        </tbody>
    </table>
    <br>
    <div>
        <a href="${pageContext.request.contextPath}/item">Back to Item</a>
    </div>
</body>
</html>
```

4.4 Pagination

4.4.1 Overview

This chapter describes the Pagination functionality wherein the data matching the search criteria is divided into pages and displayed.

It is recommended to use pagination functionality when a large amount of data matches the search criteria.
Retrieving and displaying a large amount of data at a time on screen may lead to following problems.

- Memory exhaustion at server side

  `java.lang.OutOfMemoryError` occurs when multiple requests are executed simultaneously.

- Network load

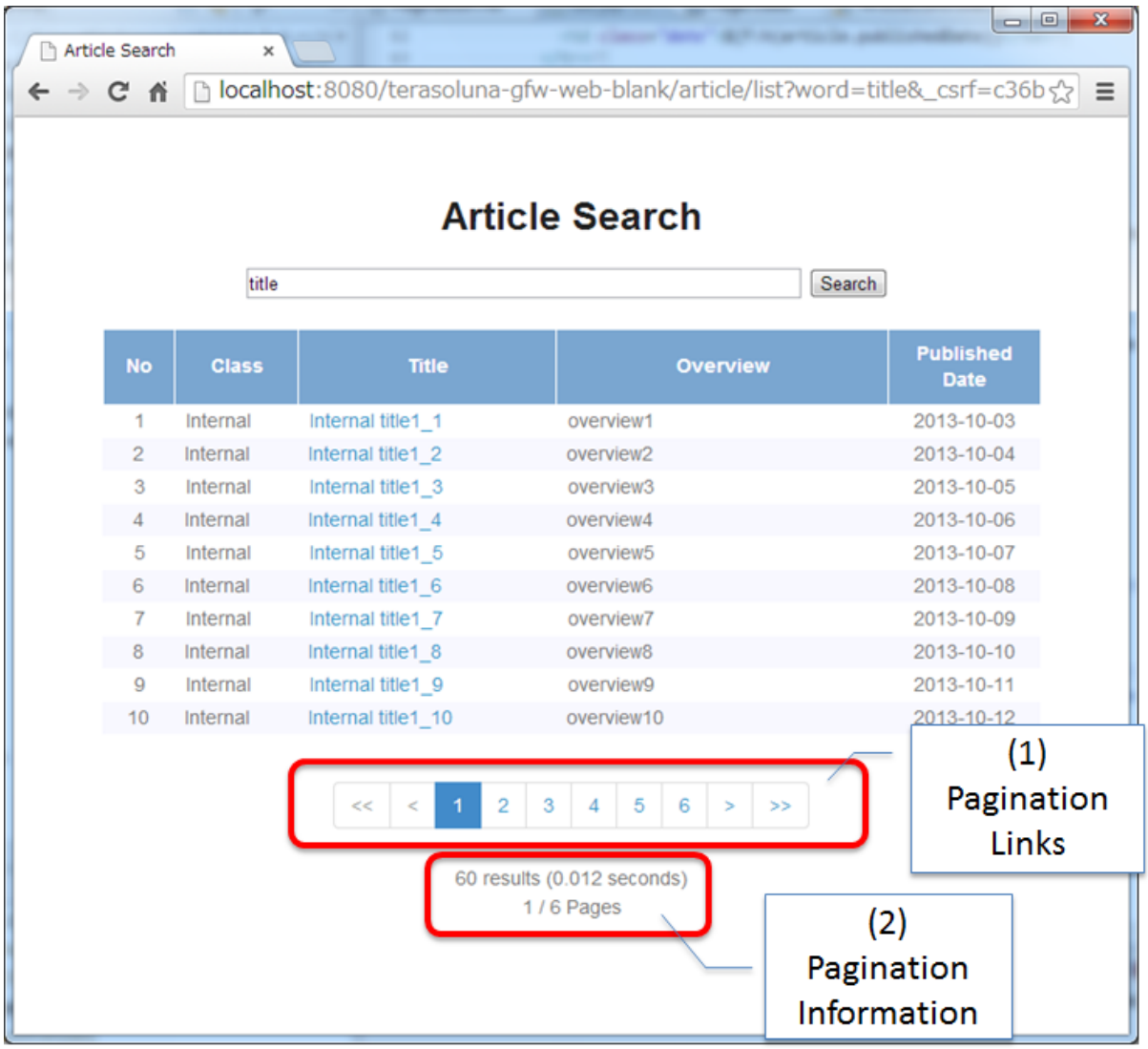
  Transferring unnecessary data over the network results in increased network load and thereby may affect the response time of the entire system.

- Delay in response on screen

  Server process, network traffic process and client rendering process take time to handle a large amount of data; hence the response on screen may get delayed.

Display of list screen at the time of dividing into pages

When a page is divided using pagination, the screen looks as follows:



Sr. No.	Description
(1)	Display the link to navigate to various pages. On clicking link, send a request to display the corresponding page. JSP tag library to display this area is provided as common library.
(2)	Display the information related to pagination (total records, total pages and number of displayed pages etc.). Tag library to display this area does not exist; hence it should be implemented separately as JSP processing.

Page search

For implementing pagination, it is essential to first implement the server-side search processing to make page searching possible.

It is assumed in this guideline that the mechanism provided by Spring Data is used for page search at server side.

Page search functionality of Spring Data

Page search functionality provided by Spring Data is as follows:

Sr. No.	Description
1	Extract the information required for page search (location of page to be searched, number of records to be fetched and sort condition) from request parameter and pass the extracted information as objects of `org.springframework.data.domain.Pageable` to the argument of Controller. This functionality is provided as `org.springframework.data.web.PageableHandlerMethodArgumentResolver` class and is enabled by adding to `<mvc:argument-resolvers>` element of `spring-mvc.xml` . For request parameters, refer to "*Note column*".
2	Save the page information (total records, data of corresponding page, location of page to be searched, number of records to be fetched and sort condition). This functionality is provided as `org.springframework.data.domain.Page` interface and `org.springframework.data.domain.PageImpl` is provided as default implementation class. **As per specifications, it fetches the required data from Page object in JSP tag library to output pagination link provided by common library.**
3	When Spring Data JPA is used for database access, the information of corresponding page is returned as `Page` object by specifying `Pageable` object as an argument of Repository Query method. All the processes such as executing SQL to fetch total records, adding sort condition and extracting data matching the corresponding page are carried out automatically. When MyBatis is used for database access, the process that is automatically carried out in Spring Data JPA needs to be implemented in Java (Service) and SQL mapping file.

Note: Request parameters for page search

Request parameters for page search provided by Spring Data are as follows:

Sr. No.	Parameter name	Description
1.	page	Request parameter to specify the location of page to be searched Specify value greater than or equal to 0. As per default setting, page location starts from `0` (zero). Hence, specify `0` (zero) to fetch the data of the first page and `1` (one) to fetch the data of the second page.
2.	size	Request parameter to specify the count of fetched records. Specify value greater than or equal to 1. When value specified is greater than the value in `maxPageSize` of `PageableHandlerMethodArgumentResolver`, `maxPageSize` value becomes `size` value.
3.	sort	Parameter to specify sort condition (multiple parameters can be specified). Specify the value in `"{sort item name(,sort order)}"` format. Specify either `"ASC"` or `"DESC"` as sort order. When nothing is specified, `"ASC"` is used. Multiple item names can be specified using `","` separator. For example, when `"sort=lastModifiedDate,id,DESC&sort=subId"` is specified as query string, Order By clause `"ORDER BY lastModifiedDate DESC,id DESC,subId ASC"` is added to the query.

Warning: Operations at the time of specifying "size=0" in spring-data-commons 1.6.1.RELEASE

spring-data-commons 1.6.1.RELEASE having terasoluna-gfw-common 1.0.0.RELEASE has a bug wherein if `"size=0"` is specified, all the records matching the specified condition are fetched. As a result, `java.lang.OutOfMemoryError` may occur when a large amount of records are fetched.

This problem is handled using JIRA DATACMNS-377 of Spring Data Commons and is being resolved in spring-data-commons 1.6.3.RELEASE. Post modification, if `"size<=0"` is specified, the default value when size parameter is omitted will be applied.

In cases where terasoluna-gfw-common 1.0.0.RELEASE is used, the version should be upgraded to terasoluna-gfw-common 1.0.1.RELEASE or higher version.

Warning: About operations when invalid values are specified in request parameters of spring-data-commons 1.6.1.RELEASE

spring-data-commons 1.6.1.RELEASE having terasoluna-gfw-common 1.0.0.RELEASE has a bug wherein if an invalid value is specified in request parameters for page search (page, size, sort etc.), `java.lang.IllegalArgumentException` or `java.lang.ArrayIndexOutOfBoundsException` occurs and SpringMVC settings when set to default values leads to system error (HTTP status code=500).

This problem is handled using JIRA DATACMNS-379 and DATACMNS-408 and is being resolved in spring-data-commons 1.6.3.RELEASE. Post modification, if invalid values are specified, the default value when parameters are omitted will be applied.

In cases where terasoluna-gfw-common 1.0.0.RELEASE is used, the version should be upgraded to terasoluna-gfw-common 1.0.1.RELEASE or higher version.

Note: Points to be noted due to the changes in API specifications of Spring Data Commons

In case of "terasoluna-gfw-common 5.0.0.RELEASE or later version" dependent spring-data-commons (1.9.1 RELEASE or later), there is a change in API specifications of interface for page search functionality (`org.springframework.data.domain.Page`) and class (`org.springframework.data.domain.PageImpl` and `org.springframework.data.domain.Sort.Order`).

Specifically,

- In `Page` interface and `PageImpl` class, `isFirst()` and `isLast()` methods are added in spring-data-commons 1.8.0.RELEASE, and `isFirstPage()` and `isLastPage()` methods are deleted from spring-data-commons 1.9.0.RELEASE.
- In `Sort.Order` class, `nullHandling` property is added in spring-data-commons 1.8.0.RELEASE.

If deleted API is used, there will be a compilation error and application will have to be modified. In addition, when using `Page` interface (`PageImpl` class) as resource object of REST API, that application may also need to be modified, as JSON and XML format get changed.

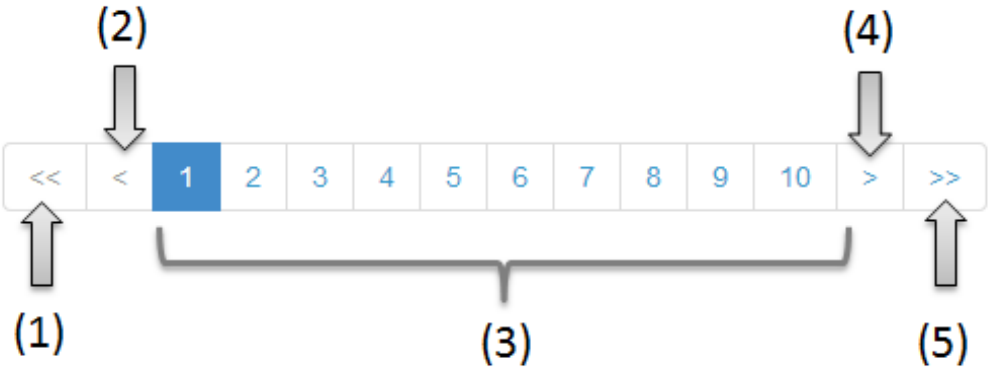
Display of pagination link

This section describes the pagination link which is output using JSP tag library of common library.

Style sheet to display pagination link is not provided from common library, hence it should be created in each project.
Bootstrap v3.0.0 style sheet is applied for the screens used in the explanation below.

Structure of pagination link

Pagination link consists of the following elements.



Sr. No.	Description
(1)	Link to navigate to the first page.
(2)	Link to navigate to the previous page.
(3)	Link to navigate to the specified page.
(4)	Link to navigate to the next page.
(5)	Link to navigate to the last page.

Pagination link has the following status.



Sr. No.	Description
(6)	Status indicating link where operations cannot be performed on the currently displayed page. The status is specifically “Link to navigate to the first page” and “Link to navigate to the previous page” when the first page is displayed and “Link to navigate to the next page” “Link to navigate to the last page” when the last page is displayed. This status is defined as `"disabled"` in the JSP tag library of common library.
(7)	Status indicating currently displayed page. This status is defined as `"active"` in the JSP tag library of common library.

HTML to be output using common library is as follows:

The numbers in figure correspond to serial numbers of “Structure of pagination link” and “Status of pagination link” mentioned above.

- JSP

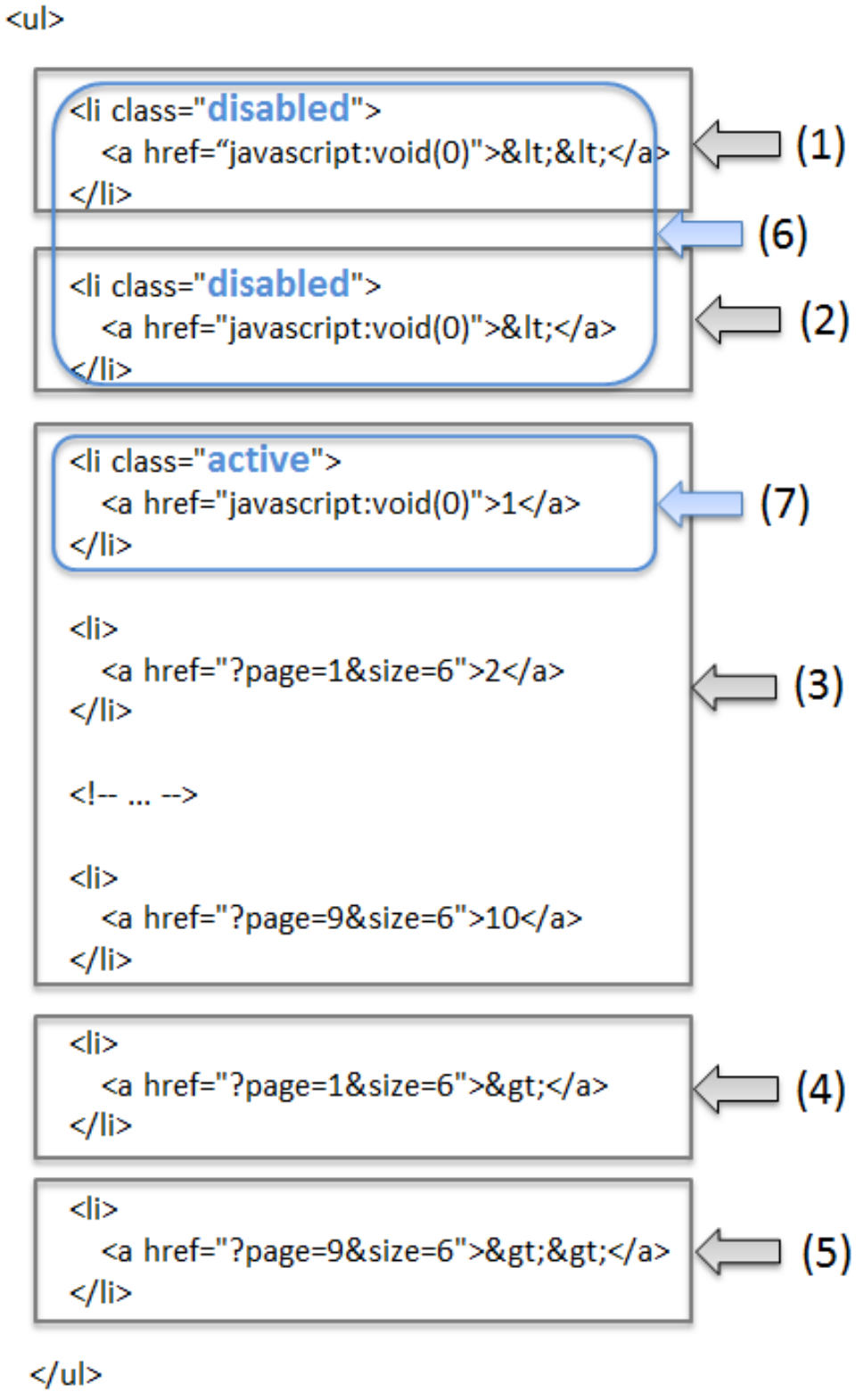
```
<t:pagination page="${page}" />
```

- HTML to be output

HTML of pagination link

HTML of pagination link to be output using common library is as follows:

- HTML



<li class="disabled">
<<

(1)
(6)
<li class="disabled">
<

(2)
<li class="active">
1

(7)

2

<!-- ... -->

10

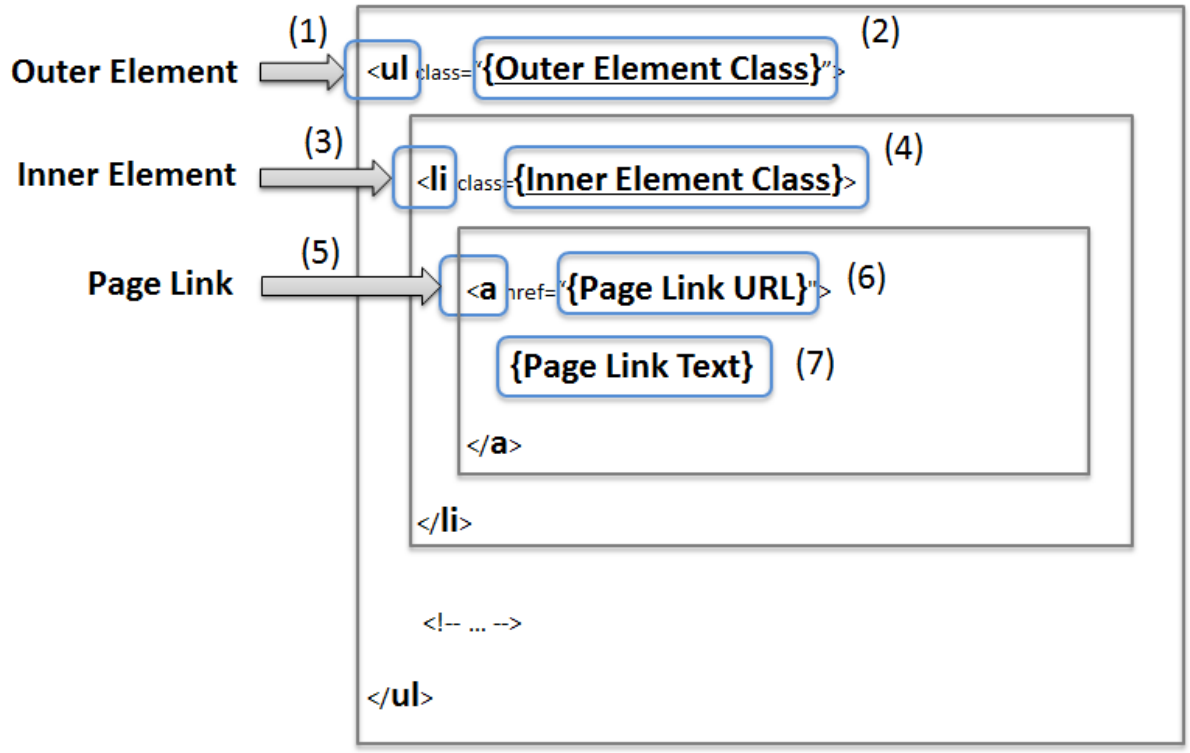
(3)

>

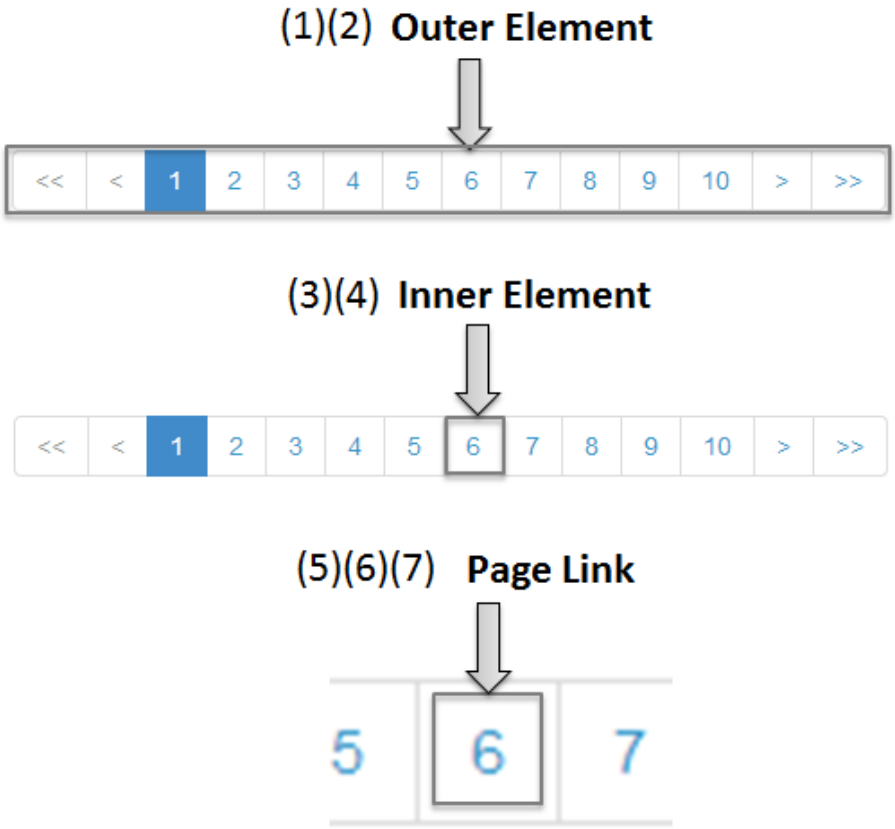
(4)

>>

(5)



- Screen image



Sr. No.	Description	Default values
(1)	Elements to combine the components of pagination link. In common library, this part is called “Outer Element” which stores multiple “Inner Elements”. The elements to be used can be changed using the parameters of JSP tag library.	`<ul>` element
(2)	Attribute to specify style class of “Outer Element”. In common library, this part is called “Outer Element Class”. The attribute values are specified using the parameters of JSP tag library.	No specification

次のページに続く

Table. 4.22 – 前のページからの続き

Sr. No.	Description	Default values
(3)	Elements to configure pagination link. Call this portion as “Inner Element” and maintain <a> element to send request for navigating the page in common library. The elements can be changed using parameter of JSP tag library.	elements
(4)	Attribute to specify style class of “Inner Elements”. In common library, this part is called “Inner Element Class”. The attribute values are switched during JSP tag library processing according to the location of the displayed page.	Refer to “*Note column*”.
(5)	Element to send page navigation request. In common library, this part is called “Page Link”.	Fixed as *<a>*
(6)	Attribute to specify URL for page navigation. In common library, this part is called “Page Link URL”.	Refer to the following “*Note column*”.
(7)	Specify the text to be displayed for page navigation link In common library, this part is called “Page Link Text”.	Refer to the following ” *Note column* ”.

Note: About number of "Inner Elements"

As per default setting, there are maximum 14 "Inner Elements". Their division is as follows:

- Link to navigate to the first page : 1
- Link to navigate to the previous page : 1
- Link to navigate to the specified page : Maximum 10
- Link to navigate to the next page : 1
- Link to navigate to the last page : 1

The number of "Inner Elements" can be changed by specifying parameters of JSP tag library.

Note: About setting values of "Inner Element Class"

As per default setting, following are the three values depending on location of the page.

- `"disabled"` : Style class indicating the link which cannot be operated on the currently displayed page.
- `"active"` : Style class indicating the link of currently displayed page.
- No specification : Indicating the link other than those mentioned above.

`"disabled"` and `"active"` values can be changed by specifying the parameters of JSP tag library.

Note: About default values of "Page Link URL"

When link status is `"disabled"` and `"active"`, the default value is `"javascript:void(0)"` and in other cases, the default value is `"?page={page}&size={size}"`.

"Page Link URL" can be changed to another value by specifying parameters of JSP tag library.

From terasoluna-gfw-web 5.0.0.RELEASE, default value of link in `"active"`status is changed from `"?page={page}&size={size}"` to `"javascript:void(0)"`. This is to match with the implementation of pagination links of major Web sites and the implementation of major CSS libraries (Bootstrap, etc.)

Note: About default values of "Page Link Text"

Sr. No.	Link name	Default values
1.	Link to navigate to the first page	"<<"
2.	Link to navigate to the previous page	"<"
3.	Link to navigate to the specified page	Page number of the corresponding page (cannot be changed)
4.	Link to navigate to the next page	">"
5.	Link to navigate to the last page	">>"

Links other than “Link to navigate to the specified page” can be changed as per the specification of parameters of JSP tag library.

Parameters of JSP tag library

Default operations can be changed by specifying values in parameters of JSP tag library.

List of parameters is shown below.

Parameters to control layout

Sr. No.	Parameter name	Description
1.	outerElement	Specify HTML element name to be used as “Outer Element”. Example: div
2.	outerElementClass	Specify class name of style sheet to be set in “Outer Element Class”. Example: pagination

次のページに続く

Table. 4.23 – 前のページからの続き

Sr. No.	Parameter name	Description
3.	innerElement	Specify HTML element name to be used as “Inner Element”. Example: span
4.	disabledClass	Specify the value to be set in the class attribute of “Inner Element” with "`disabled`" status. Example: hiddenPageLink
5.	activeClass	Specify the value to be set in the class attribute of “Inner Element” with "`active`" status. Example: currentPageLink
6.	firstLinkText	Specify the value to be set in “Page Link Text” of “Link to navigate to the first page”. If "" is specified, “Link to navigate to the first page” itself is not output. Example: First
7.	previousLinkText	Specify the value to be set in “Page Link Text” of “Link to navigate to the previous page”. If "" is specified, “Link to navigate to the previous page” itself is not output. Example: Prev
8.	nextLinkText	Specify the value to be set in “Page Link Text” of “Link to navigate to the next page”. If "" is specified, “Link to navigate to the next page” itself is not output. Example: Next

次のページに続く

Table. 4.23 – 前のページからの続き

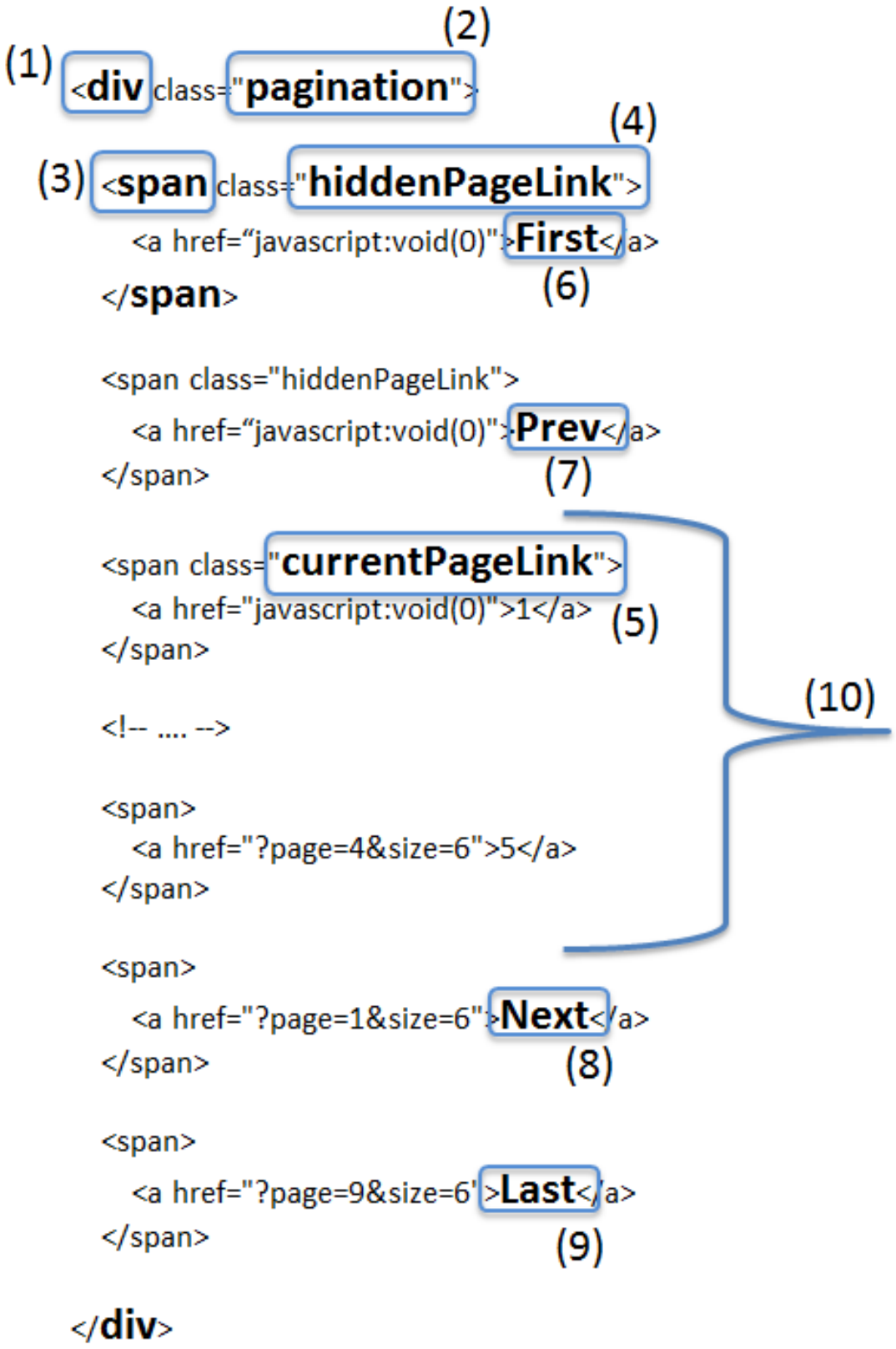
Sr. No.	Parameter name	Description
9.	lastLinkText	Specify the value to be set in “Page Link Text” of “Link to navigate to the last page”. If `""` is specified, “Link to navigate to the next page” itself is not output. Example: Last
10.	maxDisplayCount	Specify maximum display count for “Link to navigate to the specified page”. If `0` is specified, “Link to navigate to the specified page” itself is not output. Example: 5

When default values of all parameters to control the layout are changed, the following HTML is output. The numbers in figure correspond to serial numbers in the parameter list mentioned above.

- JSP

```
<t:pagination page="${page}"
    outerElement="div"
    outerElementClass="pagination"
    innerElement="span"
    disabledClass="hiddenPageLink"
    activeClass="currentPageLink"
    firstLinkText="First"
    previousLinkText="Prev"
    nextLinkText="Next"
    lastLinkText="Last"
    maxDisplayCount="5"
    />
```

- Output HTML


(1)
(2)

(3)
(4)

First

(6)

Prev

(7)

1
(5)

<!-- .... -->

5

(10)

Next

(8)

Last

(9)

Parameters to control operations

Sr. No.	Parameter name	Description
1.	disabledHref	Specify the value to be set in “Page Link URL” having `"disabled"` state.
2.	pathTmpl	Specify the template of request path to be set in “Page Link URL”. When request path at the time of page display and the request path for page navigation are different, request path for page navigation needs to be specified in this parameter. In the template of request path to be specified, location of page (page) and number of records to be fetched (size) can be specified as path variables (placeholders). The specified value of URL is encoded in UTF-8.
3.	queryTmpl	Specify the template of query string of “Page Link URL”. Specify the template for generating pagination related query string (page, size, sort parameters) required at the time of page navigation. When setting request parameter name for location of page or the number of records to be fetched to values other than default values, the query string needs to be specified in this parameter. In the template of query string to be specified, location of page (page) and number of records to be fetched (size) can be specified as path variables (placeholders). The specified value of URL is encoded in UTF-8. This attribute is used to generate pagination related query string (page, size, sort parameters); hence query string for adding search conditions should be specified in criteriaQuery attribute.

次のページに続く

Table. 4.24 – 前のページからの続き

Sr. No.	Parameter name	Description
4.	criteriaQuery	Specify the query string for search conditions to be added to “Page Link URL”. **Specify the query string for search conditions in this parameter when adding search conditions to “Page Link URL”.** **The specified value of URL is not encoded; hence encoded URL query string needs to be specified.** If EL function (`f:query(Object)`) of common library is used when converting the search conditions stored in form object into encoded URL query string, the search conditions can be added easily. This parameter can be used in terasoluna-gfw-web 1.0.1.RELEASE or higher version.
5.	disableHtmlEscapeOfCriteriaQuery	Flag to disable HTML escaping for the values specified in `criteriaQuery` parameter. When the flag is set to `true`, HTML escaping is no longer possible for the values specified in `criteriaQuery` parameter. (Default value is `false`). **When specifying true, it should be ensured that the characters vulnerable to XSS are not included in the query string.** This parameter can be used in terasoluna-gfw-web 1.0.1.RELEASE or higher version.

次のページに続く

Table. 4.24 – 前のページからの続き

Sr. No.	Parameter name	Description
6.	enableLinkOfCurrentPage	Flag to send request for redisplaying the corresponding page on clicking page link in `"active"` status. When it is set to `true`, URL (default value is `"?page={page}&size={size}"`) to redisplay the corresponding page is set to “Page Link URL”. (If default value is `false`, the value of `disabledHref` attribute is set to “Page Link URL”) This parameter can be used in terasoluna-gfw-web 5.0.0.RELEASE or higher version.

Note: About setting values of disabledHref

"`javascript:void(0)`"is set in `disabledHref` attribute by default. It may remain as default in order to disable only the operation of page link click.

However, if the focus is moved or on mouseover to page link in default state, "`javascript:void(0)`" may be displayed on browser status bar. To change this behavior, it is necessary to disable the operation of page link click by using JavaScript. Refer to "*To disable page link using JavaScript*" for implementation example.

From terasoluna-gfw-web 5.0.0.RELEASE, default value of `disabledHref` attribute is changed from "`#`" to "`javascript:void(0)`". By doing so, the focus does not move to top of the page on clicking page link in "`disabled`" state.

Note: Path variables (placeholders)

Path variables that can be specified in `pathTmpl` and `queryTmpl` are as follows:

Sr. No.	Path variable name	Description
1.	page	Path variable for inserting page location.
2.	size	Path variable for inserting number of records to be fetched.
3.	sortOrderProperty	Path variable for inserting sort field of sort condition.
4.	sortOrderDirection	Path variable for inserting sort order of sort condition.

Specify path variables in "`{Path variable name}`" format.

Warning: Constraints related to sort condition

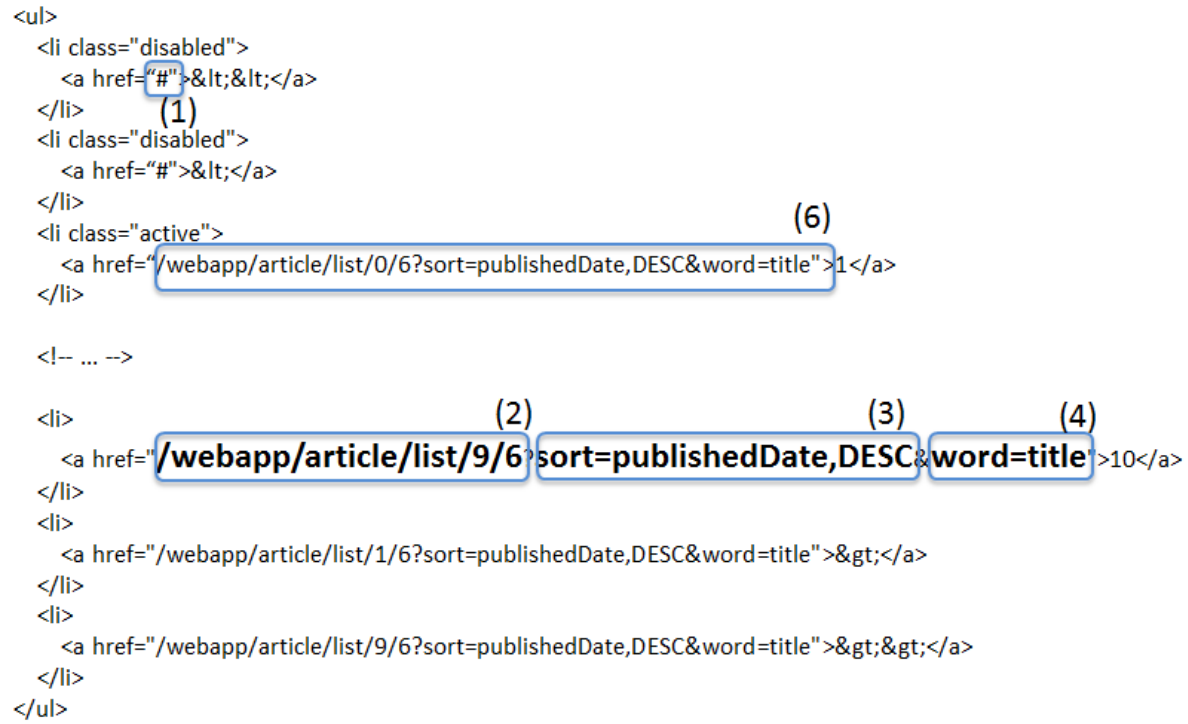
Only one sort condition can be set as a sort condition path variable. Therefore, when the search result obtained by specifying multiple sort condition needs to be displayed using pagination, it is necessary to extend the JSP tag library of common library.

When parameters to control the operations are changed, the following HTML is output. The numbers in figure correspond to serial numbers of parameter list mentioned above.

- JSP

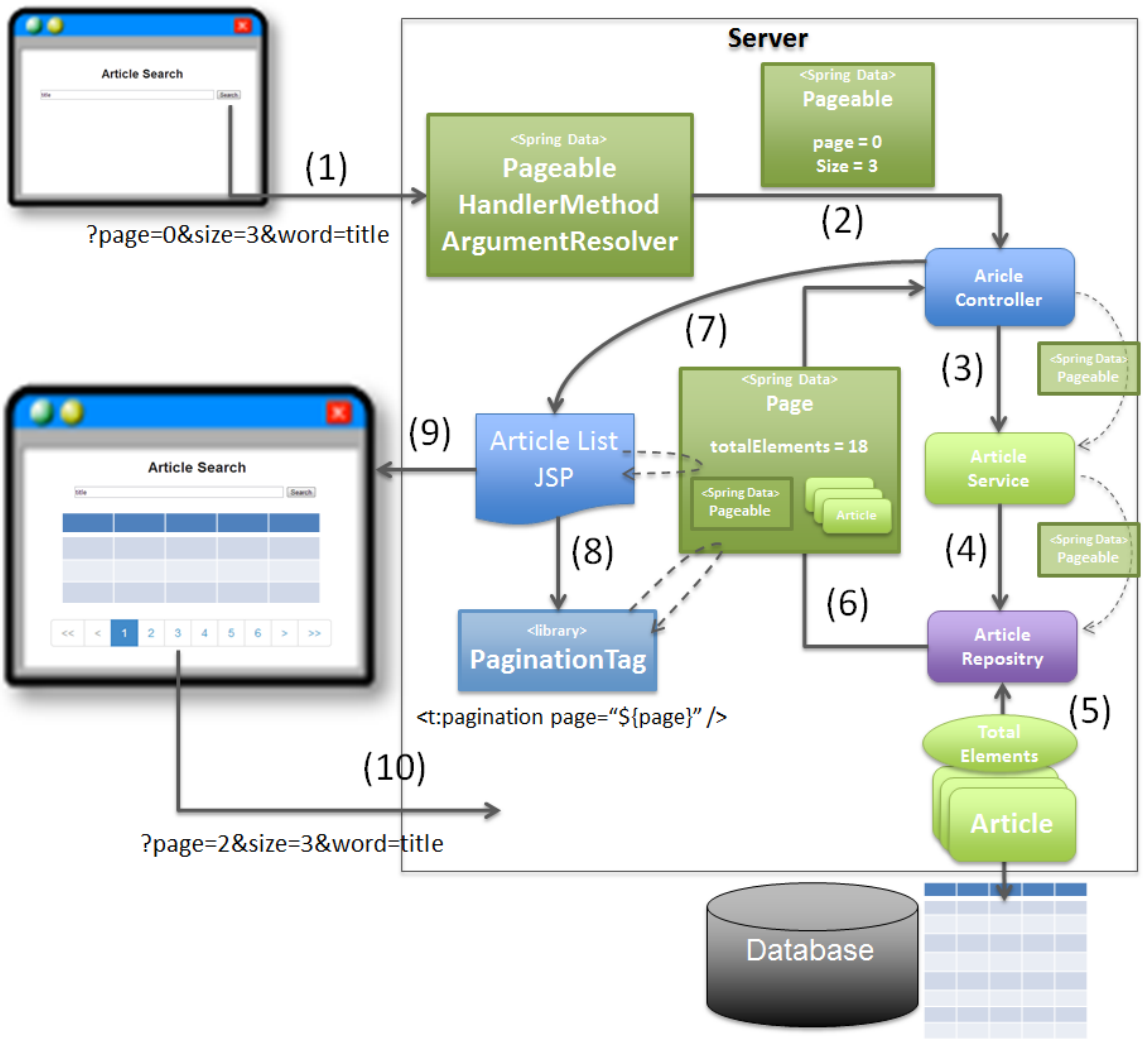
```
<t:pagination page="${page}"
    disabledHref="#"
    pathTmpl="${pageContext.request.contextPath}/article/list/{page}/{size}
↪"
    queryTmpl="sort={sortOrderProperty},{sortOrderDirection}"
    criteriaQuery="${f:query(articleSearchCriteriaForm)}"
    enableLinkOfCurrentPage="true" />
```

- HTML to be output



Process flow when pagination is used

Process flow when using pagination functionality of Spring Data and JSP tag library of common library is as follows:



Sr. No.	Description
(1)	Apart from search conditions, specify the location of page to be searched (page) and number of records to be fetched (size) as request parameters and send the request.
(2)	`PageableHandlerMethodArgumentResolver` fetches location of page to be searched (page) and number of records to be fetched (size) specified in request parameter and creates `Pageable` object. The created `Pageable` object is set as an argument of Controller handler method.

次のページに続く

Table. 4.25 – 前のページからの続き

Sr. No.	Description
(3)	Controller passes the `Pageable` object received as an argument to Service method.
(4)	Service passes the `Pageable` object received as an argument to Query method of Repository.
(5)	Repository fetches total records (totalElements) of data matching the search conditions. It also fetches the data falling in the range of page location (page) and number of records to be fetched (size) specified in `Pageable` object received as an argument, from the database.
(6)	Repository creates `Page` object based on total records fetched (totalElements), fetched data (content) and `Pageable` object received as an argument, and returns it to Service and Controller.
(7)	Controller stores the returned `Page` object in `Model` object and displays JSP.
(8)	JSP fetches the `Page` object stored in `Model` object and calls JSP tag library (`<t:pagination>`) for pagination provided by common library. JSP tag library for pagination refers to `Page` object and creates pagination link.
(9)	The HTML created in JSP is returned to client (browser).
(10)	On clicking pagination link, the request to display the corresponding page is sent.

Note: Implementation of Repository

The implementation method of (5) & (6) are different depending on the O/R Mapper to be used.

- When using MyBatis3, implementation of Java (Service) and SQL mapping file is necessary.
- When using Spring Data JPA, implementation is not necessary because it is carried out automatically through the use of Spring Data JPA functionality.

For implementation example refer to:

- *Database Access (MyBatis3)*
- *Database Access (JPA)*

4.4.2 How to use

The method of using pagination functionality is as follows:

Application settings

Settings for enabling pagination functionality of Spring Data

Location of page to be searched (page), number of records to be fetched (size) and sort condition (sort) are specified in the request parameter. The functionality to set these properties in the argument of Controller as `Pageable` object should be enabled.
The settings mentioned below are preset in a blank project.

`spring-mvc.xml`

```
<mvc:annotation-driven>
    <mvc:argument-resolvers>
        <!-- (1) -->
        <bean
            class="org.springframework.data.web.
↪PageableHandlerMethodArgumentResolver" />
    </mvc:argument-resolvers>
</mvc:annotation-driven>
```

Sr. No.	Description
(1)	Specify `org.springframework.data.web.PageableHandlerMethodArgumentResolver` in `<mvc:argument-resolvers>`. For the properties that can be specified in `PageableHandlerMethodArgumentResolver`, refer to "*About property values of PageableHandlerMethodArgumentResolver*".

Page search

The method of implementing page search is as follows:

Implementation of application layer

The information required for page search (such as location of page to be searched, number of records to be fetched and sort condition) is received as an argument and passed to Service method.

- Controller

```
@RequestMapping("list")
public String list(@Validated ArticleSearchCriteriaForm form,
        BindingResult result,
        Pageable pageable, // (1)
        Model model) {

    ArticleSearchCriteria criteria = beanMapper.map(form,
            ArticleSearchCriteria.class);

    Page<Article> page = articleService.searchArticle(criteria, pageable); //␣
↪(2)

    model.addAttribute("page", page); // (3)

    return "article/list";
}
```

Sr. No.	Description
(1)	Specify `Pageable` as an argument of handler method. `Pageable` object stores the information required for page search (such as location of page to be searched, number of records to be fetched and sort condition).
(2)	Specify the `Pageable` object as an argument of Service method and then call the same.
(3)	Add the search result (`Page` object) returned by Service to `Model`. It can be referred from View (JSP) after it is added to `Model`.

Note: Operations when the information required for page search is not specified in request parameter

Default values are applied when the information required for page search (such as location of page to be searched, number of records to be fetched and sort condition) is not specified in request parameter. Default values are as follows:

- Location of page to be searched: *0* (first page)
- number of records to be fetched: *20*
- Sort condition: *null* (no sort condition)

Default values can be changed using the following two methods.

- Define the default values by specifying `@org.springframework.data.web.PageableDefault` annotation as an argument of `Pageable` of handler method.
- Specify `Pageable` object wherein default values are defined in `fallbackPageable` property of `PageableHandlerMethodArgumentResolver`.

See the method below for specifying default values using `@PageableDefault` annotation.

Use `@PageableDefault` annotation to change the default values for each page search.

```
@RequestMapping("list")
public String list(@Validated ArticleSearchCriteriaForm form,
        BindingResult result,
        @PageableDefault( // (1)
                page = 0,    // (2)
                size = 50,   // (3)
                direction = Direction.DESC,  // (4)
                sort = {     // (5)
                    "publishedDate",
                    "articleId"
                    }
                ) Pageable pageable,
        Model model) {
    // ...
    return "article/list";
}
```

Sr. No.	Description	Default values
(1)	Specify `@PageableDefault` annotation as an argument of `Pageable`.	-
(2)	To change the default value of location of page, specify the value in page attribute of `@PageableDefault` annotation. Normally it need not be changed.	`0` (first page)
(3)	To change the default value of number of records to be fetched, specify the value in size or value attribute of `@PageableDefault` annotation.	`10`
(4)	To change the default value of sort condition, specify the value in direction attribute of `@PageableDefault` annotation.	`Direction.ASC` (Ascending order)
(5)	Specify the sort fields of sort condition in sort attribute of `@PageableDefault` annotation. When sorting the records using multiple sort fields, specify the property name to be sorted in array. In the above example, sort condition `"ORDER BY publishedDate DESC,articleId DESC"` is added to Query.	Empty array (No sort field)

Note: About sort order that can be specified using @PageableDefault annotation

Sort order that can be specified using `@PageableDefault` annotation is either ascending or descending; hence when you want to specify different sort order for each field, it is necessary to use `@org.springframework.data.web.SortDefaults` annotation. For example, when sorting the fields using `"ORDER BY publishedDate DESC,articleId ASC"` sort order.

Tip: Specifying annotation when only the default value of number of records to be fetched is to be changed

In order to change only the default value of number of records to be fetched, the annotation can also be

specified as `@PageableDefault(50)`. This operation is same as `@PageableDefault(size = 50)`.

See the method below to specify default values using `@SortDefaults` annotation.

`@SortDefaults` annotation is used when sorting needs to be done on multiple fields and in order to have different sort order for each field.

```
@RequestMapping("list")
public String list(
        @Validated ArticleSearchCriteriaForm form,
        BindingResult result,
        @PageableDefault(size = 50)
        @SortDefaults(  // (1)
                {
                    @SortDefault(  // (2)
                                 sort = "publishedDate",     // (3)
                                 direction = Direction.DESC // (4)
                                ),
                    @SortDefault(
                                 sort = "articleId"
                                )
                }) Pageable pageable,
        Model model) {
    // ...
    return "article/list";
}
```

Sr. No.	Description	Default values
(1)	Specify `@SortDefaults` annotation as an argument of `Pageable`. Multiple `@org.springframework.data.web.SortDefault` annotations can be specified as arrays in `@SortDefaults` annotation.	-
(2)	Specify `@SortDefault` annotation as value attribute of `@SortDefaults` annotation. Specify as array when specifying multiple annotations.	-
(3)	Specify sort fields in sort or value attribute of `@PageableDefault`. Specify as array when specifying multiple fields.	Empty array (No sort field)
(4)	Specify value in direction attribute of `@PageableDefault` to change default sort condition.	`Direction.ASC` (Ascending)

In the above example, sort condition called `"ORDER BY publishedDate DESC,articleId ASC"` is added to query.

Tip: Specifying annotation when only the default value of sort fields is to be specified

In order to specify only the records to be fetched, the annotation can also be specified as `@PageableDefault("articleId")`. This operation is same as `@PageableDefault(sort = "articleId")` and `@PageableDefault(sort = "articleId",direction = Direction.ASC)`.

When it is necessary to change the default values of entire application, specify `Pageable` object wherein default values are defined in `fallbackPageable` property of `PageableHandlerMethodArgumentResolver` that is defined in `spring-mvc.xml`.

For description of `fallbackPageable` and example of settings, refer to "*About property values of Pageable-*

HandlerMethodArgumentResolver".

Implementation of domain layer (MyBatis3)

When accessing the database using MyBatis3, extract the necessary information from `Pageable` object received from Controller and pass it to the Repository.
Sort conditions and SQL for extracting the corresponding data need to be implemented in SQL mapping.

For details on page search process to be implemented at domain layer, refer to:

- *Pagination search of Entity (MyBatis3 standard method)*
- *Pagination search for Entity (SQL refinement method)*

Implementation of domain layer (JPA)

When accessing the database using JPA (Spring Data JPA), pass the `Pageable` object received from Controller to Repository.

For details on page search process to be implemented at domain layer, refer to:

- *Searching page of entities matching the conditions*

Implementation of JSP (Base version)

The method to display pagination link and pagination information (total records, total pages, number of displayed pages etc.) by displaying the `Page` object fetched during page search on list screen, is described below.

Display of fetched data

An example to display the data fetched during page search is shown below.

- Controller

```
@RequestMapping("list")
public String list(@Validated ArticleSearchCriteriaForm form, BindingResult
↪result,
        Pageable pageable, Model model) {

    if (!StringUtils.hasLength(form.getWord())) {
        return "article/list";
    }

    ArticleSearchCriteria criteria = beanMapper.map(form,
            ArticleSearchCriteria.class);

    Page<Article> page = articleService.searchArticle(criteria, pageable);

    model.addAttribute("page", page); // (1)

    return "article/list";
}
```

Sr. No.	Description
(1)	Store `Page` object with the attribute name `"page"` in `Model`. In JSP, `Page` object can be accessed by specifying attribute name `"page"`.

- JSP

```
<%-- ... --%>

<%-- (2) --%>
<c:when test="${page != null && page.totalPages != 0}">

  <table class="maintable">
    <thead>
      <tr>
        <th class="no">No</th>
        <th class="articleClass">Class</th>
        <th class="title">Title</th>
        <th class="overview">Overview</th>
        <th class="date">Published Date</th>
      </tr>
    </thead>

    <%-- (3) --%>
    <c:forEach var="article" items="${page.content}" varStatus="rowStatus">
      <tr>
        <td class="no">
```

```
          ${(page.number * page.size) + rowStatus.count}
        </td>
        <td class="articleClass">
          ${f:h(article.articleClass.name)}
        </td>
        <td class="title">
          ${f:h(article.title)}
        </td>
        <td class="overview">
          ${f:h(article.overview)}
        </td>
        <td class="date">
          ${f:h(article.publishedDate)}
        </td>
      </tr>
    </c:forEach>

  </table>

  <div class="paginationPart">

    <%-- ... --%>

  </div>
</c:when>

<%-- ... --%>
```

Sr. No.	Description
(2)	In the above example, it is checked whether data matching the specified conditions exists. If there is no such data, the header row is also not displayed. When it is necessary to display the header row even if there is no matching data, this branching is no longer required.
(3)	Display the list of data fetched using `<c:forEach>` tag of JSTL. The fetched data is stored in a list in `content` property of `Page` object.

- Example of screen output in JSP

No	Class	Title	Overview	Published Date
1	Internal	Internal title1_20	overview20	2013-10-29
2	International	International title2_20	overview20	2013-10-29
3	Economy	Economy title3_20	overview20	2013-10-29
4	Internal	Internal title1_19	overview19	2013-10-28
5	International	International title2_19	overview19	2013-10-28
6	Economy	Economy title3_19	overview19	2013-10-28

.
.
.

Display of Pagination link

See the example below to display the link for page navigation (pagination link).

Pagination link is output using JSP tag library of common library.

- `include.jsp`

  Declare the JSP tag library of common library. The settings below are carried out in a blank project.

```
<%@ taglib uri="http://terasoluna.org/tags" prefix="t"%>        <%-- (1) --%>
<%@ taglib uri="http://terasoluna.org/functions" prefix="f"%>  <%-- (2) --%>
```

Sr. No.	Description
(1)	JSP tag to display pagination link is stored.
(2)	EL function of JSP used at the time of using pagination link, is stored.

- JSP

```
<t:pagination page="${page}" /> <%-- (3) --%>
```

Sr. No.	Description
(3)	Use `<t:pagination>` tag. In page attribute, specify `Page` object stored in `Model` of Controller.

- Output HTML

  The example below shows the search results obtained upon specifying "?page=0&size=6".

```
 <ul>
    <li class="disabled"><a href="javascript:void(0)">&lt;&lt;</a></li>
    <li class="disabled"><a href="javascript:void(0)">&lt;</a></li>
    <li class="active"><a href="javascript:void(0)">1</a></li>
    <li><a href="?page=1&size=6">2</a></li>
    <li><a href="?page=2&size=6">3</a></li>
    <li><a href="?page=3&size=6">4</a></li>
    <li><a href="?page=4&size=6">5</a></li>
    <li><a href="?page=5&size=6">6</a></li>
    <li><a href="?page=6&size=6">7</a></li>
    <li><a href="?page=7&size=6">8</a></li>
    <li><a href="?page=8&size=6">9</a></li>
    <li><a href="?page=9&size=6">10</a></li>
    <li><a href="?page=1&size=6">&gt;</a></li>
    <li><a href="?page=9&size=6">&gt;&gt;</a></li>
</ul>
```

If style sheet for pagination link is not created, the display will be as follows:

As it is visible below, the pagination link is not established.

The display will be as follows if minimal changes are carried out like adding a definition of style sheet for pagination link and changing the JSP.

- Screen image

- JSP

- <<
- <
- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- >
- >>

<< < 1 2 3 4 5 6 7 8 9 10 > >>

```
<%-- ... --%>

<t:pagination page="${page}"
    outerElementClass="pagination" /> <%-- (4) --%>

<%-- ... --%>
```

Sr. No.	Description
(4)	Specify the class name indicating that it is a pagination link. By specifying the class name, the applicable range of styles to be specified in style sheet can be restricted to pagination link.

- Style sheet

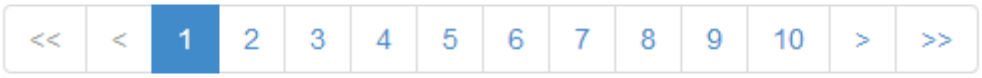
```
.pagination li {
    display: inline;
}

.pagination li>a {
    margin-left: 10px;
}
```

Even after the pagination link is established, the following two problems still persist.

- clickable and non-clickable links cannot be distinguished.
- The location of currently displayed page cannot be identified.

When Bootstrap v3.0.0 style sheet is applied to resolve the above problems, the display is as follows:

- Screen image



- Style sheet

  Place the css file of bootstrap v3.0.0 under `$WEB_APP_ROOT/resources/vendor/bootstrap-3.0.0/css/bootstrap.css`.

  Abstract of pagination related style definition.

```
.pagination {
  display: inline-block;
  padding-left: 0;
  margin: 20px 0;
  border-radius: 4px;
}

.pagination > li {
  display: inline;
}

.pagination > li > a,
.pagination > li > span {
  position: relative;
  float: left;
  padding: 6px 12px;
  margin-left: -1px;
  line-height: 1.428571429;
  text-decoration: none;
  background-color: #ffffff;
  border: 1px solid #dddddd;
}
```

```
.pagination > li:first-child > a,
.pagination > li:first-child > span {
  margin-left: 0;
  border-bottom-left-radius: 4px;
  border-top-left-radius: 4px;
}

.pagination > li:last-child > a,
.pagination > li:last-child > span {
  border-top-right-radius: 4px;
  border-bottom-right-radius: 4px;
}

.pagination > li > a:hover,
.pagination > li > span:hover,
.pagination > li > a:focus,
.pagination > li > span:focus {
  background-color: #eeeeee;
}

.pagination > .active > a,
.pagination > .active > span,
.pagination > .active > a:hover,
.pagination > .active > span:hover,
.pagination > .active > a:focus,
.pagination > .active > span:focus {
  z-index: 2;
  color: #ffffff;
  cursor: default;
  background-color: #428bca;
  border-color: #428bca;
}

.pagination > .disabled > span,
.pagination > .disabled > a,
.pagination > .disabled > a:hover,
.pagination > .disabled > a:focus {
  color: #999999;
  cursor: not-allowed;
  background-color: #ffffff;
  border-color: #dddddd;
}
```

- JSP

  Add a definition to read the css file placed under JSP.

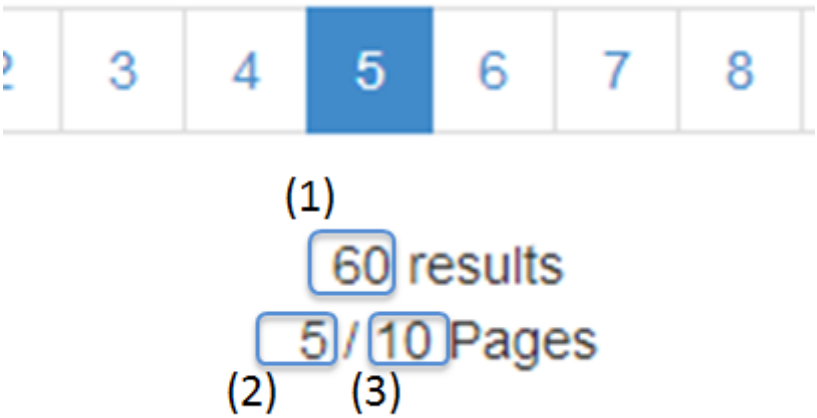
```
<link rel="stylesheet"
    href="${pageContext.request.contextPath}/resources/vendor/bootstrap-3.0.0/
↪css/bootstrap.css"
```

```
    type="text/css" media="screen, projection">
```

Display of pagination information

An example to display the information related to pagination (such as total records, total pages and total displayed pages) is as follows:

- Screen example



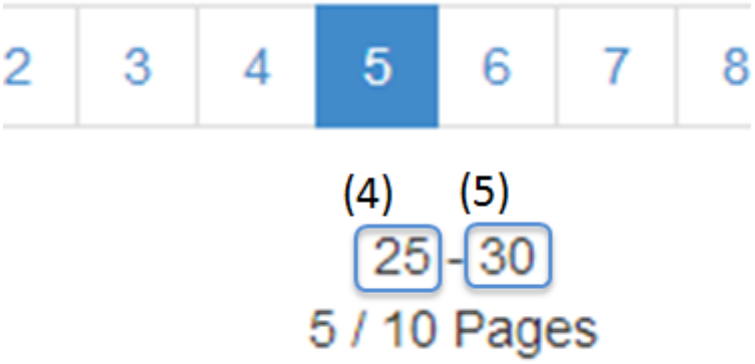
- JSP

```
<div>
    <fmt:formatNumber value="${page.totalElements}" /> results <%-- (1) --%>
</div>
<div>
    ${f:h(page.number + 1) } /        <%-- (2) --%>
    ${f:h(page.totalPages)} Pages     <%-- (3) --%>
</div>
```

Sr. No.	Description
(1)	To display total number of data records matching the search conditions, fetch value from `totalElements` property of `Page` object.
(2)	To display number of displayed pages, fetch value from `number` property of `Page` object and increment the value by 1. `number` property of `Page` object starts with `0`; hence value should be incremented by 1 at the time of displaying the page number.
(3)	To display total pages of data matching the search conditions, fetch the value from `totalPages` property of `Page` object.

Example to display the display data range of the corresponding page is shown below.

- Example of screen



- JSP

```
<div>
    <%-- (4) --%>
    <fmt:formatNumber value="${(page.number * page.size) + 1}" /> -
    <%-- (5) --%>
    <fmt:formatNumber value="${(page.number * page.size) + page.
↪numberOfElements}" />
</div>
```

Sr. No.	Description
(4)	To display start location, calculate the value using `number` property and `size` property of `Page` object. `number` property of `Page` object starts with `0`; hence the value needs to be incremented by 1 at the time of displaying data start location.
(5)	To display end location, calculate the value using `number` property, `size` property and `numberOfElements` property of `Page` object. `numberOfElements` needs to be calculated since the last page is likely to be a fraction.

Tip: About format of numeric value

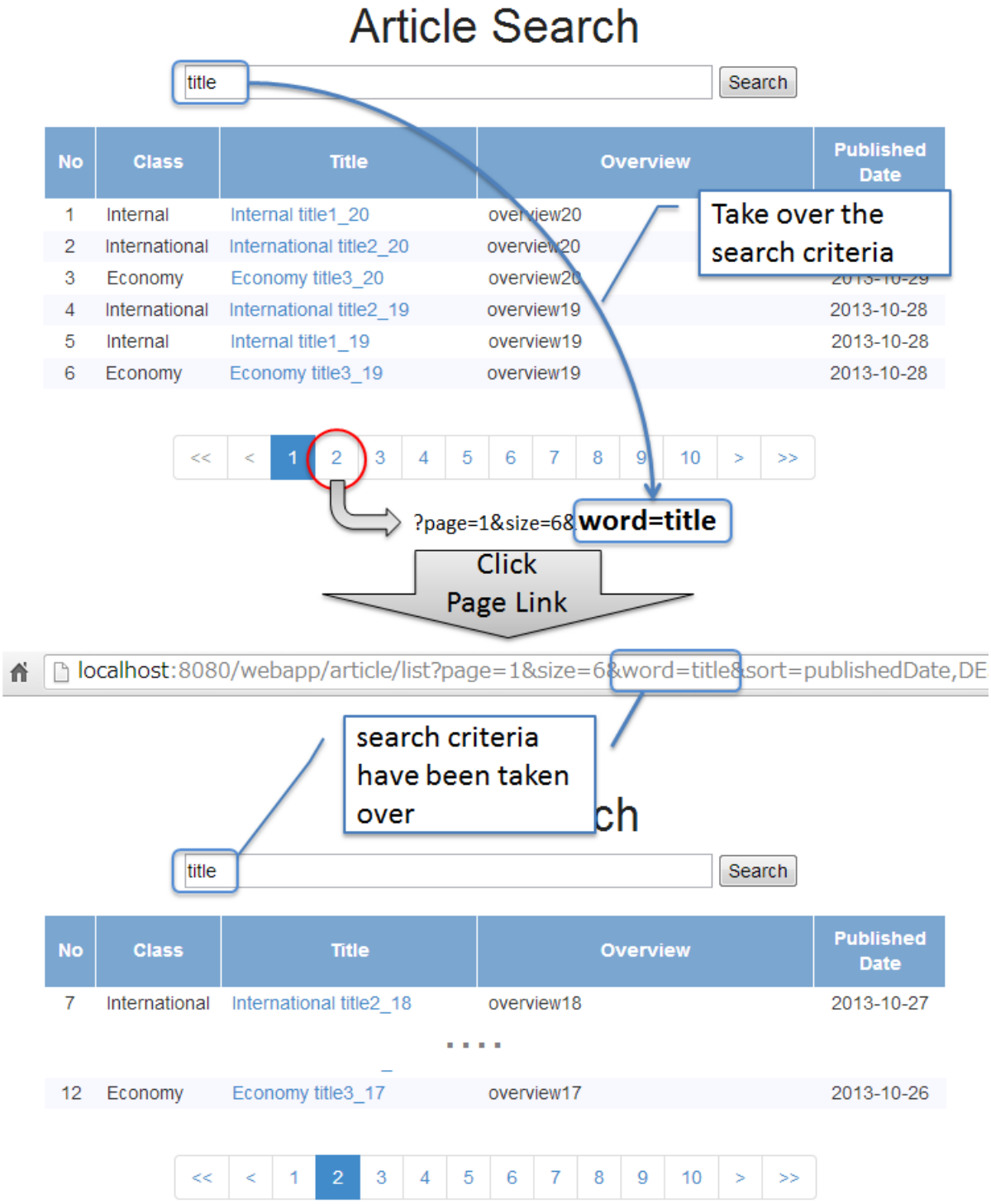
When the numeric value to be displayed needs to be formatted, use tag library (`<fmt:formatNumber>`) provided by JSTL.

Carrying forward search conditions using page link

The method of carrying forward the search conditions to the page navigation request is shown below.

- JSP

```
<%-- (1) --%>
<div id="criteriaPart">
  <form:form action="${pageContext.request.contextPath}/article/list" method=
↪"get"
             modelAttribute="articleSearchCriteriaForm">
```


Article Search
title
Search
No
Class
Title
Overview
Published Date
1 Internal Internal title1_20 overview20
2 International International title2_20 overview20
3 Economy Economy title3_20 overview20
4 International International title2_19 overview19 2013-10-28
5 Internal Internal title1_19 overview19 2013-10-28
6 Economy Economy title3_19 overview19 2013-10-28
Take over the search criteria
<< < 1 2 3 4 5 6 7 8 9 10 > >>
?page=1&size=6&word=title
Click Page Link
localhost:8080/webapp/article/list?page=1&size=6&word=title&sort=publishedDate,DE
search criteria have been taken over
title
Search
No
Class
Title
Overview
Published Date
7 International International title2_18 overview18 2013-10-27
12 Economy Economy title3_17 overview17 2013-10-26
<< < 1 2 3 4 5 6 7 8 9 10 > >>

```
    <form:input path="word" />
    <form:button>Search</form:button>
    <br>
  </form:form>
</div>

<%-- ... --%>

<t:pagination page="${page}"
    outerElementClass="pagination"
    criteriaQuery="${f:query(articleSearchCriteriaForm)}" /> <%-- (2) --%>
```

Sr. No.	Description
(1)	Form to specify search conditions. `word` is specified as a search condition.
(2)	When carrying forward the search conditions to page navigation request, specify the **encoded URL query string** in `criteriaQuery` attribute. When storing the search conditions in form object, conditions can be carried forward easily if EL function (`f:query(Object)`) provided by common library is used. In the above example, query string of "`?page=page location&size=number of records to be fetched&word=input value`" format is generated. `criteriaQuery`attribute can be used in terasoluna-gfw-web 1.0.1.RELEASE or higher version.

Note: Specifications of f:query(Object)

JavaBean of form object and `Map` object can be specified as an argument of `f:query`. In case of JavaBean, property name is treated as request parameter name and in case of `Map` object, map key name is treated as request parameter. URL of the generated query string is encoded in UTF-8.

From the terasoluna-gfw-web 5.0.1.RELEASE, `f:query` has been supporting a nested structured JavaBean or `Map`.

Refer to *f:query()* for detail specification of the `f:query` (URL encoding specification etc).

Warning: Operations when Query string created using f:query is specified in queryTmpl attribute

It has been found that specifying the query string generated using `f:query`, in queryTmpl attribute leads to duplication of URL encoding. Thus, special characters are not carried forward correctly.

This URL encoding duplication can be avoided by using `criteriaQuery` attribute which can be used in terasoluna-gfw-web 1.0.1.RELEASE or higher version.

Carrying forward the sort condition using page link

The method to carry forward the sort condition to page navigation request is as follows:

- JSP

```
<t:pagination page="${page}"
    outerElementClass="pagination"
    queryTmpl="page={page}&size={size}&sort={sortOrderProperty},
↪{sortOrderDirection}" />  <%-- (1) --%>
```

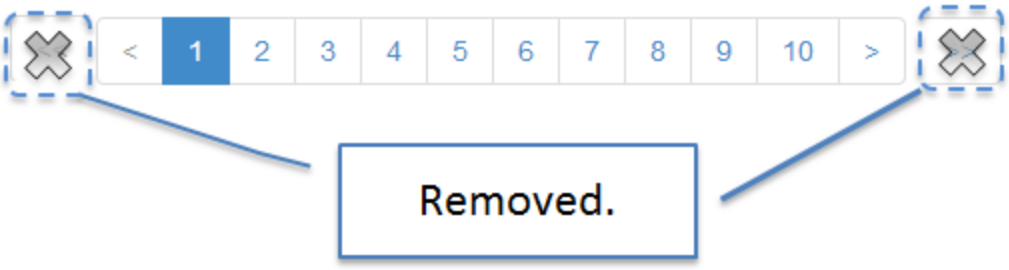
Sr. No.	Description
(1)	To carry forward the sort condition to page navigation request, specify `queryTmpl` and add sort condition to query string. For parameter specifications to specify sort condition, refer to "*Request parameters for page search* " In the above example, "`?page=0&size=20&sort=sort item,sort order(ASC or DESC)`" is a query string.

Implementation of JSP (layout change)

Removal of link to navigate to the first page and the last page

Example to remove "Link to navigate to the first page" and "Link to navigate to the last page" is shown below.

- Screen example



- JSP

```
<t:pagination page="${page}"
    outerElementClass="pagination"
    firstLinkText=""
    lastLinkText="" /> <%-- (1) (2) --%>
```

Sr. No.	Description
(1)	Specify "" as firstLinkText attribute of <t:pagination> tag to hide “Link to navigate to the first page”.
(2)	Specify "" as lastLinkText attribute of <t:pagination> tag to hide “Link to navigate to the last page”.

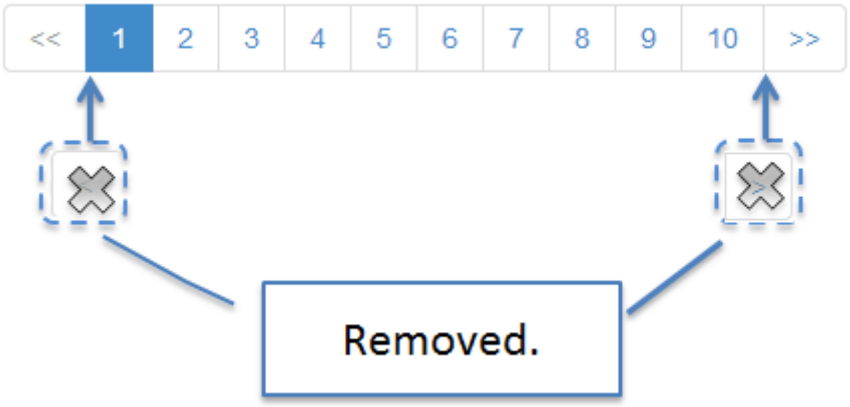
Removal of link to navigate to previous page and next page

Example to remove “Link to navigate to the first page” and “Link to navigate to the last page” is shown below.

- Screen example

- JSP

```
<t:pagination page="${page}"
    outerElementClass="pagination"
```




```
    previousLinkText=""
    nextLinkText="" /> <%-- (1) (2) --%>
```

Sr. No.	Description
(1)	Specify "" as previousLinkText attribute of <t:pagination> tag to hide “Link to navigate to the previous page”.
(2)	Specify "" as nextLinkText attribute of <t:pagination> tag to hide “Link to navigate to the next page”.

Removal of disabled link

Example to remove the link in "disabled" state is shown below.

Add the following definition to style sheet when the status is "disabled".

- Screen example

- Style sheet

```
.pagination .disabled {
    display: none;  /* (1) */
```



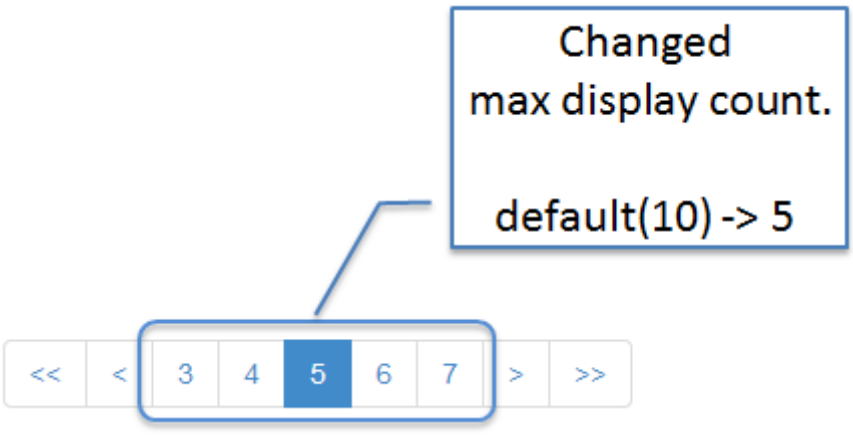

```
}
```

Sr. No.	Description
(1)	Specify "`display: none;`" as an attribute value of "`disabled`" class.

Change in maximum number of display links to navigate to the specified page

Example to change maximum number of display links to navigate to the specified page is shown below.

- Screen example



- JSP

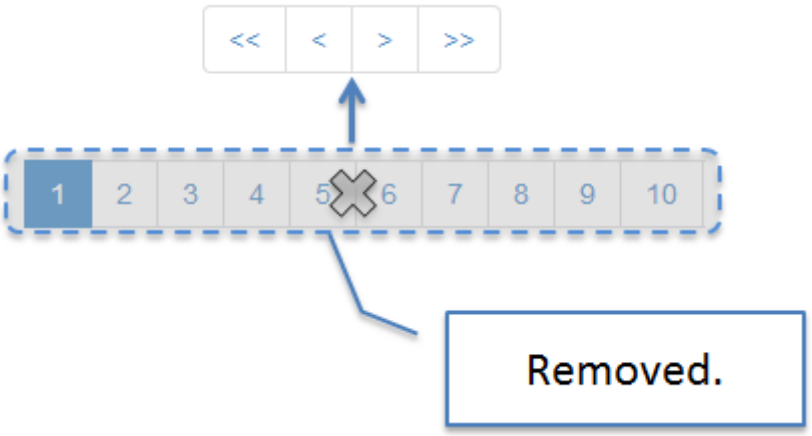
```
<t:pagination page="${page}"
    outerElementClass="pagination"
    maxDisplayCount="5" /> <%-- (1) --%>
```

Sr. No.	Description
(1)	In order to change maximum number of display links to navigate to the specified page, specify value in maxDisplayCount attribute of `<t:pagination>` tag.

Removal of link to navigate to the specified page

Example to remove link to navigate to the specified page is shown below.

- Screen example



- JSP

```
<t:pagination page="${page}"
    outerElementClass="pagination"
    maxDisplayCount="0" /> <%-- (1) --%>
```

Sr. No.	Description
(1)	In order to hide the link to navigate to the specified page, specify `"0"` as maxDisplayCount attribute of `<t:pagination>` tag.

Implementation of JSP (Operation)

Specifying sort condition

Example to specify sort condition from client is shown below.

- Screen example



- JSP

```
<div id="criteriaPart">
  <form:form
    action="${pageContext.request.contextPath}/article/search"
    method="get" modelAttribute="articleSearchCriteriaForm">
    <form:input path="word" />
    <%-- (1) --%>
    <form:select path="sort">
        <form:option value="publishedDate,DESC">Newest</form:option>
        <form:option value="publishedDate,ASC">Oldest</form:option>
    </form:select>
    <form:button>Search</form:button>
    <br>
  </form:form>
</div>
```

Sr. No.	Description
(1)	For specifying the sort condition from client, add the corresponding parameters for specifying the sort condition. For parameter specifications to specify sort condition, refer to "*Request parameters for page search* ". In the above example, publishedDate can be selected in ascending order or descending order from pull-down.

To disable page link using JavaScript

By default "`javascript:void(0)`" is set in `disabledHref` attribute of `<t:pagination>` tag to disable the operation on clicking page link in "`disabled`" state and "`active`" state. In such a state, if focus is moved or on mouseover to page link, "`javascript:void(0)`" is displayed on browser status bar. To change this behavior, it is necessary to disable the operation of page link click by using JavaScript.

Implementation example is shown below.

JSP

```
<%-- (1) --%>
<script type="text/javascript"
        src="${pageContext.request.contextPath}/resources/vendor/js/jquery.js"></
↪script>

<%-- (2) --%>
<script type="text/javascript">
    $(function(){
        $(document).on("click", ".disabled a, .active a", function(){
            return false;
        });
    });
</script>

<%-- ... --%>

<%-- (3) --%>
<t:pagination page="${page}" disabledHref="#" />
```

Sr. No.	Description
(1)	Read js file of jQuery. In the above example, jQuery API is used to disable the operation of page link click using JavaScript.
(2)	Disable click event of page link of "`disabled`" and "`active`" states by using API of jQuery. However, when `enableLinkOfCurrentPage` attribute of `<t:pagination>` tag is set to "`true`", the click event of page link in "`active`" state should not be disabled.
(3)	Set "`#`" in `disabledHref` attribute.

4.4.3 Appendix

About property values of `PageableHandlerMethodArgumentResolver`

Properties that can be specified in `PageableHandlerMethodArgumentResolver` are as follows:

Values should be changed as required in the application.

Sr. No.	Property name	Description	Default value
1.	maxPageSize	Specify maximum permissible value for the number of records to be fetched. When the specified number of records to be fetched exceeds `maxPageSize`, only the number of records specified as `maxPageSize` will be fetched.	*2000*
2.	fallbackPageable	Specify default values for page location, number of records to be fetched and sort condition of the entire application. When page location, number of records to be fetched and sort condition are not specified, the values set in fallbackPageable are used.	Page location : *0* Number of records to be fetched : *20* Sort condition : *null*

次のページに続く

Table. 4.26 – 前のページからの続き

Sr. No.	Property name	Description	Default value
3.	oneIndexedParameters	Specify start value of page location. When *false* is specified, start value of page location becomes *0* and when *true* is specified, it becomes *1*.	*false*
4.	pageParameterName	Specify request parameter name to specify page location.	`"page"`
5.	sizeParameterName	Specify request parameter name to specify number of records to be fetched.	`"size"`
6.	prefix	Specify prefix (namespace) of request parameter to specify page location and number of records to be fetched. When there is a conflict between default parameter name and the parameter to be used in the application, it is recommended to specify namespace to avoid this issue. If prefix is specified, request parameter name to specify page location will be `prefix + pageParameterName` and request parameter name to specify number of records to be fetched will be `prefix + sizeParameterName`.	`""` (No namespace)
7.	qualifierDelimiter	To search multiple pages in the same request, specify request parameter name in `qualifier + delimiter + standard parameter name` format to distinguish the information required for page search (such as location of page to be searched, number of records to be fetched). For this property, set `delimiter` value of the above format. To change this setting, it is necessary to change the setting of `qualifierDelimiter` of `SortHandlerMethodArgumentResolver`.	`"_"`

Note: Setting value of maxPageSize

Default value is `2000`; however it is recommended to change the setting to maximum permissible value for the application. If maximum permissible value for the application is *100*, maxPageSize should also be set to *100*.

Note: Setting fallbackPageable

To change default values used in the entire application, set `Pageable` (`org.springframework.data.domain.PageRequest`) object wherein default value is defined in `fallbackPageable` property . To change the default sort condition, `org.springframework.data.domain.Sort` object where default value is defined in `fallbackSort` property of `SortHandlerMethodArgumentResolver`.

It is assumed that the fields given below will normally be changed in each application to be developed. The example to change the default values of such fields is given below.

- Maximum permissible value for number of records to be fetched (`maxPageSize`)
- Default values (`fallbackPageable`) of page location and number of records to be fetched in the entire application
- Default sort condition (`fallbackSort`)

```
<mvc:annotation-driven>
    <mvc:argument-resolvers>
        <bean
            class="org.springframework.data.web.
↪PageableHandlerMethodArgumentResolver">
            <!-- (1) -->
            <property name="maxPageSize" value="100" />
            <!-- (2) -->
            <property name="fallbackPageable">
                <bean class="org.springframework.data.domain.PageRequest">
                    <!-- (3) -->
                    <constructor-arg index="0" value="0" />
                    <!-- (4) -->
                    <constructor-arg index="1" value="50" />
                </bean>
            </property>
```

```
            <!-- (5) -->
            <constructor-arg index="0">
                <bean class="org.springframework.data.web.
↪SortHandlerMethodArgumentResolver">
                    <!-- (6) -->
                    <property name="fallbackSort">
                        <bean class="org.springframework.data.domain.Sort">
                            <!-- (7) -->
                            <constructor-arg index="0">
                                <list>
                                    <!-- (8) -->
                                    <bean class="org.springframework.data.
↪domain.Sort.Order">
                                        <!-- (9) -->
                                        <constructor-arg index="0" value="DESC
↪" />
                                        <!-- (10) -->
                                        <constructor-arg index="1" value=
↪"lastModifiedDate" />
                                    </bean>
                                    <!-- (8) -->
                                    <bean class="org.springframework.data.
↪domain.Sort.Order">
                                        <constructor-arg index="0" value="ASC
↪" />
                                        <constructor-arg index="1" value="id"␣
↪/>
                                    </bean>
                                </list>
                            </constructor-arg>
                        </bean>
                    </property>
                </bean>
            </constructor-arg>
        </bean>
    </mvc:argument-resolvers>
</mvc:annotation-driven>
```

Sr. No.	Description
(1)	In the above example, maximum value of number of records to be fetched is set to *100*. When value specified in number of records to be fetched (size) is *101* or more, search is performed for *100* records only.

次のページに続く

Table. 4.27 – 前のページからの続き

Sr. No.	Description
(2)	Create an instance of `org.springframework.data.domain.PageRequest` and set to `fallbackPageable`.
(3)	Specify default value of page location as the first argument of constructor of `PageRequest`. In the above example, *0* is specified, hence the default value is not changed.
(4)	Specify default value of number of records to be fetched as the second argument of constructor of `PageRequest`. In the above example, the value will be considered *50* when number of records to be fetched is not specified in request parameter.
(5)	Set an instance of `SortHandlerMethodArgumentResolver` as constructor of `PageableHandlerMethodArgumentResolver`.
(6)	Create an instance of `Sort` and set to `fallbackSort`.
(7)	Set the list of `Order` objects to be used as default value as the first argument of `Sort` constructor.
(8)	Create an instance of `Order` and add to the list of `Order` objects to be used as default value. In the above example, sort condition of "`ORDER BY x.lastModifiedDate DESC,x.id ASC`" is added to query when sort condition is not specified in request parameter.
(9)	Specify sort order (ASC/DESC) as the first argument of `Order` constructor.
(10)	Specify sort item as the second argument of `Order` constructor.

Property value of `SortHandlerMethodArgumentResolver`

Properties that can be specified in `SortHandlerMethodArgumentResolver` are as follows:

Values should be changed as required in the application.

Sr. No.	Property name	Description	Default value
1.	fallbackSort	Specify default sort condition for the entire application. When sort condition is not specified, the value set in fallbackSort is used.	*null* (No sort condition)
2.	sortParameter	Specify request parameter name to specify the sort condition. When there is a conflict between default parameter name and the parameter to be used in the application, it is recommended to change the request parameter name to avoid this issue.	`"sort"`
3.	propertyDelimiter	Specify delimiter of sort items and sort order (ASC,DESC).	`","`
4.	qualifierDelimiter	To search multiple pages in the same request, specify request parameter name in ‘‘ qualifier + delimiter + sortParameter ‘‘ format to distinguish the information required for page search (such as sort condition). For this property, set `delimiter` value of the above format.	`"_"`

4.5 Double Submit Protection

4.5.1 Overview

Problems

If any of the operations given below is performed in a Web application with screens, it leads to same process being executed multiple times.

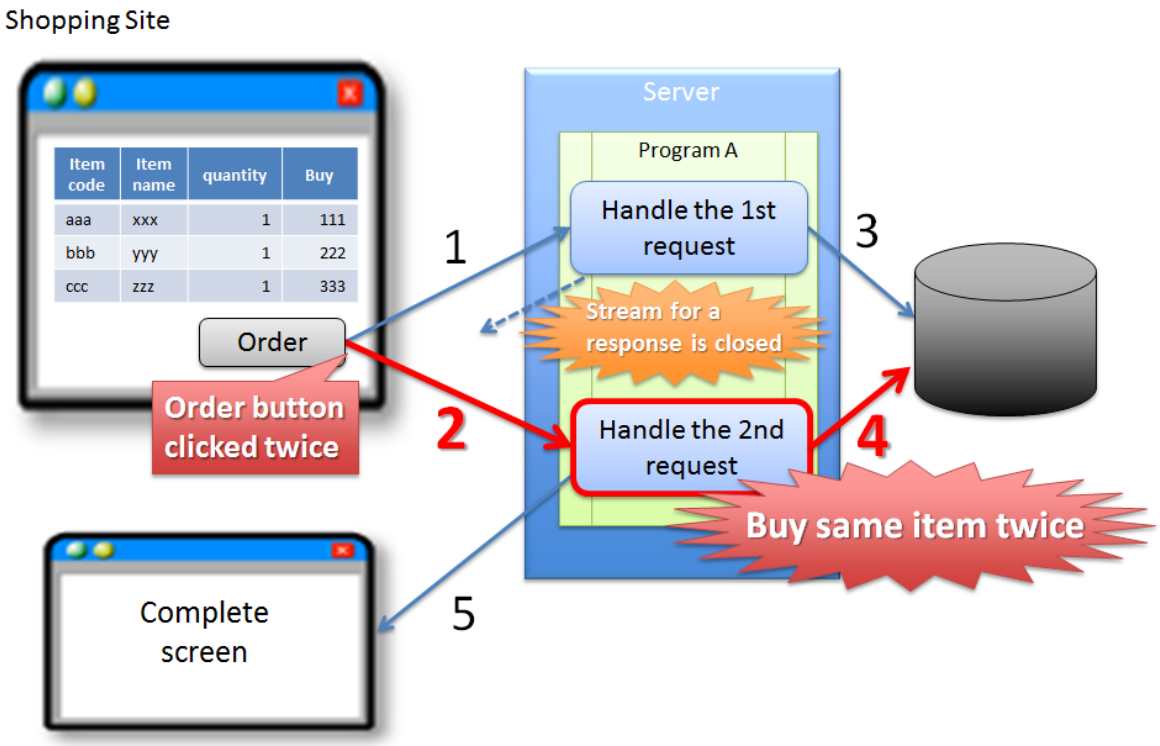
Sr. No.	Operation	Operation Overview
(1)	Double clicking of 'Update' button	Repeatedly clicking the button to perform update process.
(2)	Reloading of screen after completing the update process	Using the 'Refresh' button of the browser to reload the screen after completion of update process.
(3)	Invalid screen transition using 'Back' button of the browser	Using 'Back' button of the browser to go back to the previous page from update process completion screen and clicking the button again to perform update process.

Each problem is described in detail below.

Double clicking of 'Update' button

The following problems occur when the button to perform update process is clicked repeatedly.

The example of "Product Purchase" on a shopping site is given below to explain the problems that are likely to occur if the necessary measures are not taken.



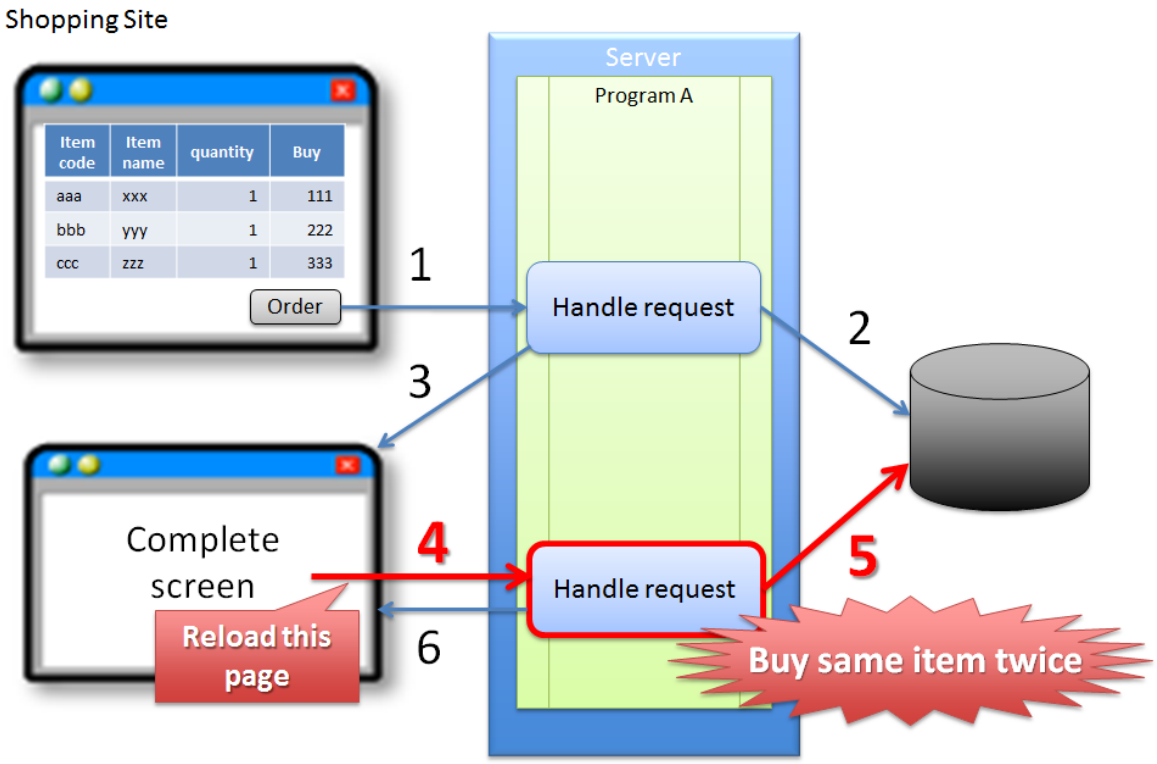
Sr. No.	Description
(1)	Buyer clicks ‘Order’ button on Product Purchase screen.
(2)	Buyer accidentally clicks ‘Order’ button again before receiving the response of (1).
(3)	Server updates DB based on purchase of the product received through request (1).
(4)	Server updates DB based on purchase of the product received through request (2).
(5)	Server sends response with a “Purchase Complete” screen for the product received through request (2).

> **Warning:** In the above case, since buyer accidentally clicks 'Order' button again, **it results in duplicate purchase of same product.** Although the problem can be attributed to erroneous operation by the buyer, it is desirable to have the application design such that the above problems do not occur.

Reloading of screen after completion of update process

The following problems occur when the screen is reloaded after completion of update process.

The example of "Product Purchase" on a shopping site is given below to explain the problems that are likely to occur if the necessary measures are not taken.



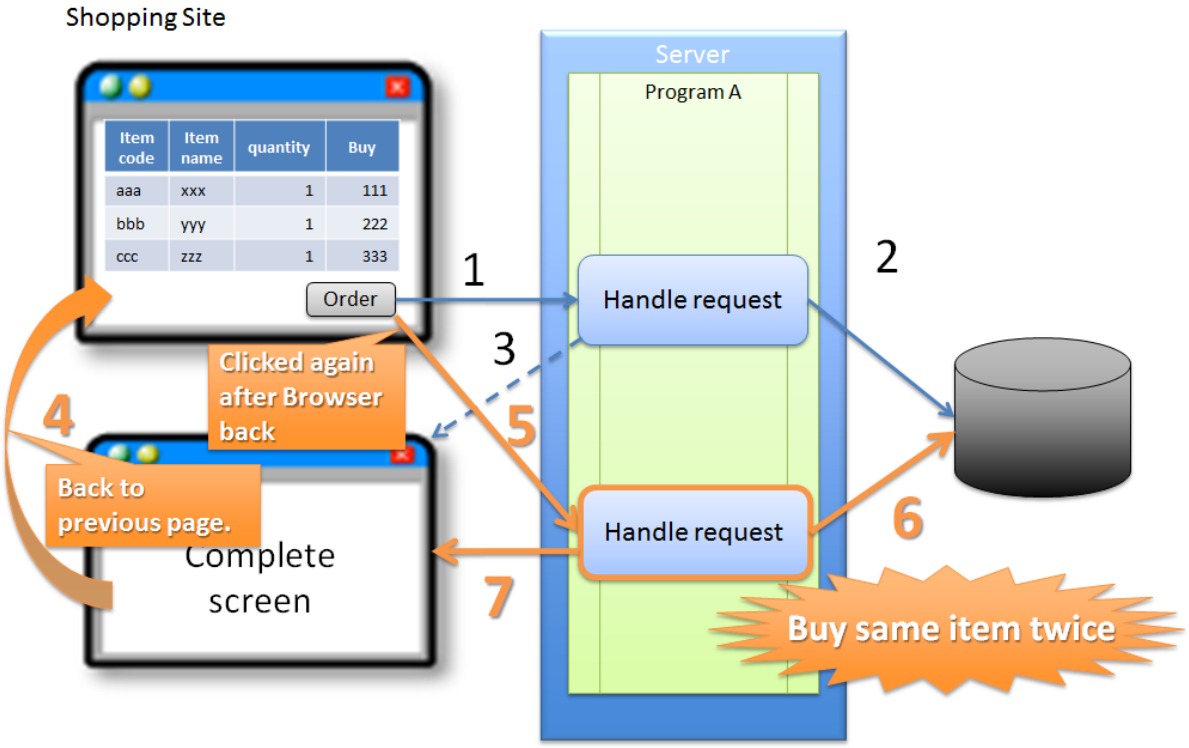
Sr. No.	Description
(1)	Buyer clicks 'Order' button on "Product Purchase" screen.
(2)	Server updates DB based on purchase of the product received through request (1).
(3)	Server sends response with a "Purchase complete" screen for the product received through request (1).
(4)	Buyer accidently executes Reload functionality of the browser.
(5)	Server updates DB based on the purchase of the product received through request (4).
(6)	Server sends response with a "Purchase Complete" screen for the product received through request (4).

Warning: In the above case, since buyer accidentally executes Reload functionality of the browser, **it results in duplicate purchase of same product.** Although the problem can be attributed to erroneous operation by the buyer, it is desirable to have the application design such that the above problems do not occur.

Invalid screen transition using 'Back' button of the browser

The following problems occur if invalid screen transition is performed using 'Back' button of the browser.

The example of "Product Purchase" on a shopping site is given below to explain the problems that are likely to occur if the necessary measures are not taken.



Sr. No.	Description
(1)	Buyer clicks ‘Order’ button on “Product Purchase” screen.
(2)	Server updates DB based on the purchase of the product received through request (1).
(3)	Server sends response with a “Purchase complete” screen of the product received through request (1).
(4)	Buyer uses ‘Back’ button of the browser to go back to “Product Purchase” screen.
(5)	Buyer again clicks ‘Order’ button on “Product Purchase” screen which is re-displayed by clicking ‘Back’ button of the browser.

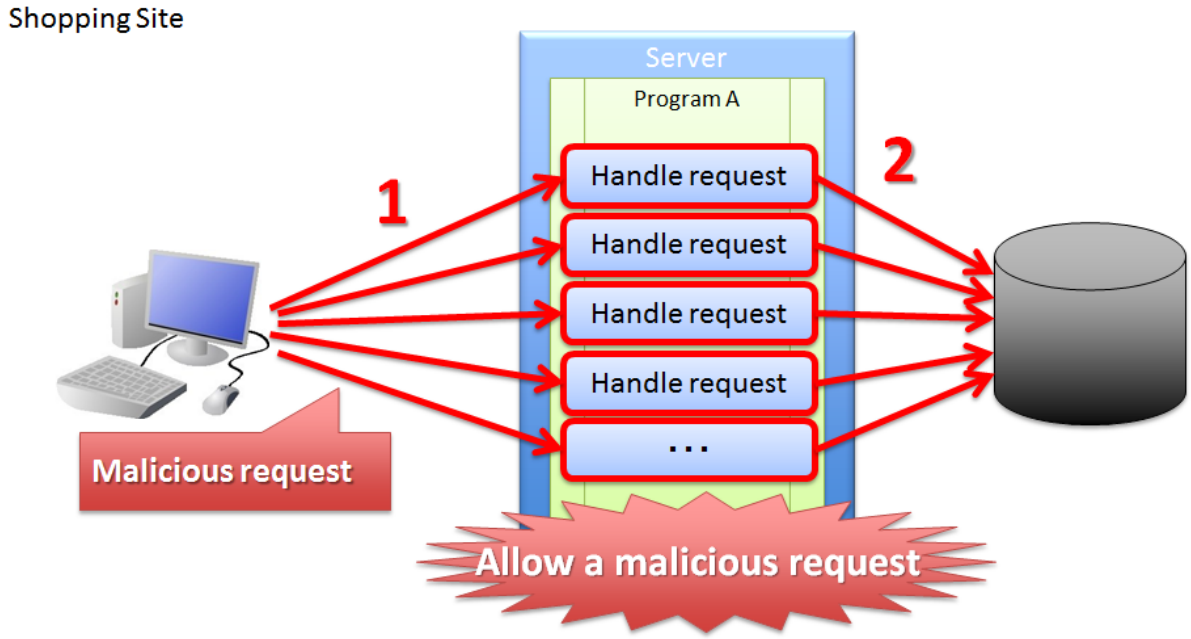
次のページに続く

Table. 4.28 – 前のページからの続き

Sr. No.	Description
(6)	Server updates DB based on the purchase of the product received through request (5).
(7)	Server sends response with a “Purchase Complete” screen for the product received through request (5).

Note: In the above case, since buyer does not perform any erroneous operation, the problem is not attributed to the buyer.

However, if update process gets executed even after performing invalid screen operations, the following problems occur.



Warning: As described above, update process getting executed even after performing invalid screen operations increases the risk of direct updates by a malicious attacker bypassing a valid route.

Sr. No.	Description
(1)	Attacker executes request for processing a direct product purchase without going through a valid screen transition.

(2)	Server cannot detect that the request is getting executed through an invalid route; hence it updates DB based on the purchase of the product received through that request.

Execution of purchase process through an invalid request increases the load on each server resulting in inability to purchase products through a valid route. As a result, the problem causes a ripple effect for the users who purchase the products through valid routes. Hence, it is desirable to have the application design such that the above problems do not occur.

Solutions

The following measures should be taken to resolve the problems described above.

In view of malicious operations such as tampering with requests, **(3) "Applying transaction token check" is mandatory.**

Sr. No.	Solution	Overview
(1)	Preventing double clicking of a button using JavaScript	When a button to perform update process is clicked, the button control using JavaScript prevents the submission of a request if the button is clicked again.
(2)	Applying PRG (Post-Redirect-Get) pattern	A redirect command is returned as a response to the request for performing update process (request by POST method) and then a screen for transition is returned as a response of GET method which is automatically requested from a browser. When a PRG pattern is used, the request generated while reloading the page after the screen is displayed is a GET method; hence re-execution of update process can be prevented.
(3)	Applying transaction token check	Issue a token value for each screen transition and compare the token value sent from browser with the token value stored on the Server to make sure that invalid screen operations do not occur in the transaction. Implementation of transaction token check can prevent re-execution of update process after the page is reloaded using 'Back' button of the browser. Deleting the token value stored on the Server after performing the token check can prevent double submission as a Server side process.

Note: When only transaction token check is performed, even a simple operational mistake can lead to transaction token error which in turn results in a low-usability application for the user.

To ensure usability as well as to prevent the problems that occur due to double submission, measures such as "Preventing double clicking of a button using JavaScript" and "Applying PRG (Post-Redirect-Get) pattern" are necessary.

** Although this guideline recommends that you implement all the measures, the decision should be taken depending on application requirements.**

Warning: In Ajax and Web services, since it is difficult to transfer transaction tokens which change for each request, transaction token check need not be used. In Ajax, double submit protection should be performed using only one of the above measures i.e. "Preventing double clicking of a button using

JavaScript".

Todo

TBD

There is further scope for reviewing the check methods in Ajax and Web services.

Preventing double clicking of a button using JavaScript

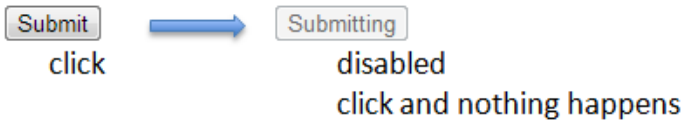
Prevent double clicking of buttons like button to perform update process or button which is used to perform time-consuming search process.

When a button is clicked, use JavaScript to disable that button or link.

Typical examples of control used for disabling a button or link are given below

1. By disabling the button or link so that it cannot be clicked
2. By maintaining a flag for tracking process status and displaying notification "Process in progress" when the button or link is clicked in the middle of the process.

The image when a button is disabled will be as follows:

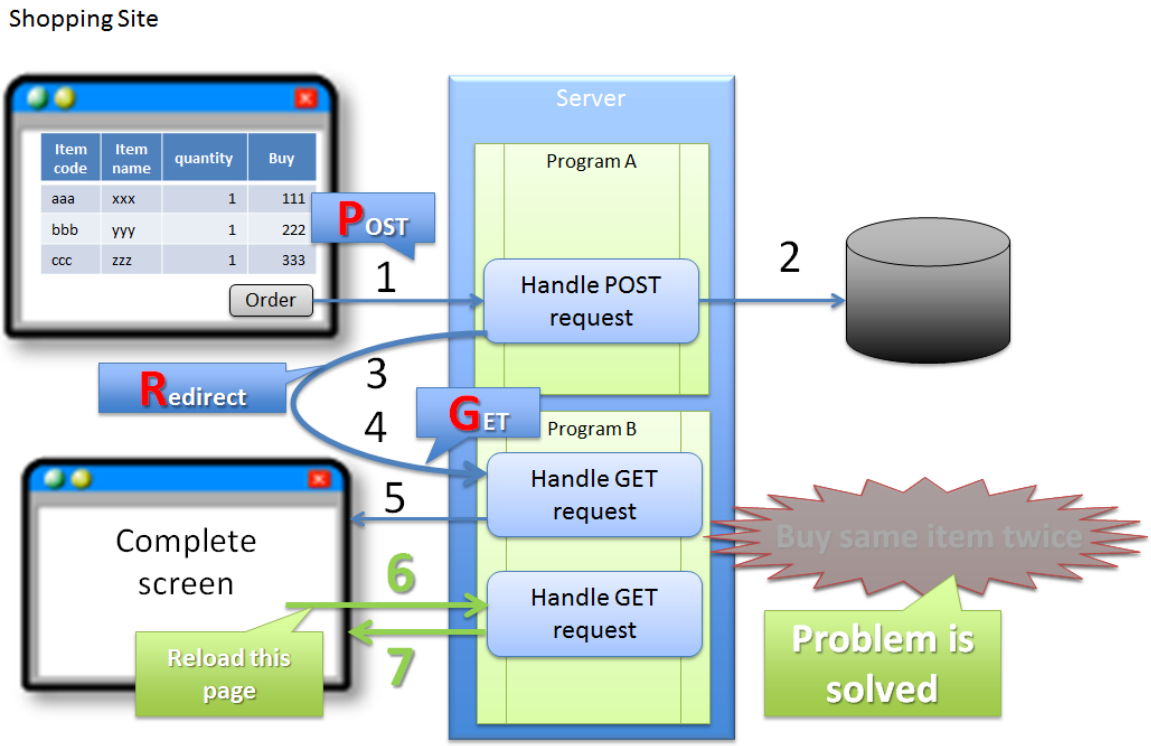


Warning: If all the buttons and links on the screen are disabled, the screen operations can no longer be performed if there is no response from the Server. Therefore, it is recommended not to disable buttons or links that execute events such as "Return to previous screen" or "Go to top screen" etc.

About PRG (Post-Redirect-Get) pattern

A redirect command is returned as a response to the request for performing update process (request by POST method) and then a screen for transition is returned as a response of GET method which is automatically requested from a browser.

When a PRG pattern is used, the request generated while reloading the page after the screen is displayed is a GET method; hence re-execution of update process can be prevented.


Shopping Site
Item code
Item name
quantity
Buy
aaa
xxx
1
111
bbb
yyy
1
222
ccc
zzz
1
333
Order
POST
1
Server
Program A
Handle POST request
2
Redirect
3
4
GET
Program B
Handle GET request
5
Complete screen
6
Handle GET request
Reload this page
7
Buy same item twice
Problem is solved

Sr. No.	Description
(1)	Buyer clicks 'Order' button on "Product Purchase" screen. **The request is submitted using POST method.**
(2)	Server updates DB based on the purchase of the product received through request (1).
(3)	**Server sends a redirect response for the URL to display the "Purchase Complete" screen for the product.**
(4)	Browser submits request for the URL to display the "Purchase Complete" screen for the product. **The request is submitted using GET method.**
(5)	Server sends response with a "Purchase Complete" screen for the product.
(6)	Buyer accidentally executes Reload functionality of the browser. The request called by Reload functionality displays "Purchase Complete" screen of the product; hence **update process is not re-executed.**
(7)	Server sends response with a "Purchase Complete" screen.

Note: It is recommended to use PRG pattern for the processes associated with update process and implement a control so that a request of GET method is sent when 'Refresh' button of the browser is clicked.

Warning: In the PRG pattern, the re-execution of update process cannot be prevented by clicking 'Back' button of the browser on Completion screen. A transaction token check must be performed to prevent re-execution of update process after an invalid screen transition using 'Back' button of the browser.

Transaction Token Check

Transaction token check consists of the following 3 processes.

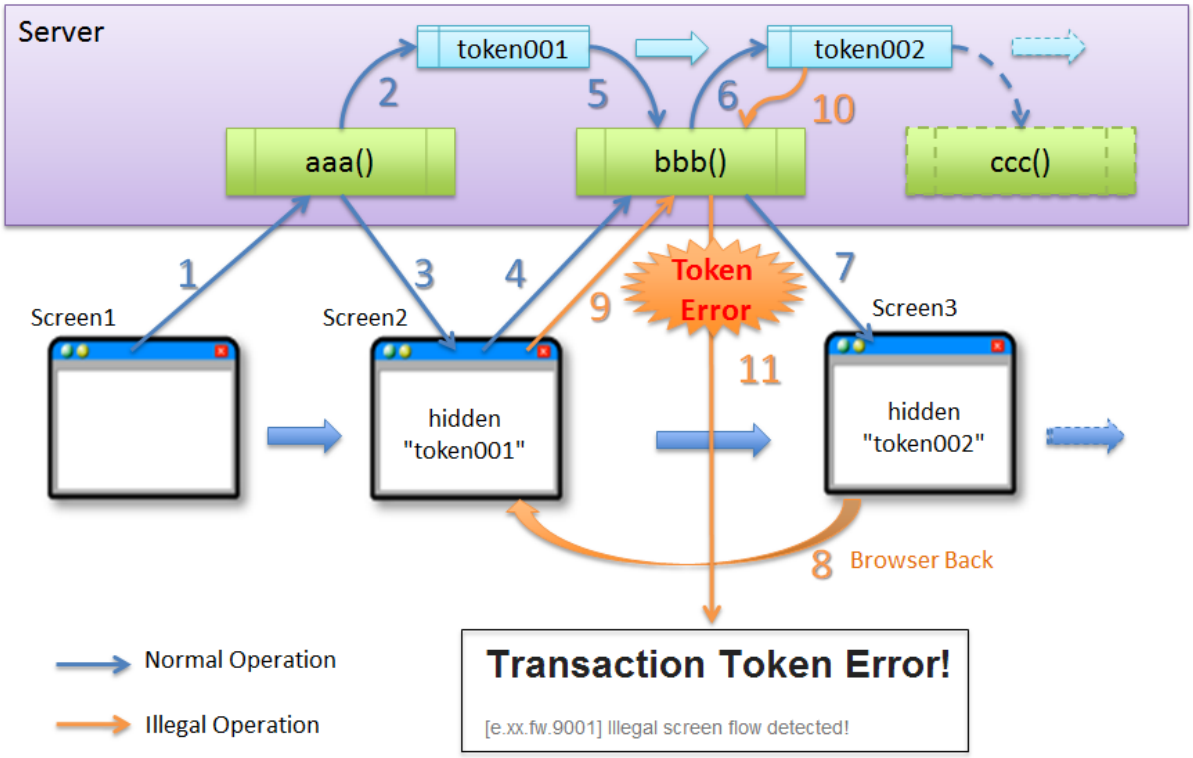
- When a request is received from Client, the Server stores a value (hereafter referred to as transaction token) for uniquely identifying a transaction on the Server.
- Server passes the transaction token to the Client. In case of a Web application with screens, it passes the transaction token to the Client using hidden tag of form.
- When submitting the next request, Client sends the transaction token that was passed from the Server. Server compares the transaction token received from the Client and the transaction token stored on the Server.

When the transaction token sent in the request does not match with the transaction token stored on the Server, it is treated as invalid request and an error is returned.

> **Warning:** Misuse of transaction token check leads to poor usability of the application; hence the scope of its usage should be defined by considering the following points.
>
> - It is not necessary to include reference-type requests that do not involve data update and requests that perform only screen transitions in the scope of transaction token check.
>
>   If the scope of transactions is extended unnecessarily, transaction token errors are more likely to occur which in turn reduces the usability of the application.
> - Transaction token check is not mandatory for the processes wherein there is no problem even if the data gets updated multiple times from business perspective (user information update etc.).
> - Transaction token check is mandatory for the processes such as deposit process or product purchase process etc. wherein there is a risk of duplicate execution.

The process flow when expected operations are performed and process flow when unexpected operations are performed using transaction token check are shown below.

The process flow when expected operations are performed is as follows:



Sr. No.	Description
(1)	Client sends the request.
(2)	Server creates the transaction token (token001) and stores it on the Server.
(3)	Server passes the created transaction token (token001) to the Client.
(4)	Client sends the request along with the transaction token (token001).
(5)	Server checks whether the transaction token (token001) stored on the Server and the transaction token (token001) submitted by the Client are same. **Since the values are same, the request is considered as valid.**

次のページに続く

Table. 4.30 – 前のページからの続き

Sr. No.	Description
(6)	Server generates transaction token (token002) to be used in the next request and updates the value stored on the Server. At this point, the transaction token (token001) is discarded.
(7)	Server passes the updated transaction token (token002) to the Client.

The process flow when unexpected operations are performed is as follows:

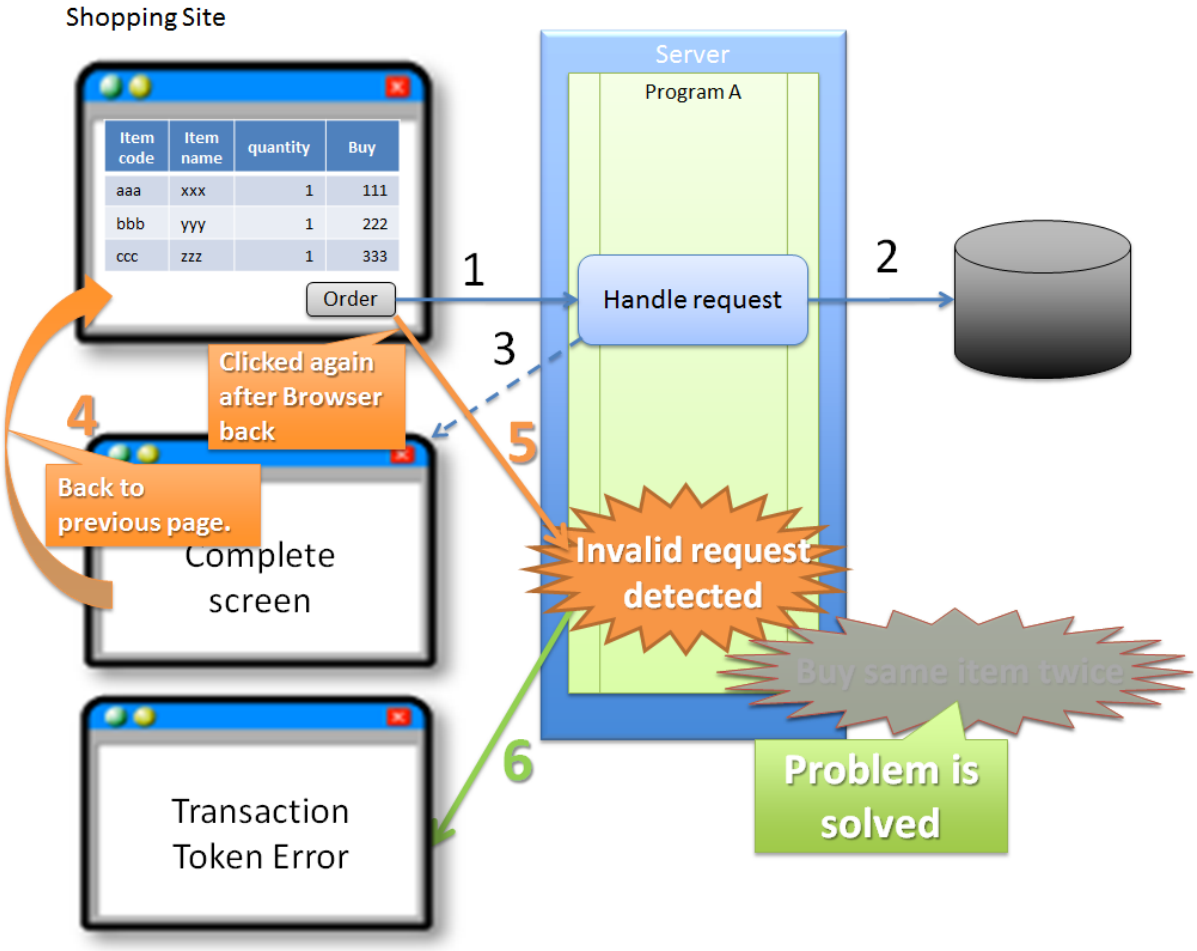
Here, 'Back' button of the browser is taken as an example; however, this is also applicable for the direct requests from shortcuts etc.

Sr. No.	Description
(8)	'Back' button of the browser on Client side is clicked.
(9)	Request is sent from Client side along with the transaction token (token001) of the screen which is displayed after clicking 'Back' button.
(10)	Server checks whether the transaction token (token002) stored on the Server and the transaction token (token001) submitted by the Client are same. **Since the values are different, the request is considered as invalid and a transaction token error is thrown.**
(11)	Server sends response with an error screen to notify that a transaction token error has occurred.

The 3 events described below can be prevented by transaction token check.

- Invalid screen transition in case of a business process that requires fixed screen transition
- Data update due to invalid requests that do not involve valid screen transitions
- Duplicate execution of update process due to double submission

Invalid screen transition in case of a business process that requires fixed screen transition, can be prevented by the flow shown below.



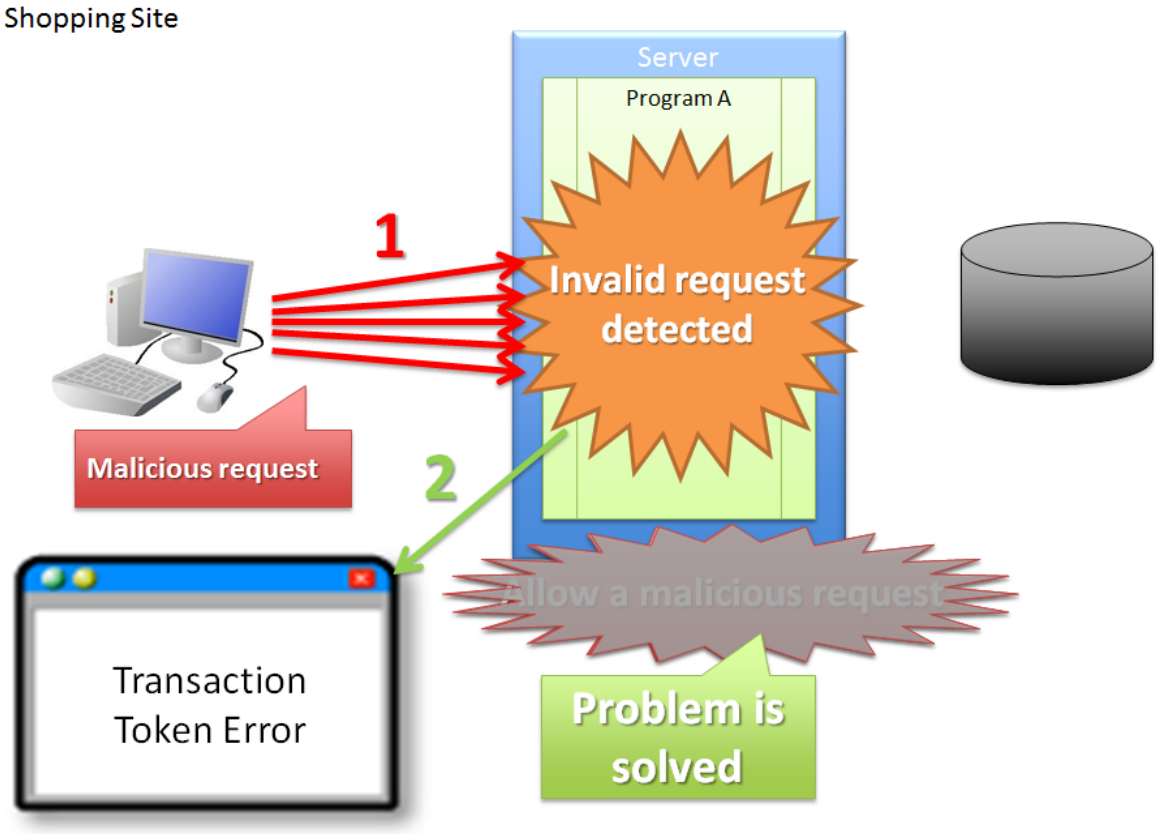
Sr. No.	Description
(1)	Buyer clicks 'Order' button on "Product Purchase" screen. Since the transaction token stored on the Server and the transaction token submitted by the Client match, the process to purchase the product is executed. **At this time, the value of the transaction token stored on the Server is discarded and updated to a new token value.**
(2)	Server updates DB based on the purchase of the product received through request (1).

次のページに続く

Table. 4.31 – 前のページからの続き

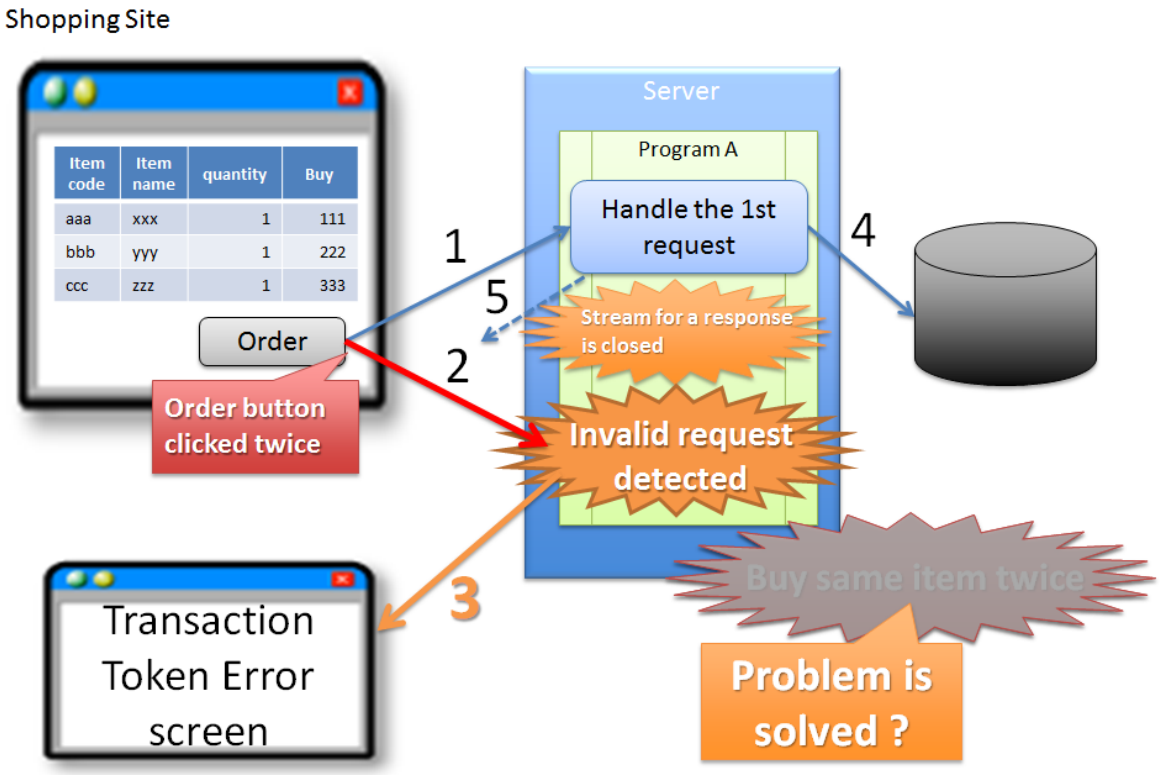
Sr. No.	Description
(3)	Server sends response with a “Purchase Complete” screen for the product received through request (1).
(4)	Buyer uses ‘Back’ button of the browser to go back to “Product Purchase” screen.
(5)	Buyer again clicks ‘Order’ button on “Product Purchase” screen which is displayed using ‘Back’ button of the browser. **Since the transaction token sent by the Client is a value which has already been discarded, a transaction token error occurs.**
(6)	Server sends response with an error screen to notify that a transaction token error has occurred.

Data updated by an invalid request which does not involve a valid screen transition can be prevented by the flow shown below.



Sr. No.	Description
(1)	Attacker sends a request to purchase the product directly without performing a valid screen transition. **Since the request for generating a transaction token is not executed, a transaction token error occurs.**
(2)	Server sends response with an error screen to notify that a transaction token error has occurred.

Duplicate execution of update process at the time of double submission can be prevented by the flow shown below.



Sr. No.	Description
(1)	Buyer clicks 'Order' button on "Product Purchase" screen. Since the transaction token stored on the server and the transaction token submitted by the Client match, the process to purchase the product is executed. **At this time, the value of the transaction token that is stored on the server is discarded and updated to a new token value.**
(2)	Buyer accidentally clicks 'Order' button again before the response of (1) is returned. **Since the transaction token sent by the Client is a value which has already been discarded, a transaction token error occurs** when process of (1) is executed.
(3)	Server **sends response with an error screen to notify that a transaction token error has occurred** for the request (2).
(4)	Server updates DB based on the purchase of the product received through request (1).
(5)	Server attempts to respond with a "Purchase Complete" screen for the product received through request (1); however since the stream for responding to the request of (1) is closed due to the transmission of the request of (2), it fails to send response with a "Purchase Complete" screen.

Warning: This can prevent duplicate execution of update process at the time of double submission; however this does not resolve the problem of server not being able to respond with a screen to notify the completion of process. Therefore, it is also recommended to deal with this problem by preventing double clicking of a button using JavaScript.

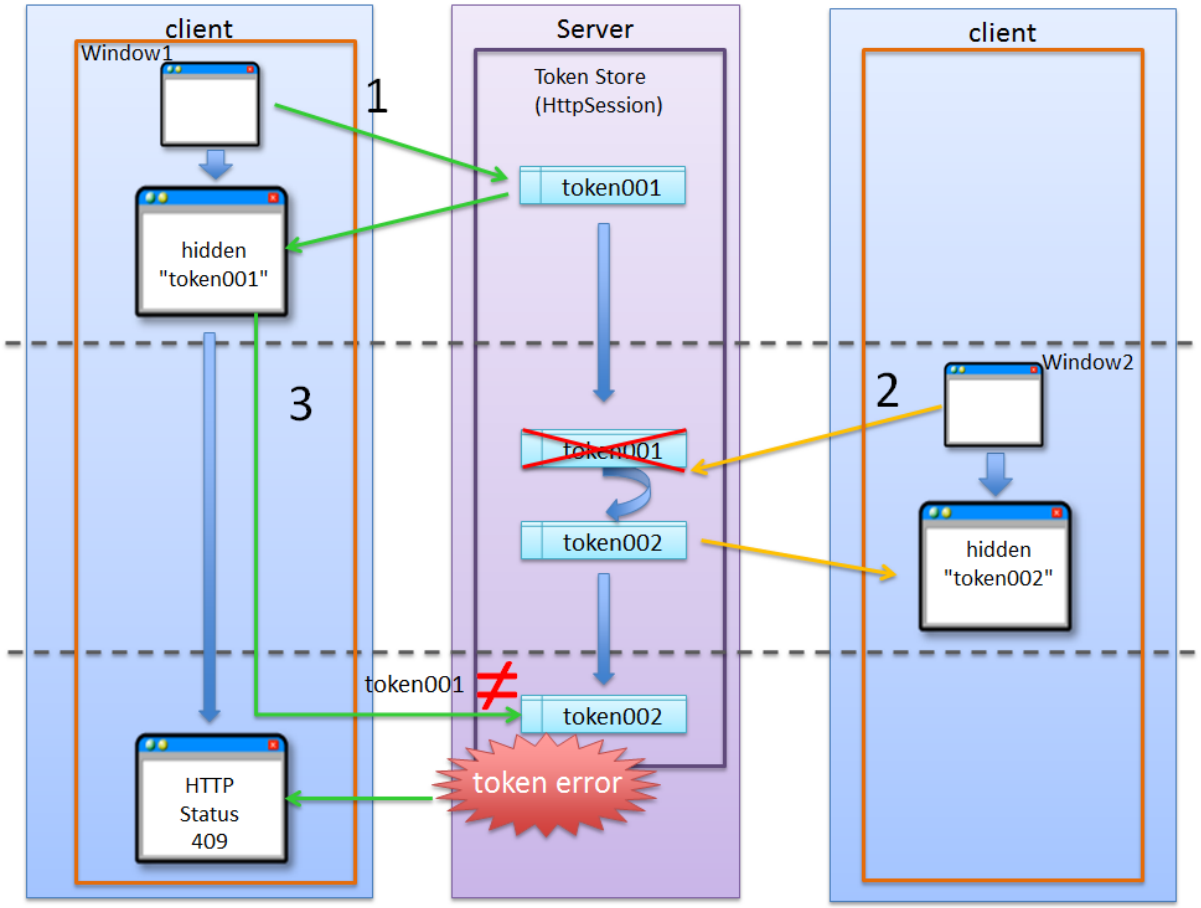
About NameSpace of transaction token

In the transaction token check functionality provided by the common library, it is possible to set a NameSpace in the container for storing transaction token. This enables parallel execution of update process using a tab browser or multiple windows.

Problems that occur when there is no NameSpace

Let us see the problems that occur when there is no NameSpace.

The figure below illustrates an example wherein two clients are arranged side by side; basically 2 windows are launched on same machine.



Sr. No.	Description
(1)	The request is sent from Window 1, then the transaction token (token001) received as a response is stored in the browser. The transaction token stored on the server is considered token001.
(2)	The request is sent from Window 2, then the transaction token (token002) received as a response is stored in the browser. **The transaction token stored on the server is considered token002. The transaction token (token001) generated by the process (1) is discarded at this point.**
(3)	The request is sent from Window 1 along with the transaction token (token001) stored in the browser. Since the transaction token stored on the server (token002) and the transaction token sent by the request (token001) do not match, the request is considered as invalid resulting in a transaction token error.

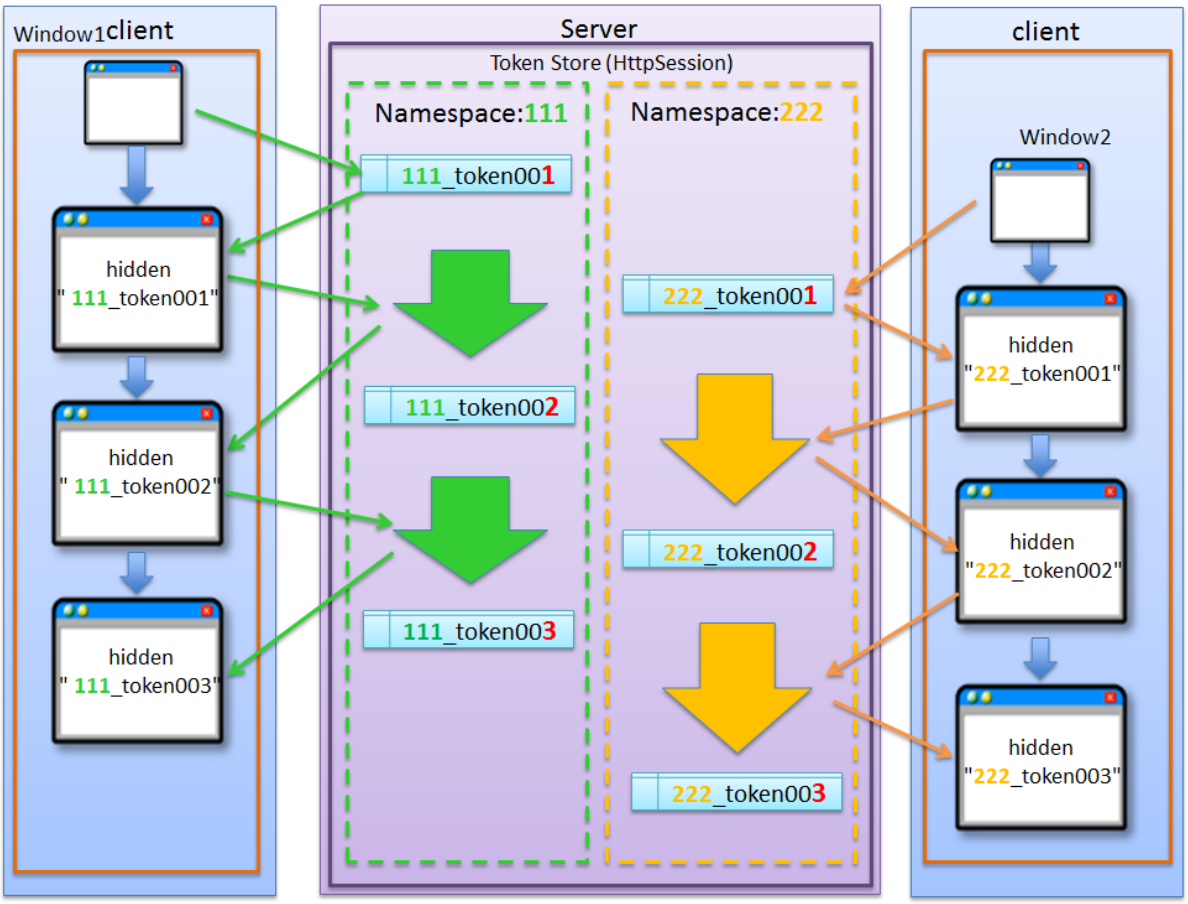
Warning: The update process cannot be executed concurrently in absence of NameSpace; hence the application becomes a low-usability application.

Behavior when NameSpace is specified

Let us now see the behavior when NameSpace is specified.

The problem wherein the update process could not be executed concurrently in the absence of NameSpace can now be resolved by specifying a NameSpace.

The figure below illustrates an example wherein two clients are arranged side by side; basically 2 windows are launched on same machine.



NameSpaces are shown as 111, 222 in the figure.

** When a NameSpace is specified, the transaction token in the NameSpace allocated to the transaction is handled independently. Hence, it does not affect the transactions of another NameSpace.**

Here, even though the browser is explained using different Windows, the tab browser works in the same way. For generated keys and usage method, refer to *Using transaction token check*.

4.5.2 How to use

Preventing double clicking of button using JavaScript

Double clicking of a button on Client side can be prevented using JavaScript.

Once a button is clicked, it should not be possible to click it again till it is re-generated.

Todo

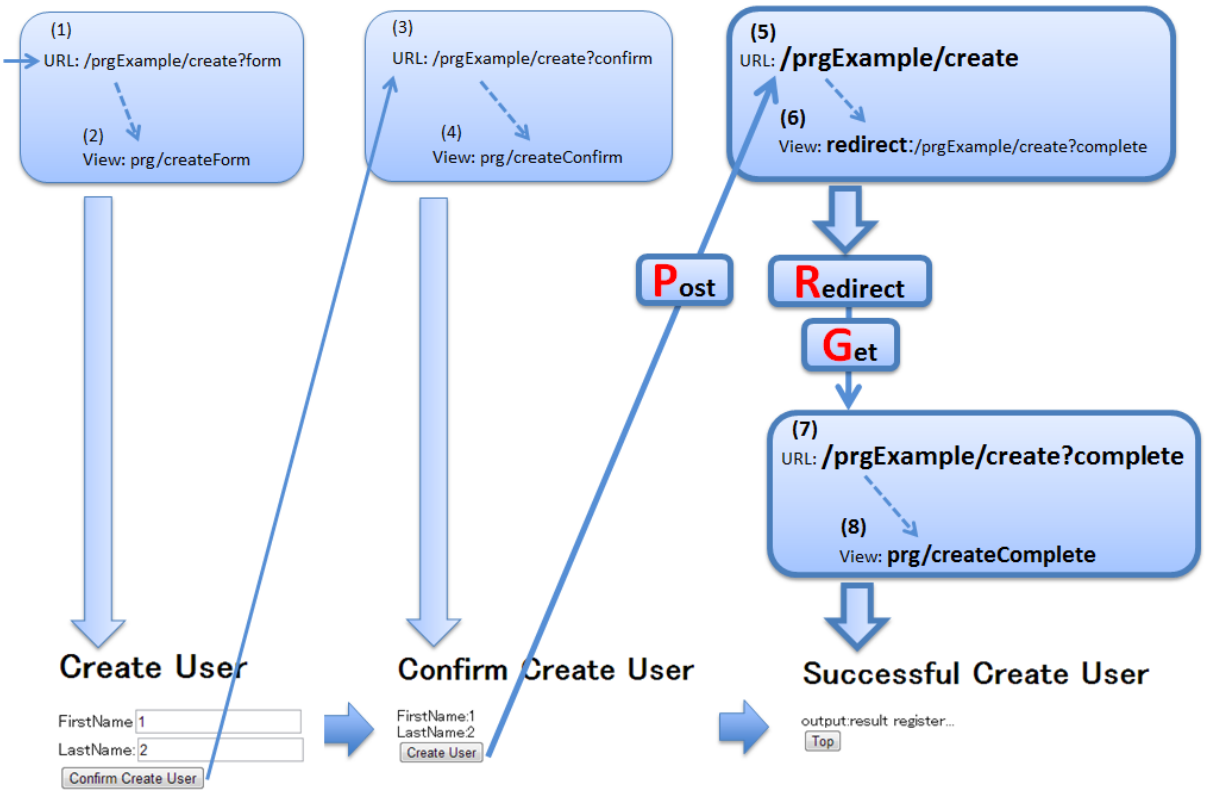
TBD

The check method in JavaScript will be described in detail in subsequent versions.

Using PRG (Post-Redirect-Get) pattern

The example of implementing PRG (Post-Redirect-Get) pattern is given below.

The application which involves a simple screen transition such as Input Screen-> Confirmation Screen-> Completion Screen is taken as an example.



The image numbers and comment number of source code are linked.

However, since (1)-(4) is not directly related to the PRG pattern, the explanation is omitted.

- Controller

```
@Controller
@RequestMapping("prgExample")
public class PostRedirectGetExampleController {

    @Inject
    UserService userService;

    @ModelAttribute
    public PostRedirectGetForm setUpForm() {
        PostRedirectGetForm form = new PostRedirectGetForm();
        return form;
    }

    @RequestMapping(value = "create",
                    method = RequestMethod.GET,
                    params = "form") // (1)
    public String createForm(
        PostRedirectGetForm postRedirectGetForm,
        BindingResult bindingResult) {
        return "prg/createForm"; // (2)
    }

    @RequestMapping(value = "create",
                    method = RequestMethod.POST,
                    params = "confirm") // (3)
    public String createConfirm(
        @Validated PostRedirectGetForm postRedirectGetForm,
        BindingResult bindingResult) {
        if (bindingResult.hasErrors()) {
            return "prg/createForm";
        }
        return "prg/createConfirm"; //  (4)
    }

    @RequestMapping(value = "create",
                    method = RequestMethod.POST) // (5)
    public String create(
        @Validated PostRedirectGetForm postRedirectGetForm,
        BindingResult bindingResult,
        RedirectAttributes redirectAttributes) {
        if (bindingResult.hasErrors()) {
            return "prg/createForm";
        }

        // omitted

        String output = "result register..."; // (6)
        redirectAttributes.addFlashAttribute("output", output); // (6)
        return "redirect:/prgExample/create?complete"; // (6)
    }
```

```
    @RequestMapping(value = "create",
                    method = RequestMethod.GET,
                    params = "complete") // (7)
    public String createComplete() {
        return "prg/createComplete"; // (8)
    }
}
```

Sr. No.	Description
(5)	A handler method to perform a process when 'Register' button (Create User button) on Confirmation screen is clicked. **The request is received by POST method.**
(6)	**It is redirected to URL for displaying Completion screen.** In the above example, a request is sent to URL "`prgExample/create?complete`" by `GET` method. When data is to be delivered to redirect destination, addFlashAttribute method of `RedirectAttributes` is called and the data to be delivered is added. addAttribute method of `Model` cannot deliver data to the redirect destination.
(7)	A handler method to display Completion screen. **A request is received by GET method.**
(8)	View (JSP) is called to display the Completion screen and responds with Completion screen. Since the extension of JSP is assigned by `ViewResolver` defined in `spring-mvc.xml`, it is omitted from the return value of the handler method.

Note:

- At the time of redirecting, assign "redirect:" as the prefix of transition information to be returned by the handler method as the return value.
- When the data is to be delivered to the process of redirect destination, call addFlashAttribute method of `RedirectAttributes` and add the data to be delivered.

- createForm.jsp

```
<h1>Create User</h1>
<div id="prgForm">
  <form:form
    action="${pageContext.request.contextPath}/prgExample/create"
    method="post" modelAttribute="postRedirectGetForm">
    <form:label path="firstName">FirstName</form:label>
    <form:input path="firstName" /><br>
    <form:label path="lastName">LastName:</form:label>
    <form:input path="lastName" /><br>
    <form:button name="confirm">Confirm Create User</form:button>
  </form:form>
</div>
```

- createConfirm.jsp

```
<h1>Confirm Create User</h1>
<div id="prgForm">
  <form:form
    action="${pageContext.request.contextPath}/prgExample/create"
    method="post"
    modelAttribute="postRedirectGetForm">
    FirstName:${f:h(postRedirectGetForm.firstName)}<br>
    <form:hidden path="firstName" />
    LastName:${f:h(postRedirectGetForm.lastName)}<br>
    <form:hidden path="lastName" />
    <form:button>Create User</form:button> <%-- (6) --%>
  </form:form>
</div>
```

Sr. No.	Description
(6)	When the button to perform update process is clicked, **a request is sent by POST method.**

- createComplete.jsp

```
<h1>Successful Create User Completion</h1>
<div id="prgForm">
  <form:form
    action="${pageContext.request.contextPath}/prgExample/create"
    method="get" modelAttribute="postRedirectGetForm">
    output:${f:h(output)}<br> <%-- (7) --%>
    <form:button name="backToTop">Top</form:button>
  </form:form>
</div>
```

Sr. No.	Description
(7)	When the data delivered from update process is to be referred at the redirect destination, **specify the attribute name of the data added by the addFlashAttribute method** of `RedirectAttributes`. In the above example, `"output"` is the attribute name to refer to the delivered data.

Using transaction token check

The example of implementation using transaction token check is given below.

Transaction token check functionality is provided by the common library and not by Spring MVC.

Transaction token check provided by common library

Transaction token check functionality of common library provides `@org.terasoluna.gfw.web.token.transaction.Tr` annotation to perform the following tasks:

- Creation of NameSpace for transaction token
- Starting the transaction
- Token value check in the transaction
- Ending the transaction

The transaction token check can be performed declaratively by assigning `@TransactionTokenCheck` annotation for the Controller class and the handler methods of the Controller class.

Attributes of `@TransactionTokenCheck` annotation

The attributes that can be specified in `@TransactionTokenCheck` annotation are explained below.

Table.4.32: `@TransactionTokenCheck`Annotation Parameter List

Sr. No.	Attribute Name	Contents	default	Example
1.	value	Any character string. Used as NameSpace.	None	`value = "create"` `value =` part can be omitted if there is only one argument.
2.	namespace	Any string.Used as NameSpace. It is an alias of value attribute.	None	`namespace = "create"` Synonymous with `value = "create"`. It is used as an alternative to value attribute since value attribute cannot be used while using `@TransactionTokenCheck` as a meta-annotation.

次のページに続く

Table. 4.32 – 前のページからの続き

Sr. No.	Attribute Name	Contents	default	Example

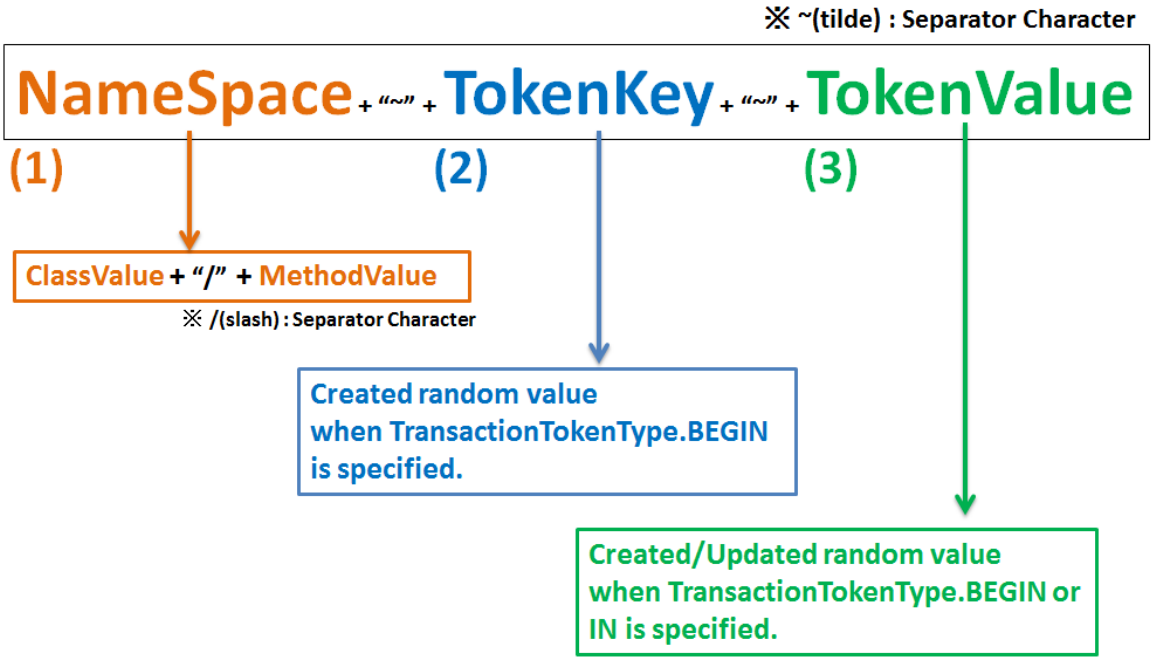
3.	type	**BEGIN**
		A transaction token is created and a new transaction is started.
		
		IN
		Transaction token is validated.
		When the requested token value and the token value stored on the server match, the token value of transaction token is updated.
		
		CHECK
		Transaction token is validated.
		Even when the requested token value and the token value stored on the server match,the token value of transaction token is not updated.
		For used cases, refer "*When a process which does not update screen for file downloading process is included in the use case*".
			IN	`type = TransactionTokenType.BEGI`
				
				`type = TransactionTokenType.IN`
				
				`type = TransactionTokenType.CHEC`

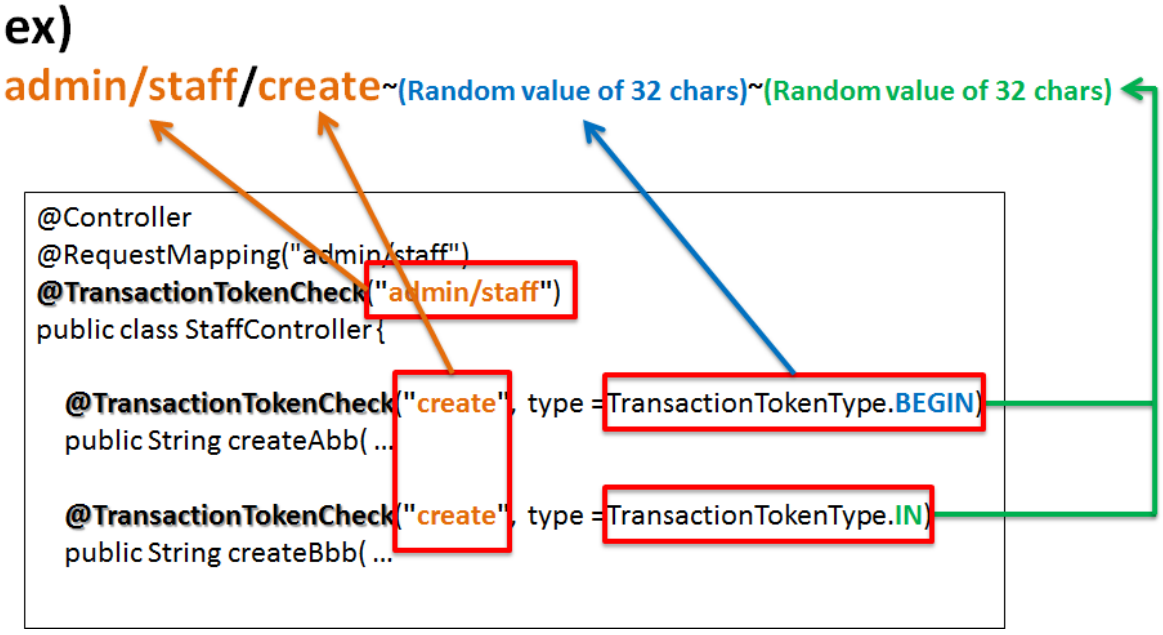
Note: It is recommended that the value to be set in value attribute or namespace attribute should be same as the config value of "value" attribute for `@RequestMapping` annotation.

Note: In "type" attribute, **NONE** and **END** can be specified; however, the description is omitted as normally they are not used.

Format of transaction token

Format for the transaction token used in the transaction token check of common library is as follows:


※ ~(tilde) : Separator Character
NameSpace + "~" + TokenKey + "~" + TokenValue
(1)
(2)
(3)
ClassValue + "/" + MethodValue
※ /(slash) : Separator Character
Created random value
when TransactionTokenType.BEGIN
is specified.
Created/Updated random value
when TransactionTokenType.BEGIN or
IN is specified.


ex)
admin/staff/create~(Random value of 32 chars)~(Random value of 32 chars)
@Controller
@RequestMapping("admin/staff")
@TransactionTokenCheck("admin/staff")
public class StaffController{
@TransactionTokenCheck("create", type =TransactionTokenType.BEGIN)
public String createAbb(...
@TransactionTokenCheck("create", type =TransactionTokenType.IN)
public String createBbb(...

Sr. No.	Components	Description
(1)	NameSpace	• NameSpace is an element for assigning a logical name to identify a series of screen transitions. • By setting a NameSpace, it is possible to prevent the requests belonging to different Namespaces from interfering with each other and it is also possible to increase the number of screen transitions that can operate in parallel. • The value specified in the "value" attribute of `@TransactionTokenCheck` annotation is used as the value to be used for NameSpace. • When both "value" attribute of the "class" annotation and "value" attribute of the "method" annotation are specified, the value which concatenates both the values with `"/"` is used as NameSpace. When the same value is specified in multiple methods, the methods belong to same NameSpace. • When the "value" attribute is specified only in "class" annotation, all the NameSpaces of the transaction tokens generated in that class will be the value specified in "class" annotation. • When the "value" attribute is specified only in "method" annotation, the NameSpace of the generated transaction tokens will be the value specified in "method" annotation. When the same value is specified in multiple methods, the methods belong to same NameSpace. • When both "value" attribute of "class" annotation and "value" attribute of "method" annotation are omitted, the method belong to the global token. For global token, refer to *Global Tokens*.

次のページに続く

Table. 4.33 – 前のページからの続き

Sr. No.	Components	Description
(2)	TokenKey	• TokenKey is an element for identifying the transactions stored in the Namespace. • TokenKey is generated upon execution of a method wherein `TransactionTokenType.BEGIN` is declared in the “type” attribute of `@TransactionTokenCheck` annotation. • A maximum limit exists for the number of multiple TokenKeys which can be concurrently stored and the default number is 10. The count of stored TokenKeys is managed for each NameSpace. • When the number of TokenKeys stored for each NameSpace reaches the maximum value at the time of `TransactionTokenType.BEGIN`, a new transaction will be stored as a valid transaction by discarding the TokenKey with the oldest date and time of execution (Least Recently Used (LRU)). • When the access is made by using the discarded transaction token, a transaction token error is thrown.
(3)	TokenValue	• TokenValue is an element for storing the token value of the transaction. • TokenValue is generated upon execution of the method wherein `TransactionTokenType.BEGIN` or `TransactionTokenType.IN` is declared in the “type” attribute of `@TransactionTokenCheck` annotation.

Warning: When the "value" attribute is specified only in "method" annotation and if the same value is specified in another Controller, it should be noted that it will be handled as a request for carrying out a series of screen transitions. "value" attribute should be specified by this method only when the screen transitions across Controllers are to be treated as the same transaction.

Basically, it is recommended not to use the method wherein "value" attribute is specified only in "method" annotation.

Note: The method for specifying NameSpace is classified according to creation granularity of the Controller,

- when "value" attributes of both "class" annotation and "method" annotation are specified
- when "value" attribute is specified only in "class" annotation

1. When a handler method which corresponds to multiple usecases is to be implemented in Controller, "value" attributes of both "class" annotation and "method" annotation are specified.

   For example, this pattern is used when registration, change, deletion of users is to be implemented in a single Controller.

2. When a handler method which corresponds to a single usecase is to be implemented in Controller, "value" attribute is specified only in "class" annotation.

   For example, this pattern is used when a Controller is implemented for every registration, modification, deletion of users.

Lifecycle of transaction token

The lifecycle (Generate, Update, Discard) control of transaction token is performed in the following scenarios.

Sr. No.	Lifecycle Control	Description
(1)	Token Generation	A new token is generated and transaction is started when the processing of the method wherein `TransactionTokenType.BEGIN` is specified in "type" attribute of `@TransactionTokenCheck` annotation, is terminated.

次のページに続く

Table. 4.34 – 前のページからの続き

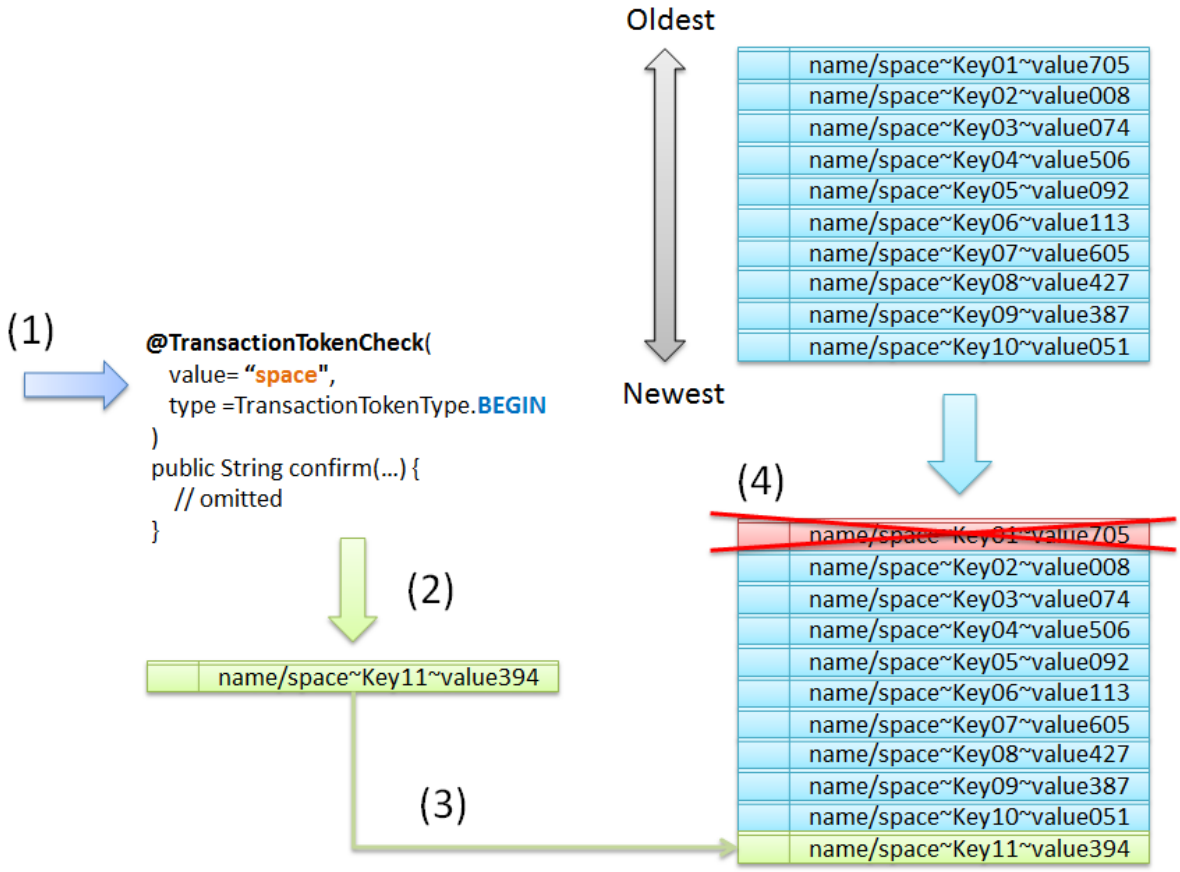
Sr. No.	Lifecycle Control	Description
(2)	Token Update	The token (TokenValue) is updated and transaction is continued when the processing of the method wherein `TransactionTokenType.IN` is specified in “type” attribute of `@TransactionTokenCheck` annotation, is terminated.
(3)	Token Discard	The tokens are discarded in any of the following scenarios and the transaction is terminated. [1] When the method wherein ‘‘TransactionTokenType.BEGIN‘‘ is specified in “type” attribute of `@TransactionTokenCheck` annotation, is called, the transaction token specified in the request parameter is discarded and unnecessary transaction is terminated. [2] If a new transaction starts when number of transaction tokens (TokenKey) that can be stored in NameSpace has reached the maximum limit, the transaction token with the oldest date and time of execution is discarded and the transaction is forcibly terminated. [3] When exceptions such as system error occur, the transaction token specified in the request parameter is discarded and the transaction is terminated.
(4)	Token inherited	Token on the server is inherited within the termination of the process for the method which specifies `TransactionTokenType.CHECK` in type attribute of `@TransactionTokenCheck` annotation and the transaction is continued.

Note: A maximum limit is provided for the number of transaction tokens (TokenKey) which can be stored in a NameSpace. When the maximum limit is reached while generating a new transaction token, a new transaction is managed as a valid transaction, by discarding the transaction token which has the TokenKey with the oldest date and time of execution (Least Recently Used (LRU)).

The maximum limit of transaction tokens that can be stored for each NameSpace is 10 by default. To change the maximum limit, refer to *How to change the maximum limit of transaction tokens* .

The behavior when the maximum limit is reached while generating a new transaction token is explained below.
The pre-requisites are as given below.

- A default value (10) is specified as the maximum limit for the number of transaction tokens that can be stored in the NameSpace.
- `@TransactionTokenCheck("name")` is specified as the class annotation of Controller.
- Transaction tokens of the same NameSpace have reached the maximum limit.



Sr. No.	Description
(1)	A request to start a new transaction is received when the transaction tokens of the same NameSpace have reached the maximum limit.
(2)	A new transaction token is generated.
(3)	The generated transaction token is added to the location where tokens are stored. ** At this point, the transaction tokens that exceed the maximum limit are present in the NameSpace.**

次のページに続く

Table. 4.35 – 前のページからの続き

Sr. No.	Description
(4)	The transaction tokens exceeding the maximum limit are deleted from the NameSpace. **The transaction tokens are deleted in a sequence starting with the transaction token with the oldest date and time of execution.**

Settings for using a transaction token check

The settings for using the transaction token check provided by the common library are shown below.

- spring-mvc.xml

```
<mvc:interceptors>
    <mvc:interceptor> <!-- (1) -->
        <mvc:mapping path="/**" /> <!-- (2) -->
        <mvc:exclude-mapping path="/resources/**" /> <!-- (2) -->
        <mvc:exclude-mapping path="/**/*.html" /> <!-- (2) -->
        <!-- (3) -->
        <bean
            class="org.terasoluna.gfw.web.token.transaction.
↪TransactionTokenInterceptor" />
    </mvc:interceptor>
</mvc:interceptors>

<bean id="requestDataValueProcessor"
    class="org.terasoluna.gfw.web.mvc.support.
↪CompositeRequestDataValueProcessor">
    <constructor-arg>
        <util:list>
            <!-- (4) -->
            <bean class="org.terasoluna.gfw.web.token.transaction.
↪TransactionTokenRequestDataValueProcessor" />
            <!-- omitted -->
        </util:list>
    </constructor-arg>
</bean>
```

Sr. No.	Description
(1)	Set `HandlerInterceptor` to generate and check transaction tokens.
(2)	Specify a request path wherein `HandlerInterceptor` is to be applied. In the above example, it is applicable to all the requests except the requests under /resources and the requests to HTML.
(3)	Specify a class (`TransactionTokenInterceptor`) to generate and check transaction tokens using `@TransactionTokenCheck` annotation.
(4)	Set a class (`TransactionTokenRequestDataValueProcessor`) for automatic embedding of the transaction token to the Hidden area using `<fomr:form>` tag of Spring MVC.

Settings for handling transaction token errors

When a transaction token error occurs,
‘‘org.terasoluna.gfw.web.token.transaction.InvalidTransactionTokenException‘‘ is generated.

Therefore, in order to handle transaction token errors, it is necessary to add the handling definition of `InvalidTransactionTokenException` to the following settings.

- `ExceptionCodeResolver` defined in `applicationContext.xml`
- `SystemExceptionResolver` defined in `spring-mvc.xml`

For adding the settings, refer to the following:

- *Common Settings*
- *Application Layer Settings*

How to use transaction token check in Controller

In order to perform transaction token check, it is necessary to define the method to start the transaction and the method to carry out the checks in Controller.

The explanation below is about the implementation of handler method required in a single usecase using a controller.

- Controller

```
@Controller
@RequestMapping("transactionTokenCheckExample")
@TransactionTokenCheck("transactionTokenCheckExample") // (1)
public class TransactionTokenCheckExampleController {

    @RequestMapping(params = "first", method = RequestMethod.GET)
    public String first() {
        return "transactionTokenCheckExample/firstView";
    }

    @RequestMapping(params = "second", method = RequestMethod.POST)
    @TransactionTokenCheck(type = TransactionTokenType.BEGIN) // (2)
    public String second() {
        return "transactionTokenCheckExample/secondView";
    }

    @RequestMapping(params = "third", method = RequestMethod.POST)
    @TransactionTokenCheck // (3)
    public String third() {
        return "transactionTokenCheckExample/thirdView";
    }

    @RequestMapping(params = "fourth", method = RequestMethod.POST)
    @TransactionTokenCheck // (3)
    public String fourth() {
        return "transactionTokenCheckExample/fourthView";
    }

    @RequestMapping(params = "fifth", method = RequestMethod.POST)
    @TransactionTokenCheck // (3)
    public String fifth() {
        return "redirect:/transactionTokenCheckExample?complete";
    }

    @RequestMapping(params = "complete", method = RequestMethod.GET)
    public String complete() { // (4)
        return "transactionTokenCheckExample/fifthView";
    }
```

```
}
```

Sr. No.	Description
(1)	NameSpace is specified in "value" attribute of "class" annotation. In the above example, the value same as the "value" attribute of `@RequestMapping` which is the recommended pattern of this guideline is specified.
(2)	The transaction is started and a new transaction token is issued. Here, since the transaction tokens are managed at Controller level, "value" attribute of "method" annotation is not specified.
(3)	The transaction token is checked and transaction token value is updated. If the type attribute is omitted, the behavior remains the same as when `@TransactionTokenCheck(type = TransactionTokenType.IN)` is specified.
(4)	Since it is not necessary to perform transaction token check in the request for displaying a screen which notifies the completion of the usecase, `@TransactionTokenCheck` annotation is not specified.

Note:

- When BEGIN is specified in the "type" attribute of `@TransactionTokenCheck` annotation, transaction token is not checked since a new TokenKey is generated.
- When IN is specified in the "type" attribute of `@TransactionTokenCheck` annotation, it is checked whether the token value specified in the request and the token value stored on the server are the same.

How to use transaction token check in View (JSP)

When transaction token is to be checked, View (JSP) should be implemented in such a way that the issued transaction token is submitted as a request parameter.

A method wherein a transaction is automatically embedded in hidden elements by using `<form:form>` tag is recommended as a method to submit it as a request parameter after carrying out *Settings for using a transaction*

token check.

- `firstView.jsp`

```
<h1>First</h1>
<form:form method="post" action="transactionTokenCheckExample">
  <input type="submit" name="second" value="second" />
</form:form>
```

- `secondView.jsp`

```
<h1>Second</h1>
<form:form method="post" action="transactionTokenCheckExample"><!-- (1) -->
  <input type="submit" name="third" value="third" />
</form:form>
```

- `thirdView.jsp`

```
<h1>Third</h1>
<form:form method="post" action="transactionTokenCheckExample"><!-- (1) -->
  <input type="submit" name="fourth" value="fourth" />
</form:form>
```

- `fourthView.jsp`

  When `<form:form>`tag is used

```
<h1>Fourth</h1>
<form:form method="post" action="transactionTokenCheckExample"><!-- (1) -->
  <input type="submit" name="fifth" value="fifth" />
</form:form>
```

  When ‘‘<form>‘‘ tag of HTML is used

```
<h1>Fourth</h1>
<form method="post" action="transactionTokenCheckExample">
  <t:transaction /><!-- (2) -->
  <!-- (3) -->
  <input type="hidden" name="${f:h(_csrf.parameterName)}"
                       value="${f:h(_csrf.token)}"/>
  <input type="submit" name="fifth" value="fifth" />
</form>
```

- `fifthView.jsp`

```
<h1>Fifth</h1>
<form:form method="get" action="transactionTokenCheckExample">
```

```
  <input type="submit" name="first" value="first" />
</form:form>
```

Sr. No.	Description
(1)	When <form:form> tag is used in JSP and if BEGIN or IN is specified in "type" attribute of @TransactionTokenCheck annotation, the Value of name="_TRANSACTION_TOKEN" is automatically embedded as a hidden tag.
(2)	When <form> tag of HTML is used, a hidden tag same as (1) is embedded using <t:transaction />.
(3)	When <form> tag of HTML is used, csrf token necessary for CSRF token check provided by Spring Security needs to be embedded as a hidden item. For csrf token necessary for CSRF token check, refer to *Coordination by using Spring MVC*.

Note: If <form:form> tag is used, the parameters necessary for CSRF token check are also automatically embedded. Refer to *Coordination while using HTML form* for the parameters necessary for CSRF token check.

Note: <t:transaction /> is a JSP tag library provided by the common library. For the "t:" used in (2), refer to *Creating common JSP for include*.

- Example of Output HTML

Following observations can be made upon verifying the output HTML.

- For NameSpace, the value specified in "value" attribute of the class annotation is set.

  In the above example, "transactionTokenCheckExample" (underlined in orange) is the NameSpace.

- For TokenKey, the value that was issued at the start of the transaction is circulated and set.

  In the above example, "c0123252d531d7baf730cd49fe0422ef" (underlined in blue) is the TokenKey.



- Value to be set for TokenValue varies depending on request.

  In the above example, `"3f610684e1cb546a13b79b9df30a7523"`, `"da770ed81dbca9a694b232e84247a13b"`, `"bd5a2d88ec446b27c06f6d4f486d4428"` (underlined in green) are TokenValues.

When multiple usecases are to be implemented in one Controller

The implementation example of the transaction token check while carrying out processing of multiple usecases in one controller is given below.

In the example given below, (2), (3), (4) are handled as screen transitions of different usecases.

- Controller

```
@Controller
@RequestMapping("transactionTokenChecFlowkExample")
@TransactionTokenCheck("transactionTokenChecFlowkExample") // (1)
public class TransactionTokenCheckFlowExampleController {
```

```
@RequestMapping(value = "flowOne",
                params = "first",
                method = RequestMethod.GET)
public String flowOneFirst() {
    return "transactionTokenChecFlowkExample/flowOneFirstView";
}

@RequestMapping(value = "flowOne",
                params = "second",
                method = RequestMethod.POST)
@TransactionTokenCheck(value = "flowOne",
                       type = TransactionTokenType.BEGIN) // (2)
public String flowOneSecond() {
    return "transactionTokenChecFlowkExample/flowOneSecondView";
}

@RequestMapping(value = "flowOne",
                params = "third",
                method = RequestMethod.POST)
@TransactionTokenCheck(value = "flowOne",
                       type = TransactionTokenType.IN)    // (2)
public String flowOneThird() {
    return "transactionTokenChecFlowkExample/flowOneThirdView";
}

@RequestMapping(value = "flowTwo",
               params = "first",
                method = RequestMethod.GET)
public String flowTwoFirst() {
    return "transactionTokenChecFlowkExample/flowTwoFirstView";
}

@RequestMapping(value = "flowTwo",
                params = "second",
                method = RequestMethod.POST)
@TransactionTokenCheck(value = "flowTwo",
                       type = TransactionTokenType.BEGIN) // (3)
public String flowTwoSecond() {
    return "transactionTokenChecFlowkExample/flowTwoSecondView";
}

@RequestMapping(value = "flowTwo",
                params = "third",
                method = RequestMethod.POST)
@TransactionTokenCheck(value = "flowTwo",
                       type = TransactionTokenType.IN) // (3)
public String flowTwoThird() {
    return "transactionTokenChecFlowkExample/flowTwoThirdView";
}
```

```
    @RequestMapping(value = "flowThree",
                    params = "first",
                    method = RequestMethod.GET)
    public String flowThreeFirst() {
        return "transactionTokenChecFlowkExample/flowThreeFirstView";
    }

    @RequestMapping(value = "flowThree",
                    params = "second",
                    method = RequestMethod.POST)
    @TransactionTokenCheck(value = "flowThree",
                           type = TransactionTokenType.BEGIN) // (4)
    public String flowThreeSecond() {
        return "transactionTokenChecFlowkExample/flowThreeSecondView";
    }

    @RequestMapping(value = "flowThree",
                    params = "third",
                    method = RequestMethod.POST)
    @TransactionTokenCheck(value = "flowThree",
                           type = TransactionTokenType.IN) // (4)
    public String flowThreeThird() {
        return "transactionTokenChecFlowkExample/flowThreeThirdView";
    }

}
```

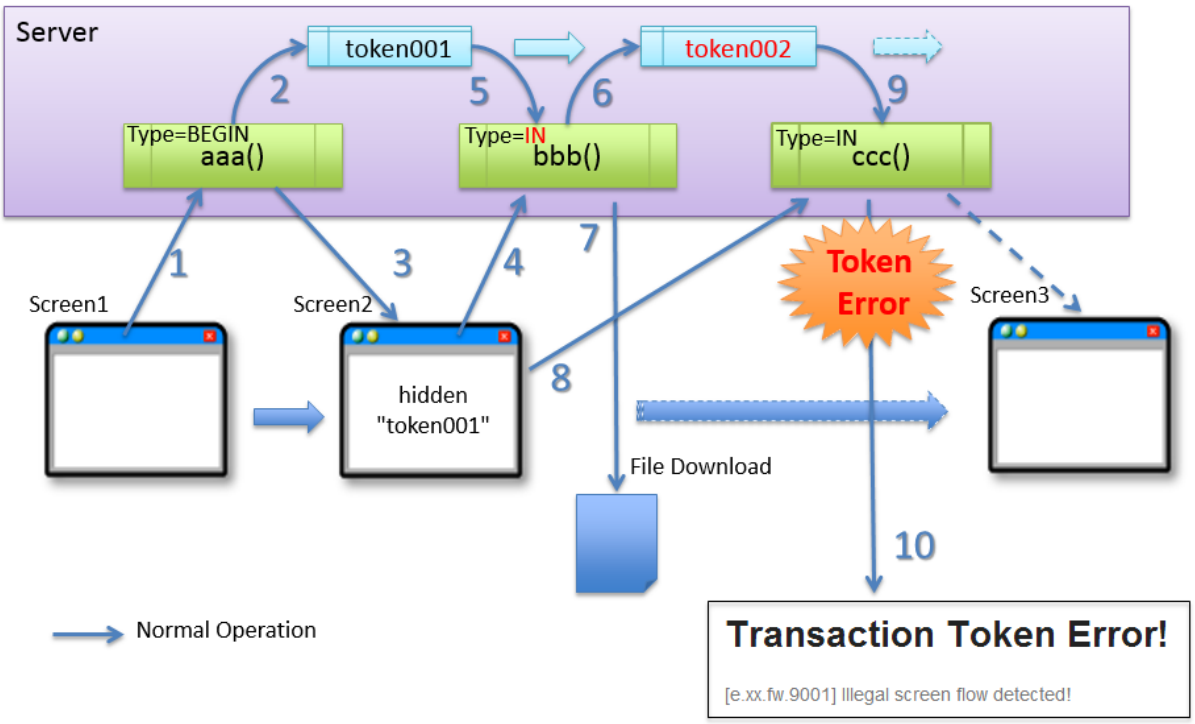
Sr. No.	Description
(1)	NameSpace is specified in “value” attribute of “class” annotation. In the above example, the value same as the “value” attribute of `@RequestMapping` which is a recommended pattern of this guideline, is specified.
(2)	The transaction token is checked for processing of usecase with name `"flowOne"`. In the above example, the value same as the “value” attribute of `@RequestMapping` which is a recommended pattern of this guideline, is specified.
(3)	The transaction token is checked for processing of usecase with name `"flowTwo"`. In the above example, the value same as the “value” attribute of `@RequestMapping` which is a recommended pattern of this guideline, is specified.
(4)	The transaction token is checked for processing of usecase with name `"flowThree"`. In the above example, the value same as the “value” attribute of `@RequestMapping` which is a recommended pattern of this guideline, is specified.

Note: Allocating a NameSpace for each usecase enables transaction token check for each usecase.

When a process which does not update screen for file downloading process is included in the use case

For the use cases which include process wherein screen is not returned to the client during file downloading process, when the TransactionTokenType is not configured appropriately, care must be taken since the transaction token error is likely to occur even during normal operations.

An example wherein the transaction token error occurs during a normal operation due to specification of inappropriate TransactionTokenType is shown below.



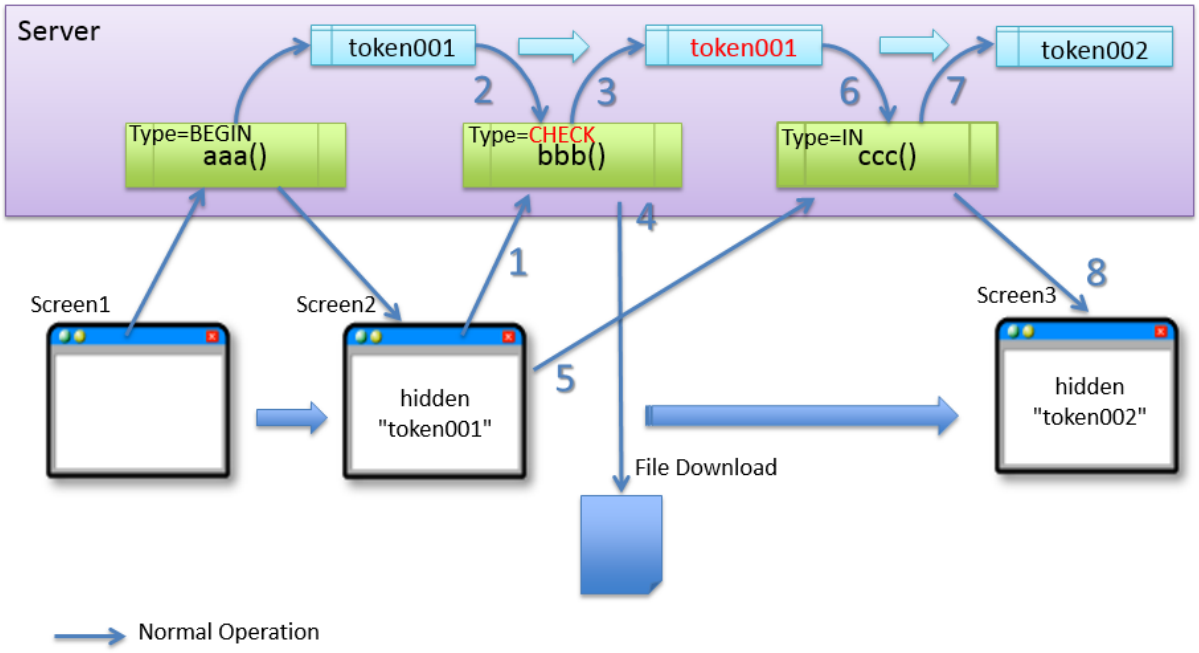
Sr. No.	Description
(1)	The client sends a request.
(2)	Server creates a token (token001) and retains it on the server.
(3)	Server delivers the token thus created (token001) to the client.
(4)	Client sends a file downloading request including the token (token001).
(5)	Server checks whether the token retained on the server (token001) and the token sent by the client (token001) are identical. **Since the value is same, the request is determined as a legitimate request.**

次のページに続く

Table. 4.36 – 前のページからの続き

Sr. No.	Description
(6)	Since `IN` is configured in `TransactionTokenType`, server generates a token (token002) to be used in the next request and updates the value which is retained on the server. At this point, the token (token001) is discarded.
(7)	Server returns the requested file. **Since the updated token is not returned to the client, token on the screen (token001) and token of the server (token002) do not match at this point.**
(8)	Client sends a request including a token (token001).
(9)	Server checks whether the token retained on the server (token001) and the token sent by the client (token002) are identical. **Since the value is not the same, the request is determined as an illegitimate request.**

As shown above, when the transaction token is applied for the transition from screen wherein file downloading process is performed (screen2) to next screen (screen3), token mismatch occurs during normal operation if `IN` is used in `TransactionTokenType` of file downloading process. For these cases, `CHECK` is used in the `TransactionTokenType`.



Sr. No.	Description
(1)	Client sends a file download request including a token (token001).
(2)	Server checks whether the token retained on the server (token001) and the token sent by client (token001) are identical. **Since the value is same, the request is determined as a legitimate request.**
(3)	Since `CHECK`is configured in `TransactionTokenType`, the token retained on the server is not updated.
(4)	Server sends the requested file.
(5)	Client sends a request including the token (token001).

次のページに続く

Table. 4.37 – 前のページからの続き

Sr. No.	Description
(6)	Server checks whether the token retained on the server (token001) and the token sent by client (token001) are identical. **Since the value is same, the request is determined as a legitimate request.**
(7)	Server generates a token to be used in the next request (token002) and updates the value retained on the server. The token (token001) is discarded at this point.
(8)	Server delivers the updated token (token002) to the client.

Warning: A method wherein `@TransactionTokenCheck` is not assigned to file downloading method of `Controller` can also be used as a method wherein the token on the server is not updated. However, the method should be selected considering that when the `@TransactionTokenCheck` is not assigned, the token is not returned to the client.

For example, if a method wherein `@TransactionTokenCheck` is not assigned is selected when either the screen or file are likely to be returned during the process results such as displaying an error message on the screen when a reexecutable error occurs during file downloading process, the token of the screen when the error message is returned to the screen is lost and the transaction token error occurs as a result during a normal operation.

Typical example of using transaction token check

See the example below wherein transaction token check is applied for the usecase which carries out a simple screen transition such as “Input Screen-> Confirmation Screen-> Completion Screen”.

- Controller

```
@Controller
@RequestMapping("user")
@TransactionTokenCheck("user") // (1)
public class UserController {

    // omitted

    @RequestMapping(value = "create", params = "form")
    public String createForm(UserCreateForm form) { // (2)
      return "user/createForm";
    }

    @RequestMapping(value = "create",
                  params = "confirm",
                  method = RequestMethod.POST)
    @TransactionTokenCheck(value = "create",
                         type = TransactionTokenType.BEGIN) // (3)
    public String createConfirm(@Validated
    UserCreateForm form, BindingResult result) {

        // omitted

        return "user/createConfirm";
    }

    @RequestMapping(value = "create", method = RequestMethod.POST)
    @TransactionTokenCheck(value = "create") // (4)
    public String create(@Validated
    UserCreateForm form, BindingResult result) {
```

```
        // omitted

        return "redirect:/user/create?complete";
    }

    @RequestMapping(value = "create", params = "complete")
    public String createComplete() { // (5)
        return "user/createComplete";
    }

    // omitted

}
```

Sr. No.	Description
(1)	A NameSpace called "`user`" is set as “class” annotation. In the above example, the value same as “value” attribute of `@RequestMapping` annotation which is a recommended pattern, is specified.
(2)	A handler method to display input screen. **It is a screen to start a usecase; however since the process only displays data and does not involve data update, it is not necessary to start a transaction.** Therefore, `@TransactionTokenCheck` annotation is not specified in the example given above.
(3)	A handler method to perform input validation and display the Confirmation screen. A transaction is started at this point since a button to perform update process is placed on the Confirmation screen. A View (JSP) is specified for the transition destination.
(4)	A handler method to perform update. **Since this method performs update, a transaction token check is performed.**
(4)	A handler method to display the Completion screen. **Since the method only displays a Completion screen, the transaction token check is not required.** Therefore, `@TransactionTokenCheck` annotation is not specified in the example given above.

Warning: It is necessary to specify the View (JSP) for the transition destination of handler method wherein `@TransactionTokenCheck` annotation is defined. If other than View (JSP) of the redirect destination is specified as transition destination, the value of TransactionToken changes in the next process always resulting in the TransactionToken error.

Exclusion control of parallel processing while using a session

When a form object is stored in a session using `@SessionAttributes` annotation, and if multiple screen transitions of the same processing are performed in parallel, screen operations may interfere with each other and the values displayed on the screen and the values stored in the session may no longer match.

Transaction token check function can be used to prevent requests from non-conforming screens as the invalid requests.

The maximum limit of transaction tokens that can be stored for each NameSpace is set to 1.

- `spring-mvc.xml`

```
<mvc:interceptor>
    <mvc:mapping path="/**" />
    <!-- omitted -->
    <bean
        class="org.terasoluna.gfw.web.token.transaction.
↪TransactionTokenInterceptor">
        <constructor-arg value="1"/> <!-- (1) -->
    </bean>
</mvc:interceptor>
```

Sr. No.	Description
(1)	The number of transaction tokens stored for each NameSpace is set to “1”.

Note: When form objects etc. are stored in session using `@SessionAttributes` annotation, the requests from the screen displaying old data can be prevented as invalid requests by setting number of transaction token stored for each NameSpace to “1”.

4.5.3 How to extend

How to change the maximum limit of transaction tokens

The maximum limit of transaction tokens that can be stored on 1 NameSpace can be changed by performing settings given below.

- `spring-mvc.xml`

```
<mvc:interceptors>
    <mvc:interceptor>
        <mvc:mapping path="/**" />
        <mvc:exclude-mapping path="/resources/**" />
        <mvc:exclude-mapping path="/**/*.html" />
        <bean
            class="org.terasoluna.gfw.web.token.transaction.
↪TransactionTokenInterceptor" />
        <constructor-arg value="5"/> <!-- (1) -->
    </mvc:interceptor>
</mvc:interceptors>
```

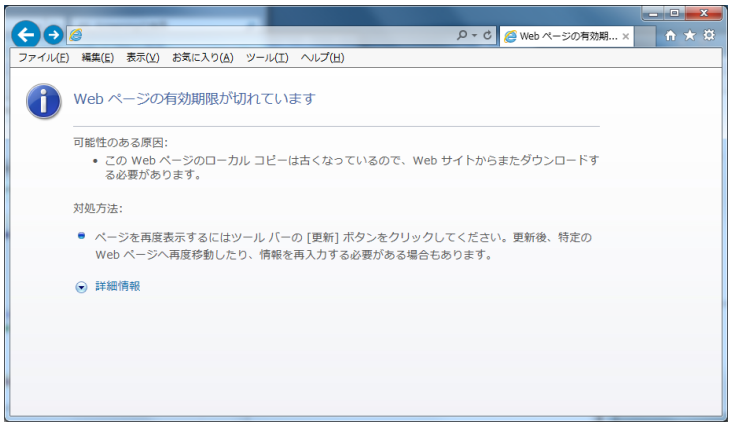
Sr. No.	Description
(1)	The maximum limit of transaction tokens that can be stored in a NameSpace is specified as a value of constructor of `TransactionTokenInterceptor`. The default value (value that is set when the default constructor is used) is 10. In the above example, the default value (10) is changed to 5.

4.5.4 Appendix

Transaction Token Check in case that the cache of browser is disabled

If the cache of web browser is disabled due to `Cache-Control` in HTTP response header, the message will display which tells the cache has been expired before transaction token error in case of the illegal operation flow in "*Transaction Token Check*"

Concretely, the following screen will display when the browser back button is clicked (8). This is an example of Internet Explorer 11.



There is no problem because the double submit itself is prevented.

In *blank projects* after 5.0.0.RELEASE, it is configured so that the cache is disabled by *Spring Security*.

If showing the transaction error screen is preferred instead of the screen above, excluding `<sec:cache-control />` is required. However, `<sec:cache-control />` should be configured from the point of view of security.

Global Tokens

If the "value" attribute of `@TransactionTokenCheck` annotation is not specified, it is handled as global transaction token.

`"globalToken"` (fixed value) is used for the NameSpace of global transaction tokens.

Note: When only a single screen transition is to be allowed as an overall application, it can be implemented by setting the maximum limit of transaction tokens that can be stored for each NameSpace to 1 and using the global token.

The settings and implementation example when only a single screen transition is to be allowed as an overall application are shown below.

Changing the maximum limit of transaction tokens that can be stored for each NameSpace

The maximum limit of transaction tokens that can be stored for each NameSpace is set to 1.

- `spring-mvc.xml`

```
<mvc:interceptor>
    <mvc:mapping path="/**" />
    <!-- omitted -->
    <bean
        class="org.terasoluna.gfw.web.token.transaction.
↪TransactionTokenInterceptor">
        <constructor-arg value="1"/> <!-- (1) -->
    </bean>
</mvc:interceptor>
```

Sr. No.	Description
(1)	The number of tokens stored for each NameSpace is set to "1".

Implementation of Controller

The value is not specified in "value" attribute of `@TransactionTokenCheck` annotation, in order to make it the NameSpace for global tokens.

- Controller

```
@Controller
@RequestMapping("globalTokenCheckExample")
public class GlobalTokenCheckExampleController { // (1)

    @RequestMapping(params = "first", method = RequestMethod.GET)
    public String first() {
        return "globalTokenCheckExample/firstView";
    }

    @RequestMapping(params = "second", method = RequestMethod.POST)
    @TransactionTokenCheck(type = TransactionTokenType.BEGIN) // (2)
    public String second() {
        return "globalTokenCheckExample/secondView";
    }

    @RequestMapping(params = "third", method = RequestMethod.POST)
    @TransactionTokenCheck // (2)
    public String third() {
        return "globalTokenCheckExample/thirdView";
    }

    @RequestMapping(params = "fourth", method = RequestMethod.POST)
    @TransactionTokenCheck // (2)
    public String fourth() {
        return "globalTokenCheckExample/fourthView";
    }

    @RequestMapping(params = "fifth", method = RequestMethod.POST)
    public String fifth() {
        return "globalTokenCheckExample/fifthView";
    }

}
```

Sr. No.	Description
(1)	`@TransactionTokenCheck` annotation is not specified as the class annotation.
(2)	The "value" attribute of `@TransactionTokenCheck` annotation to be specified as "method" annotation is not specified.

- Example of Output HTML

  JSP used is same as the one created in *How to use transaction token check in View (JSP)* .

  Other details are same except change in action from `"transactionTokenCheckExample"`to `"globalTokenCheckExample"`.



Following observations can be made upon verifying the output HTML.

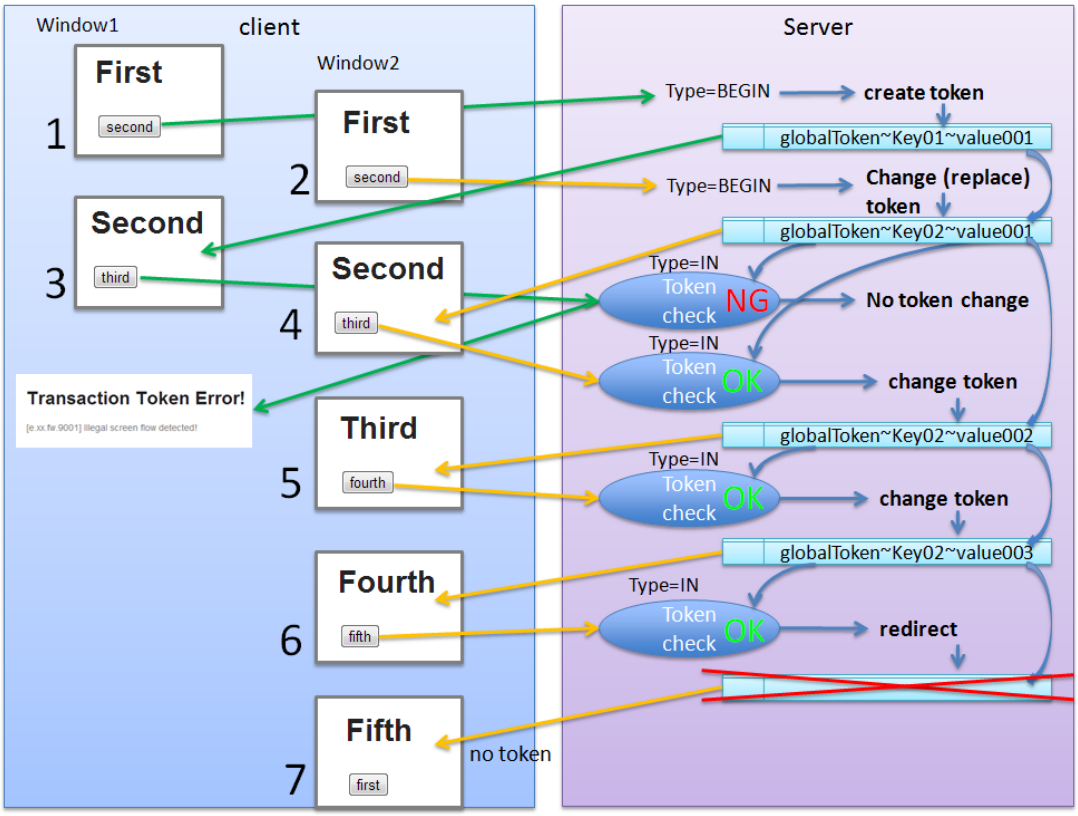
- For NameSpace, a fixed value called `"globalToken"` is set.
- For TokenKey, the value that was issued while starting the transaction is circulated and set.

In the above example, "9d937be4adc2f5dd2032292d153f1133" (underlined in blue) is the TokenKey.

- Value to be set for TokenValue varies depending on request.

  In the above example, "9204d7705ce7a17f16ca6cec24cfd88b", "69c809fefcad541dbd00bd1983af2148", "6b83f33b365f1270ee1c1b263f046719" (underlined in green) are TokenValues.

The behavior when the maximum limit of transaction tokens for each NameSpace is set to 1 and global token is used, is explained below.



Sr. No.	Description
(1)	In window 1 process, TransactionTokenType.BEGIN is performed and global token is generated.
(2)	In window 2 process, the token is updated by TransactionTokenType.BEGIN. Although internally the data is reshuffled rather than getting updated, it gives the impression that the token has been updated since one transaction token can be stored on the server.

次のページに続く

Table. 4.38 – 前のページからの続き

Sr. No.	Description
(3)	Token value is checked in TransactionTokenType.IN of window 1 process. **Transaction token generated in process 1 is submitted as request parameter; however since the specified token does not exist on the server, a transaction token error occurs.**
(4)	Token value is checked in TransactionTokenType.IN of window 2 process. The transaction token generated in process 2 is submitted as request parameter and it is checked whether the value matches with the token value stored on the server. If the value matches, the process is continued.
(5)	Similar to (4) .
(6)	Similar to (4) .
(7)	When a screen is to be displayed using redirect, hidden tag for transaction token does not exist.

Note: The transaction token existing on the server is automatically deleted when a new global token is generated.

Quick Reference

The table below illustrates an example of a business application for managing Account and Customer. It shows the required settings for transaction tokens and business limitations.

The usecases assumed in this business application are Create, Update, Delete relating to Account and Customer.

By using the table below as reference, the maximum limit of tokens and NameSpace settings should be performed as per the system requirements.

Number	Number of tokens stored for each NameSpace	NameSpace value specified in class	NameSpace value specified in method	Example of generated token	Business limitations
(1)	10 (Default)	account	Not specified	account~key~value	Number of concurrent executions of all Account usecases (create/update/delete) is restricted to 10.
(2)	10 (Default)	account	create	account/create~key~value	Number of concurrent executions of “create” operation of Account usecase is restricted to 10.

次のページに続く

Table. 4.39 – 前のページからの続き

Number	Number of tokens stored for each NameSpace	NameSpace value specified in class	NameSpace value specified in method	Example of generated token	Business limitations
(3)	10 (Default)	account	update	account/update~key~value	Number of concurrent executions of “update” operation of Account usecase is restricted to 10.
(4)	10 (Default)	account	delete	account/delete~key~value	Number of concurrent executions of “delete” operation of Account usecase is restricted to 10. (By specifying (2), (3) and (4), the number of concurrent executions of all Account usecases should be 30. Since this setting value is too high for most applications, a value smaller than default value 10 can also be specified.)

次のページに続く

Table. 4.39 – 前のページからの続き

Number	Number of tokens stored for each NameSpace	NameSpace value specified in class	NameSpace value specified in method	Example of generated token	Business limitations
(5)	10 (Default)	Unspecified	create	create~key~value	The same Namespace called “create” is created in the application and the number of concurrent executions is restricted to 10. When the Account and Customer business processes are independent, and among them when “create” is specified in the NameSpace of TransactionToken using “create” method, the total number of concurrent executions is restricted to 10.
(6)	10 (Default)	Not specified	update	update~key~value	Same as (5)
(7)	10 (Default)	Not specified	delete	delete~key~value	Same as (5)

次のページに続く

Table. 4.39 – 前のページからの続き

Number	Number of tokens stored for each NameSpace	NameSpace value specified in class	NameSpace value specified in method	Example of generated token	Business limitations
(8)	10 (Default)	Not specified	Not specified	globalToken~key~value	The total number of concurrent executions of all Account and Customer usecases is restricted to 10.
(9)	1 (Custom Setting in spring-mvc.xml)	account	Not specified	account~key~value	The number of concurrent executions of all Account usecases should be restricted to 1. Execution of only a single business process is possible for Account create/update/delete. This is valid when screen transition using only 1 screen is assumed.

次のページに続く

Table. 4.39 – 前のページからの続き

Number	Number of tokens stored for each NameSpace	NameSpace value specified in class	NameSpace value specified in method	Example of generated token	Business limitations
(10)	1 (Custom Setting in spring-mvc.xml)	account	create	account/create~key~value	The number of concurrent executions of “create” operation of Account usecases should be restricted to 1. It is not possible to simultaneously execute Account “create” by opening 2 screens.
(11)	1 (Custom Setting in spring-mvc.xml)	account	update	account/update~key~value	Same as (10)
(12)	1 (Custom Setting in spring-mvc.xml)	account	delete	account/delete~key~value	Same as (10)

次のページに続く

Table. 4.39 – 前のページからの続き

Number	Number of tokens stored for each NameSpace	NameSpace value specified in class	NameSpace value specified in method	Example of generated token	Business limitations
(13)	1 (Custom Setting in spring-mvc.xml)	Not specified	create	create~key~value	A same NameSpace called “create” is created in the application and the number of concurrent executions should be restricted to 1. When the Account and Customer business processes are independent and when “create” is specified in NameSpace of TransactionToken using “create” method, “create” for Account and Customer cannot be performed concurrently
(14)	1 (Custom Setting in spring-mvc.xml)	Not specified	update	update~key~value	Same as (13)
(15)	1 (Custom Setting in spring-mvc.xml)	Not specified	delete	delete~key~value	Same as (13)

次のページに続く

Table. 4.39 – 前のページからの続き

Number	Number of tokens stored for each NameSpace	NameSpace value specified in class	NameSpace value specified in method	Example of generated token	Business limitations
(16)	1 (Custom Setting in spring-mvc.xml)	Not specified	Not specified	globalToken~key~value	The business processes that can be concurrently executed in the application are restricted to 1. This should be used in projects where only 1 operation is performed in 1 session.

4.6 Message Management

4.6.1 Overview

A message consists of fixed text displayed on screens or reports, or dynamic text displayed depending on screen operations performed by user.

It is also recommended to define a error message in as much details as possible.

> **Warning:** In following cases, there is a risk of inability to identify error cause during the production phase or during the testing just before entering into production phase (however, such risks may not surface during the development phase).
>
> - When only one error message is defined
> - When only two types of error messages ("Important" and "Warning") are defined
>
> Thus, if messages are changed when the number of developers in the team is less, the cost to modify these messages would increase as the development progresses. It is, therefore, recommended to define the messages in advance at a detailed level.

Types of messages

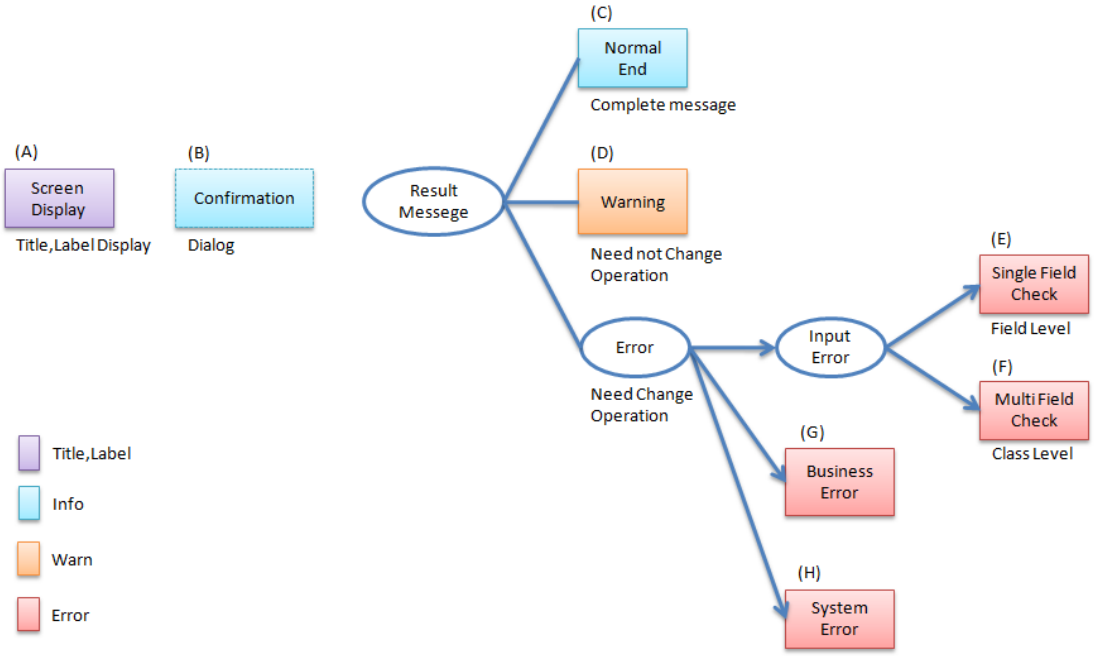
Message that display according to the result of the user's screen operation should be classified into following 3 types depending on the message contents.

When defining any message, it important to note its type.

Message type	Category	Overview
info	Information message	This message is displayed when a process is executed normally by the user.
warn	Warning message	This message is displayed to indicate waring to be focused on; however the process can be continued. (Example: Notification indicating that the password is about to expire)
error	Input error message	This message is displayed on input screen when value entered by the user is invalid.
	Business error message	This message is displayed to indicate error in the business logic
	System error message	This message is displayed when system errors (database connection failure etc.) occur and recovery is not possible by user operations.

Types of messages depending on patterns

Message output patterns are shown below.



Message patterns, message display contents and the message type are shown below.

Symbol	Pattern	Display contents	Message type	Example
(A)	Title	Screen title	-	• Employee Registration screen
	Label	Screen field name Report field name Comment Guidance	-	• User name • Password
(B)	Dialog	Confirmation message	info	• Are you sure you want to register? • Are you sure you want to delete?

次のページに続く

Table. 4.40 – 前のページからの続き

Symbol	Pattern	Display contents	Message type	Example
(C)	Result message	Successful completion	info	• Registered. • Deleted.
(D)		Warning	warn	• Password is about to expire. Please change the password. • Server is busy. Please try again later.
(E)		Single field validation error	error	• “User name” is mandatory. • Please enter “Name” within 20 characters. • Please enter the “Amount” in number.
(F)		Correlation check error	error	• “Password” and “Confirm Password” do not match.
(G)		Business error	error	• Failed to cancel the reservation as cancellation period has elapsed. • Failed to register as number of allowed registrations exceeded.
(H)		System error	error	• XXXSystem is blocked, please try again later. • Timeout has occurred. • System error.

Message ID

For effective message management, adding an ID to the message is recommended.

The advantages of adding an ID are as follows:

- To change the message without modifying the source code.
- To be able to identify the message output location easily
- To support internationalization

From maintenance perspective, it is strongly recommended that you define the message IDs by creating and standardizing the rules.

See the example below for Message ID rules for each message pattern.

Refer to these rules when message ID rules are not defined in a development project.

Title

The method of defining message ID to be used in screen title is described below.

- Format

Prefix	Delimiter	Business process name	Delimiter	Screen name
title	.	nnn*	.	nnn*

- Description

Field	Position	Contents	Remarks
Prefix	1st - 5th digit (5 digits)	“title” (fixed)	
Business process name	Variable length : Optional	Directory under prefix of viewResolver defined in spring-mvc.xml (parent directory of JSP)	
Screen name	Variable length : Optional	JSP name	“aaa” when file name is “aaa.jsp”

- Example

```
# In case of "/WEB-INF/views/admin/top.jsp"
title.admin.top=Admin Top
# In case of "/WEB-INF/views/staff/createForm.jsp"
title.staff.createForm=Staff Register Input
```

Tip: This example is valid when using Tiles. For details, refer to *Screen Layout using Tiles*. When not using Tiles, follow the *Labels* rules explained later.

Labels

The method of defining message ID to be used in screen label and fixed text of reports is described below.

- Format

Prefix	Delimiter	Project code	Delimiter	Business process name	Delimiter	Field name
label	.	xx	.	nnn*	.	nnn*

- Description

Field	Position	Contents	Remarks
Prefix	1st - 5th digit (5 digits)	“label” (Fixed)	
Project code	7th - 8th digit (2 digits)	Enter 2 alphabets of project name	
Business process name	Variable length : Optional		
Field name	Variable length : Optional	Label name, Caption	

Note: When including the field name into validation error message, define messages as follows.

- model attribute name of form + ”.” + field name

```
staffForm.staffName = Staff name
```

- filed name

```
staffName = Staff name
```

- Example

```
# Form field name on Staff Registration screen
# Project code=em (Event Management System)
label.em.staff.staffName=Staff name
# In case of a caption to be displayed on Tour Search screen
# Project code=tr (Tour Reservation System)
label.tr.tourSearch.tourSearchMessage=You can search tours with the␣
↪specified conditions.
```

Note: In case of multiple projects, define a project code to avoid duplication of message ID. Even if there is a single project, it is recommended to define a project code for future enhancements.

Result messages

Messages commonly used in business processes

To avoid duplication of messages, the messages which are common in multiple business processes are explained below.

- Format

Message type	Delimiter	Project code	Delimiter	Common message code	Delimiter	Error level	Sr. No.
x	.	xx	.	fw	.	9	999

- Description

Field	Position	Contents	Remarks
Message type	1st digit (1 digit)	info : i warn : w error : e	
Project code	3rd - 4th digit (2 digits)	Enter 2 alphabets of project name	
Common message code	6th - 7th digit (2 digits)	“fw” (fixed)	
Error level	9th digit (1 digit)	0-1 : Normal message 2-4 : Business error (semi-normal message) 5-7 : Input validation error 8 : Business error (error) 9 : System error	
Sr. No.	10th -12th digit (3 digits)	Use as per serial number (000-999)	Even if the message is deleted, serial number field should be blank and it should not be deleted.

- Example

```
# When registration is successful (Normal message)
i.ex.fw.0001=Registered successfully.
# Insufficient server resources
w.ex.fw.9002=Server busy. Please, try again.
# When system error occurs (System error)
e.ex.fw.9001=A system error has occurred.
```

Messages used individually in each business process

The messages used individually in each business process are explained below.

- Format

Message type	Delimiter	Project code	Delimiter	Business process message code	Delimiter	Error level	Sr. No.
x	.	xx	.	xx	.	9	999

- Description

Field	Position	Contents	Remarks
Message type	1st digit (1 digit)	info : i warn : w error : e	
Project code	3rd -4th digit (2 digits)	Enter 2 alphabets of project name	
Business process message code	6th -7th digit (2 digits)	2 characters defined for each business process such as Business ID	
Error level	9th digit (1 digit)	0-1 : Normal message 2-4 : Business error (semi-normal message) 5-7 : Input validation error 8 : Business error (error) 9 : System error	
Sr. No.	10th -12th digit (3 digits)	Use as per serial number (000-999)	Even if the message is deleted, serial number field should be blank and it should not be deleted.

- Example

```
# When file upload is successful.
i.ex.an.0001={0} upload completed.
# When the recommended password change interval has passed.
w.ex.an.2001=The recommended change interval has passed password. Please␣
↪change your password.
# When file size exceeds the limit.
e.ex.an.8001=Cannot upload, Because the file size must be less than {0}MB.
# When there is inconsistency in data.
e.ex.an.9001=There are inconsistencies in the data.
```

Input validation error message

For the messages to be displayed in case of input validation error, refer to *Definition of error messages*.

Note: Basic policies related to output location of input validation error are as follows:

- Single field input validation error messages should be displayed next to the target field so that it can be identified easily.
- Correlation input validation error messages should be displayed collectively on the top of the page .
- When it is difficult to display the single field validation message next to the target field, it should be displayed on the top of the page.

  In that case, field name should be included in the message.

4.6.2 How to use

Display of messages set in properties file

Settings at the time of using properties

Define implementation class of `org.springframework.context.MessageSource` which is used for performing message management.

- applicationContext.xml

```
<!-- Message -->
<bean id="messageSource"
    class="org.springframework.context.support.ResourceBundleMessageSource
↪"> <!-- (1) -->
    <property name="basenames"> <!-- (2) -->
        <list>
            <value>i18n/application-messages</value>
        </list>
    </property>
</bean>
```

Sr. No.	Description
(1)	Definition of `MessageSource`. Here, use `ResourceBundleMessageSource` .
(2)	Define the base name of message property to be used. Specify it with relative class path. In this example, read "src/main/resources/i18n/application-messages.properties".

Display of messages set in properties

- application-messages.properties

  See the example below for defining the messages in `application-messages.properties` .

```
label.aa.bb.year=Year
label.aa.bb.month=Month
label.aa.bb.day=Day
```

Note: Earlier, it was necessary to convert the characters (such as Japanese characters etc.) that cannot be expressed into "ISO-8859-1" with the help of `native2ascii` command. However, from JDK version 6 onwards, it has become possible to specify the character encoding; hence character conversion is no longer needed. By setting the character encoding to UTF-8, Japanese characters etc. can be used directly in properties file.

  - application-messages.properties

```
label.aa.bb.year= Year
label.aa.bb.month= Month
label.aa.bb.day= Day
```

  In such a case, it is necessary to specify the character encoding that can also be read in `ResourceBundleMessageSource` .

- applicationContext.xml

```
<bean id="messageSource"
    class="org.springframework.context.support.
↪ResourceBundleMessageSource">
    <property name="basenames">
        <list>
            <value>i18n/application-messages</value>
        </list>
    </property>
    <property name="defaultEncoding" value="UTF-8" />
</bean>
```

ISO-8859-1 is used by default; hence when describing the Japanese characters directly in properties file, make sure that the character encoding is set as value of `defaultEncoding` property.

- JSP

Messages set above can be displayed using `<spring:message>` tag in JSP. For using it, settings mentioned in *Creating common JSP for include* must be done.

```
<spring:message code="label.aa.bb.year" />
<spring:message code="label.aa.bb.month" />
<spring:message code="label.aa.bb.day" />
```

When used with form label, it can be used as follows:

```
<form:form modelAttribute="sampleForm">
    <form:label path="year">
        <spring:message code="label.aa.bb.year" />
    </form:label>: <form:input path="year" />
    <br>
    <form:label path="month">
        <spring:message code="label.aa.bb.month" />
    </form:label>: <form:input path="month" />
    <br>
    <form:label path="day">
        <spring:message code="label.aa.bb.day" />
    </form:label>: <form:input path="day" />
</form:form>
```

It is displayed in browser as follows:

Tip: When supporting internationalization,

```
src/main/resources/i18n
                    ├ application-messages.properties (English message)
```




```
                    ├ application-messages_fr.properties (French message)
                    ├ ...
                    └ application-messages_ja.properties (Japanese␣
↪message)
```

properties file should be created for each language as shown above. For details, refer to *International-ization* .

Display of result messages

`org.terasoluna.gfw.common.message.ResultMessages` and `org.terasoluna.gfw.common.message.Res` are provided in common library, as classes storing the result messages which indicate success or failure of process at server side.

Class name	Description
`ResultMessages`	Class having list of result messages and message type. List of Result messages is expressed in terms of `List<ResultMessage>` and message type is expressed in terms of `org.terasoluna.gfw.common.message.ResultMessageType` interface.
`ResultMessage`	Class having result message ID or message text.

`<t:messagesPanel>` tag is also provided as JSP tag library for displaying this result message in JSP.

Using basic result messages

The way of creating `ResultMessages` in Controller, passing them to screen and displaying the result messages using `<t:messagesPanel>` tag in JSP, is displayed below.

- Controller class

The methods of creating `ResultMessages` object and passing the messages to screen are given below. An example of *Messages used individually in each business process* should be defined in application-messages.properties.

```
package com.example.sample.app.message;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.terasoluna.gfw.common.message.ResultMessages;

@Controller
@RequestMapping("message")
public class MessageController {

  @RequestMapping(method = RequestMethod.GET)
  public String hello(Model model) {
    ResultMessages messages = ResultMessages.error().add("e.ex.an.9001"); //
↪/ (1)
    model.addAttribute(messages); // (2)
    return "message/index";
  }
}
```

Sr. No.	Description
(1)	Create `ResultMessages` wherein message type is “error” and set result messages wherein message ID is “e.ex.an.9001”. This process is same as follows: `ResultMessages.error().add(ResultMessage.fromCode("e.ex.an.9001"));` Since it is possible to skip the creation of `ResultMessage` object if message ID is specified, it is recommended to skip the same.
(2)	Add `ResultMessages` to Model. It is ok even if the attribute is not specified. (Attribute name is “resultMessages”)

- JSP

Write WEB-INF/views/message/index.jsp as follows:

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="utf-8">
<title>Result Message Example</title>
</head>
<body>
    <h1>Result Message</h1>
    <t:messagesPanel /><!-- (1) -->
</body>
</html>
```

Sr. No.	Description
(1)	<t:messagesPanel> tag is used with default settings. By default, “resultMessages” object is displayed. Therefore, attribute name need not be specified when ResultMessages is set in Model from Controller with default settings.

It is displayed in browser as follows:



HTML output by <t:messagesPanel> is shown below. (The format makes the explanation easier).

```
<div class="alert alert-error"><!-- (1) -->
  <ul><!-- (2) -->
    <li>There are inconsistencies in the data.</li><!-- (3) -->
  </ul>
</div>
```

Sr. No.	Description
(1)	"alert-error"class is assigned in accordance with the message type. "error error-[Message type]" is assigned to `<div>` tag class by default.
(2)	Result message list is output using `<ul>` tag.
(3)	The message corresponding to message ID is resolved from `MessageSource`.

`<t:messagesPanel>` outputs only HTML with class; hence it is necessary to customize the look and feel using CSS as per the output class (explained later).

Note: Message text can be hard-coded such as `ResultMessages.error().add(ResultMessage.fromText("There are inconsistencies in the data."));`; however, to enhance maintainability, it is recommended to create `ResultMessage` object using message key, and to fetch the message text from properties file.

For inserting a value in message placeholder, set second or subsequent arguments of `add` method as follows:

```
ResultMessages messages = ResultMessages.error().add("e.ex.an.8001", 1024);
model.addAttribute(messages);
```

In such a case, the HTML shown below is output using `<t:messagesPanel />` tag.

```
<div class="alert alert-error">
  <ul>
    <li>Cannot upload, Because the file size must be less than 1,024MB.</li>
  </ul>
</div>
```

Warning: Points to be noted when inserting values in placeholder using terasoluna-gfw-web 1.0.0.RELEASE

> When using terasoluna-gfw-web 1.0.0.RELEASE, **if the user entered value is inserted in the placeholder, there is a risk of XSS vulnerability.**If the user entered value is likely to include XSS vulnerable characters, then the value should not be inserted in the placeholder.
>
> When using terasoluna-gfw-web 1.0.1.RELEASE or higher version, XSS vulnerability does not occur even after inserting the user entered value in the placeholder.

Note: `ResourceBundleMessageSource` uses `java.text.MessageFormat` at the time of creating a message; hence `1024` is displayed as `1,024` with comma. When comma is not required, perform settings in properties file as shown below.

```
e.ex.an.8001=Cannot upload, Because the file size must be less than {0,
↪number,#}MB.
```

For details, refer to Javadoc.

It is also possible to set multiple result messages as shown below.

```
ResultMessages messages = ResultMessages.error()
    .add("e.ex.an.9001")
    .add("e.ex.an.8001", 1024);
model.addAttribute(messages);
```

In such a case, HTML is output as follows (no need to change JSP).

```
<div class="alert alert-error">
  <ul>
    <li>There are inconsistencies in the data.</li>
    <li>Cannot upload, Because the file size must be less than 1,024MB.</li>
  </ul>
</div>
```

In order to display info message, it is desirable to create `ResultMessages` object using `ResultMessages.info()` method as shown below.

```
ResultMessages messages = ResultMessages.info().add("i.ex.an.0001", "XXXX");
model.addAttribute(messages);
```

HTML shown below is output.

```
<div class="alert alert-info"><!-- (1) -->
  <ul>
    <li>XXXX upload completed.</li>
  </ul>
</div>
```

Sr. No.	Description
(1)	The output class name has changed to “alert alert-**info**” in accordance with the message type.

Fundamentally the following message types are created.

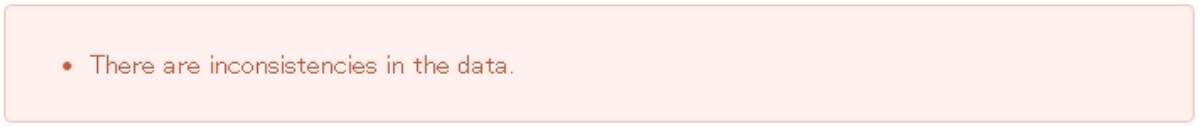
Message type	Creation of `ResultMessages` object	Default class name	Remarks
success	`ResultMessages.success()`	alert alert-success	-
info	`ResultMessages.info()`	alert alert-info	-
warn	`ResultMessages.warn()`	alert alert-warn	This type is deprecated because added “warning” as new message type from terasoluna-gfw-common 5.0.0.RELEASE. **This message type might be removed in the future.**
warning	`ResultMessages.warning()`	alert alert-warning	Added this message type to support Alerts component of Bootstrap(CSS Framework) from the terasoluna-gfw-common 5.0.0.RELEASE.
error	`ResultMessages.error()`	alert alert-error	-
danger	`ResultMessages.danger()`	alert alert-danger	-

CSS should be defined according to the message type. Example of applying CSS is given below.

```
.alert {
  margin-bottom: 15px;
  padding: 10px;
  border: 1px solid;
  border-radius: 4px;
  text-shadow: 0 1px 0 #ffffff;
}
```

```
.alert-info {
  background: #ebf7fd;
  color: #2d7091;
  border-color: rgba(45, 112, 145, 0.3);
}
.alert-warning {
  background: #fffceb;
  color: #e28327;
  border-color: rgba(226, 131, 39, 0.3);
}
.alert-error {
  background: #fff1f0;
  color: #d85030;
  border-color: rgba(216, 80, 48, 0.3);
}
```

- Example wherein `ResultMessages.error().add("e.ex.an.9001")` is output using `<t:messagesPanel />`



- Example wherein `ResultMessages.warning().add("w.ex.an.2001")` is output using `<t:messagesPanel />`



- Example wherein `ResultMessages.info().add("i.ex.an.0001","XXXX")` is output using `<t:messagesPanel />`



Note: "success" and "danger" are provided to have diversity in style. In this guideline, success is synonymous with info and error is synonymous with danger.

Tip: Alerts component of Bootstrap 3.0.0 which is a CSS framework can be used with default settings of `<t:messagePanel />`.

Warning: In this example, message keys are hardcoded. However, in order to improve maintainability, it is recommended to define message keys in constant class.

Refer to *Auto-generation tool of message key constant class*.

Specifying attribute name of result messages

Attribute name can be omitted when adding `ResultMessages` to Model.

However, `ResultMessages` cannot represent more than one message type.

In order to **simultaneously** display the `ResultMessages` of different message types on 1 screen, it is necessary to specify the attribute name explicitly and set it in Model.

- Controller (Add to MessageController)

```
@RequestMapping(value = "showMessages", method = RequestMethod.GET)
public String showMessages(Model model) {

    model.addAttribute("messages1",
                ResultMessages.warning().add("w.ex.an.2001")); // (1)
    model.addAttribute("messages2",
                ResultMessages.error().add("e.ex.an.9001")); // (2)

    return "message/showMessages";
}
```

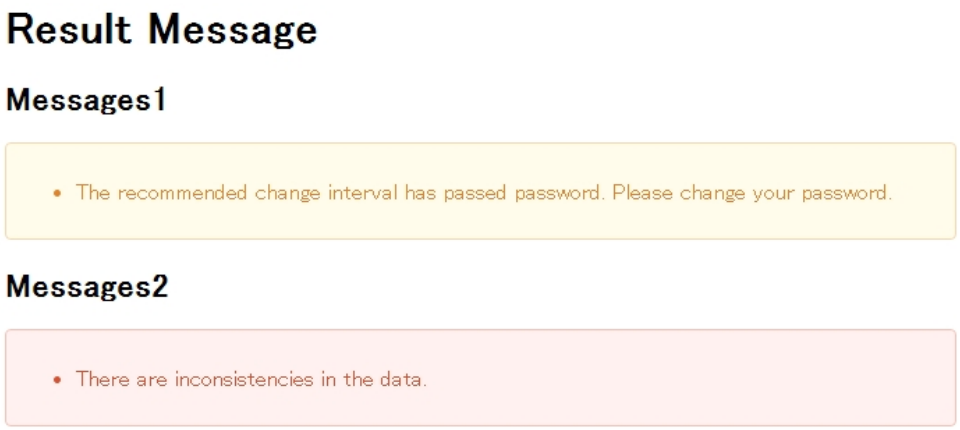
Sr. No.	Description
(1)	Add `ResultMessages` of "warning" message type to Model with attribute name "messages1".
(2)	Add `ResultMessages` of "info" message type to Model with attribute name "messages2".

- JSP (WEB-INF/views/message/showMessages.jsp)

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="utf-8">
<title>Result Message Example</title>
<style type="text/css">
.alert {
    margin-bottom: 15px;
    padding: 10px;
    border: 1px solid;
    border-radius: 4px;
    text-shadow: 0 1px 0 #ffffff;
}

.alert-info {
    background: #ebf7fd;
    color: #2d7091;
    border-color: rgba(45, 112, 145, 0.3);
}

.alert-warning {
    background: #fffceb;
    color: #e28327;
    border-color: rgba(226, 131, 39, 0.3);
}

.alert-error {
    background: #fff1f0;
    color: #d85030;
    border-color: rgba(216, 80, 48, 0.3);
}
</style>
</head>
<body>
    <h1>Result Message</h1>
    <h2>Messages1</h2>
    <t:messagesPanel messagesAttributeName="messages1" /><!-- (1) -->
    <h2>Messages2</h2>
    <t:messagesPanel messagesAttributeName="messages2" /><!-- (2) -->
</body>
</html>
```

Sr. No.	Description
(1)	Display `ResultMessages` having attribute name “messages1”.
(2)	Display `ResultMessages` having attribute name “messages2”.

It is displayed in browser as follows:



Displaying business exception messages

`org.terasoluna.gfw.common.exception.BusinessException` and `org.terasoluna.gfw.common.exception.ResourceNotFoundException` stores `ResultMessages` internally.

When displaying the business exception message, `BusinessException` wherein `ResultMessages` is set should be thrown in Service class.

Catch `BusinessException` in Controller class and add the result message fetched from the caught exception to Model.

- Service class

```
@Service
@Transactional
public class UserServiceImpl implements UserService {
    // omitted

    public void create(...) {
```

```
        // omitted...

        if (...) {
            // illegal state!
            ResultMessages messages = ResultMessages.error()
                                                    .add("e.ex.an.9001"); /
↪/ (1)
            throw new BusinessException(messages);
        }
    }

}
```

Sr. No.	Description
(1)	Create error message using `ResultMessages` and set in `BusinessException`.

- Controller class

```
@RequestMapping(value = "create", method = RequestMethod.POST)
public String create(@Validated UserForm form, BindingResult result, Model␣
↪model) {
    // omitted

    try {
        userService.create(user);
    } catch (BusinessException e) {
        ResultMessages messages = e.getResultMessages(); // (1)
        model.addAttribute(messages);

        return "user/createForm";
    }

    // omitted
}
```

Sr. No.	Description
(1)	Fetch `ResultMessages` held by `BusinessException` and add to Model.

Normally, this method should be used to display error message instead of creating `ResultMessages` object in Controller.

4.6.3 How to extend

Creating independent message types

The method of creating independent message type is given below.

Normally, the available message types are sufficient. However, new message type may need to be added depending upon the CSS library. See the example below for adding the message type “notice”.

First, create independent message type class wherein `org.terasoluna.gfw.common.message.ResultMessageType` interface is implemented as follows:

```
import org.terasoluna.gfw.common.message.ResultMessageType;

public enum ResultMessageTypes implements ResultMessageType { // (1)
    NOTICE("notice");

    private ResultMessageTypes(String type) {
        this.type = type;
    }

    private final String type;

    @Override
    public String getType() { // (2)
        return this.type;
    }

    @Override
    public String toString() {
        return this.type;
    }
}
```

Sr. No.	Description
(1)	Define Enum wherein `ResultMessageType` interface is implemented. A new message type can be created using constant class; however it is recommended to create it using Enum.
(2)	Return value of `getType` corresponds to class name of CSS which is output.

Create `ResultMessages` using this message type as mentioned below.

```
ResultMessages messages = new ResultMessages(ResultMessageTypes.NOTICE) // (1)
        .add("w.ex.an.2001");
model.addAttribute(messages);
```

Sr. No.	Description
(1)	Specify `ResultMessageType` in constructor of `ResultMessages`.

In such a case, HTML shown below is output in `<t:messagesPanel />`.

```
<div class="alert alert-notice">
  <ul>
    <li>The recommended change interval has passed password. Please change your␣
↪password.</li>
  </ul>
</div>
```

Tip: For extension method, refer to `org.terasoluna.gfw.common.message.StandardResultMessageType`

4.6.4 Appendix

Changing attribute of <t:messagesPanel> tag

`<t:messagesPanel>` tag contains various attributes for changing the display format.

Table.4.41 `<t:messagesPanel>` Tag attribute list

Option	Contents	Default setting value
panelElement	Result message display panel elements	div
panelClassName	CSS class name of result message display panel.	alert
panelTypeClassPrefix	Prefix of CSS class name	alert-
messagesType	Message type. When this attribute is set, the set message type is given preference over the message type of `ResultMessages` object.	
outerElement	Outer tag of HTML configuring result messages list	ul
innerElement	Inner tag of HTML configuring result messages list	li
disableHtmlEscape	Flag for disabling HTML escaping. By setting the flag to `true`, HTML escaping is no longer performed for the message to be output. This attribute is used to create different message styles by inserting HTML into the message to be output. **When the flag is set to true, XSS vulnerable characters should not be included in the message.** This attribute can be used with terasoluna-gfw-web 1.0.1.RELEASE or higher version.	`false`

For example, following CSS is provided in CSS framework "BlueTrip".

```
.error,.notice,.success {
    padding: .8em;
    margin-bottom: 1.6em;
    border: 2px solid #ddd;
}

.error {
    background: #FBE3E4;
    color: #8a1f11;
    border-color: #FBC2C4;
}

.notice {
    background: #FFF6BF;
    color: #514721;
    border-color: #FFD324;
}
```

```
.success {
    background: #E6EFC2;
    color: #264409;
    border-color: #C6D880;
}
```

To use this CSS, the message `<div class="error">...</div>` should be output.

In this case, `<t:messagesPanel>` tag can be used as follows (no need to modify the Controller):

```
<t:messagesPanel panelClassName="" panelTypeClassPrefix="" />
```

HTML shown below is output.

```
<div class="error">
  <ul>
    <li>There are inconsistencies in the data.</li>
  </ul>
</div>
```

It is displayed in browser as follows:

- There are inconsistencies in the data.

When you do not want to use `<ul>` tag to display the message list, it can be customized using `outerElement` attribute and `innerElement` attribute.

When the attributes are set as follows:

```
<t:messagesPanel outerElement="" innerElement="span" />
```

HTML shown below is output.

```
<div class="alert alert-error">
    <span>There are inconsistencies in the data.</span>
    <span>Cannot upload, Because the file size must be less than 1,024MB.</span>
</div>
```

Set the CSS as follows:

```
.alert > span {
    display: block; /* (1) */
}
```

Sr. No.	Description
(1)	Set tag which is a child element of “alert” class to Block-level element.

It is displayed in browser as follows:

There are inconsistencies in the data.
Cannot upload, Because the file size must be less than 1,024MB.

When disableHtmlEscape attribute is set to `true`, the output will be as follows:

In the example below, font of a part of the message has been set to 16px Red.

- jsp

```
<spring:message var="informationMessage" code="i.ex.od.0001" />
<t:messagesPanel messagesAttributeName="informationMessage"
    messagesType="alert alert-info"
    disableHtmlEscape="true" />
```

- properties

```
i.ex.od.0001 = Please confirm order content. <font style="color: red; font-
↪size: 16px;">If this orders submitted, cannot cancel.</font>
```

- Output image

- Please confirm order content. If this orders submitted, cannot cancel.

When disableHtmlEscape attribute is `false`(default), the output will be as follows after HTML escaping.

- Please confirm order content. If this orders submitted, cannot cancel.

Display of result message wherein ResultMessages is not used

Apart from `ResultMessages` object, `<t:messagesPanel>` tag can also output the following objects.

- java.lang.String
- java.lang.Exception
- java.util.List

Normally <t:messagesPanel> tag is used to output the ResultMessages object; however
it can also be used to display strings (error messages) set in the request scope by the framework.

For example, at the time of authentication error, Spring Security sets the exception class with attribute name "SPRING_SECURITY_LAST_EXCEPTION"
in the request scope.

Perform the following settings if you want to output this exception message in <t:messagesPanel> tag similar to the result messages.

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="utf-8">
<title>Login</title>
<style type="text/css">
/* (1) */
.alert {
    margin-bottom: 15px;
    padding: 10px;
    border: 1px solid;
    border-radius: 4px;
    text-shadow: 0 1px 0 #ffffff;
}

.alert-error {
    background: #fff1f0;
    color: #d85030;
    border-color: rgba(216, 80, 48, 0.3);
}
</style>
</head>
<body>
    <c:if test="${param.containsKey('error')}">
        <t:messagesPanel messagesType="error"
            messagesAttributeName="SPRING_SECURITY_LAST_EXCEPTION" /><!-- (2) -->
    </c:if>
```

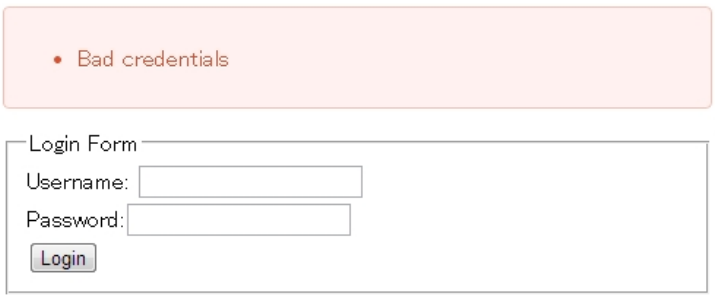
```
    <form:form
        action="${pageContext.request.contextPath}/authentication"
        method="post">
        <fieldset>
            <legend>Login Form</legend>
            <div>
                <label for="username">Username: </label><input
                    type="text" id="username" name="username">
            </div>
            <div>
                <label for="username">Password:</label><input
                    type="password" id="password" name="password">
            </div>
            <div>
                <input type="submit" value="Login" />
            </div>
        </fieldset>
    </form:form>
</body>
</html>
```

Sr. No.	Description
(1)	Re-define the CSS. It is strongly recommended to mention it in CSS file.
(2)	In `messagesAttributeName` attribute, specify the attribute name wherein `Exception` object is stored. Unlike the `ResultMessages` object, it does not contain the information of message type; hence it is necessary to explicitly specify the message type in `messagesType` attribute.

The HTML output in case of an authentication error will be,

```
<div class="alert alert-error"><ul><li>Bad credentials</li></ul></div>
```

and it will be displayed in the browser as follows:



Tip: For details on JSP for login, refer to *Authentication*.

Auto-generation tool of message key constant class

In all earlier examples, message keys were hard-coded strings; however it is recommended that you define the message keys in constant class.

This section introduces the program that auto-generates message key constant class from properties file and the corresponding usage method. You can customize and use them based on the requirements.

1. Creation of message key constant class

   First, create an empty message key constant class. Here, it is `com.example.common.message.MessageKeys`.

```
package com.example.common.message;

public class MessageKeys {

}
```

2. Creation of auto-generation class

   Next, create `MessageKeysGen` class in the same package as `MessageKeys` class and write the logic as follows:

```
package com.example.common.message;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.util.regex.Pattern;

import org.apache.commons.io.FileUtils;
import org.apache.commons.io.IOUtils;

public class MessageKeysGen {
    public static void main(String[] args) throws IOException {
        // message properties file
        InputStream inputStream = new FileInputStream("src/main/resources/
↪i18n/application-messages.properties");
        BufferedReader br = new BufferedReader(new␣
↪InputStreamReader(inputStream));
```

```
        Class<?> targetClazz = MessageKeys.class;
        File output = new File("src/main/java/"
                + targetClazz.getName().replaceAll(Pattern.quote("."), "/")
                + ".java");
        System.out.println("write " + output.getAbsolutePath());
        PrintWriter pw = new PrintWriter(FileUtils.
↪openOutputStream(output));

        try {
            pw.println("package " + targetClazz.getPackage().getName() + ";
↪");
            pw.println("/**");
            pw.println(" * Message Id");
            pw.println(" */");
            pw.println("public class " + targetClazz.getSimpleName() + " {
↪");

            String line;
            while ((line = br.readLine()) != null) {
                String[] vals = line.split("=", 2);
                if (vals.length > 1) {
                    String key = vals[0].trim();
                    String value = vals[1].trim();
                    pw.println("    /** " + key + "=" + value + " */");
                    pw.println("    public static final String "
                            + key.toUpperCase().replaceAll(Pattern.quote(".
↪"),
                                    "_").replaceAll(Pattern.quote("-"), "_
↪")
                            + " = \"" + key + "\";");
                }
            }
            pw.println("}");
            pw.flush();
        } finally {
            IOUtils.closeQuietly(br);
            IOUtils.closeQuietly(pw);
        }
    }
}
```

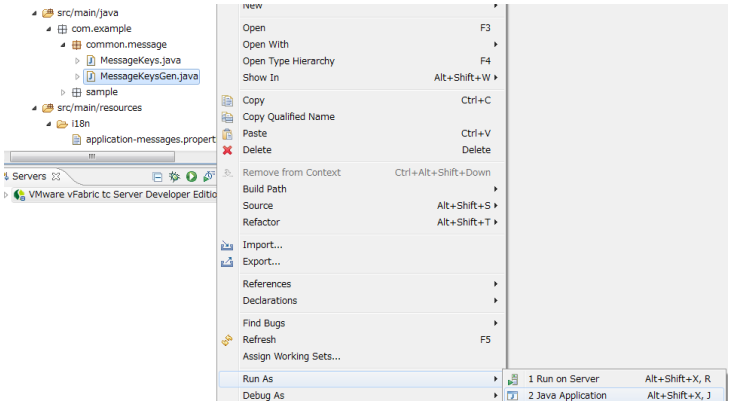
3. Provision of message properties file

   Define the messages in src/main/resource/i18m/application-messages.properties. The settings are carried out as follows:

```
i.ex.an.0001={0} upload completed.
w.ex.an.2001=The recommended change interval has passed password. Please␣
↪change your password.
e.ex.an.8001=Cannot upload, Because the file size must be less than {0}MB.
```

```
e.ex.an.9001=There are inconsistencies in the data.
```

4. Execution of auto-generation class



`MessageKeys` class is overwritten as follows:

```
package com.example.common.message;
/**
 * Message Id
 */
public class MessageKeys {
    /** i.ex.an.0001={0} upload completed. */
    public static final String I_EX_AN_0001 = "i.ex.an.0001";
    /** w.ex.an.2001=The recommended change interval has passed password.␣
↪Please change your password. */
    public static final String W_EX_AN_2001 = "w.ex.an.2001";
    /** e.ex.an.8001=Cannot upload, Because the file size must be less␣
↪than {0}MB. */
    public static final String E_EX_AN_8001 = "e.ex.an.8001";
    /** e.ex.an.9001=There are inconsistencies in the data. */
    public static final String E_EX_AN_9001 = "e.ex.an.9001";
}
```

4.7 Internationalization

4.7.1 Overview

Internationalization is a process wherein the display of labels and messages in an application is not fixed to a specific language. It supports multiple languages. The language switching can be achieved by specifying a unit called “Locale” expressing language, country and region.

This section explains how to internationalize messages to display on the screen.

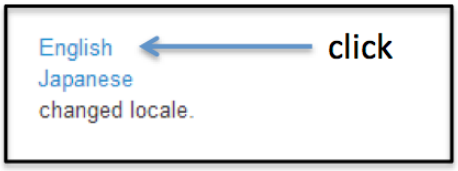
In order to internationalization, following requrements should be meeted.

- Text elements on the screen(code name, messages, labels of GUI components etc.) should be retrieve from external definitions such as properties file. (should not be hard-coding in the source code)
- The mechanism to specify the locale of the clients should be provided.

Methods to specify the locale are as follows:

- Using standard request header (specify the locale by language settings of browsers)
- Saving the locale into the Cookie using request parameter
- Saving the locale into the Session using request parameter

The image of switching locale is as follows:





Note: For internationalization of Codelist, refer to *Codelist*.

Note: When the error screen is to be internationalised, transition to error screen is performed by using MVC Controller of Spring. If a direct transition to error screen is performed without Spring MVC, it may happen that the message is not output in intended language.

For details, refer *Measures when Internationalization is not applied*.

Tip: The most commonly known abbreviation of internationalization is i18n. Internationalization is abbreviated

as i18n because the number of letters between the first “i” and the last “n” is 18 i.e. “nternationalizatio”.

4.7.2 How to use

Configuration to define messages

To internationalize the messages on the screen, use the one of following `MessageSource`imelementations for managing messages.

- `org.springframework.context.support.ResourceBundleMessageSource`
- `org.springframework.context.support.ReloadableResourceBundleMessageSource`

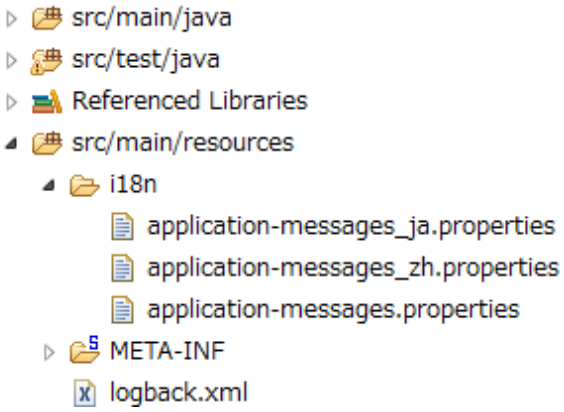
The information here explains an example of using the `ResourceBundleMessageSource`.

applicationContext.xml

```
<bean id="messageSource"
    class="org.springframework.context.support.ResourceBundleMessageSource">
    <property name="basenames">
        <list>
            <value>i18n/application-messages</value>  <!-- (1) -->
        </list>
    </property>
</bean>
```

Sr. No.	Description
(1)	Specify `i18n/application-messages` as base name of properties file. It is recommended to store message properties file under i18n directory to support internationalization. For MessageSource details and definition methods, refer to *Message Management*.

Example of storing properties files



Properties file should be created in accordance with the following rules.

- File name should be defined in `application-messages_XX.properties` format. (Specify locale in XX portion)
- The messages defined in `application-messages.properties` should be created in default language.
- **Make sure you create** `application-messages.properties`. If it does not exist, messages cannot be fetched from `MessageSource` and `JspTagException` occurs while setting the messages in JSP.

When creating property files as above, it is determined which to use the file as follows:

- When the locale resolved by `LocaleResolver` is zh, `application-messages_zh.properties` is used.
- when the locale resolved by `LocaleResolver` is ja, `application-messages_ja.properties` is used.
- When properties file does not exist corresponding to the locale resolved by `LocaleResolver`, `application-messages.properties` is used as default. ("_XX" portion does not exist in file name)

Note: The locate to use is determined in the following order until a properties file is found corresponding to the locale.

1. Locale sent from clients
2. Locale specified by JVM on which application server runs (it may not be set in some cases)
3. Locale specified by OS on which application server runs

It is frequently misunderstood that default properties file is used when properties file does not exist corresponding to the locale sent from clients . In this case, then it is checked whether the file is available corresponding to the

locale specified by the application server. If not found, finally the default properties file is used.

Tip: For description of message properties file, refer to *Message Management*.

Changing locale as per browser settings

Settings of AcceptHeaderLocaleResolver

If switch the locale using browser settings, use the `AcceptHeaderLocaleResolver`.

spring-mvc.xml

```
<bean id="localeResolver"
    class="org.springframework.web.servlet.i18n.AcceptHeaderLocaleResolver" /> <!--
↪ (1) -->
```

Sr. No.	Description
(1)	Specify `org.springframework.web.servlet.i18n.AcceptHeaderLocaleResolver` in id attribute “localeResolver” of bean tag. If this `LocaleResolver` is used, HTTP header “accept-language” is added for each request and locale gets specified.

Note: When `LocaleResolver` is not set, `org.springframework.web.servlet.i18n.AcceptHeaderLocaleRes` is used by default; hence `LocaleResolver` need not be set.

Definition of messages

See the example of definition of messages below.

application-messages.properties

```
title.admin.top = Admin Top
```

application-messages_ja.properties

```
title.admin.top = 管理画面 Top
```

Implementation of JSP

See the example of implementaion of messages below.

include.jsp(Common jsp file to be included)

```
<%@ page session="false"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt"%>
<%@ taglib uri="http://www.springframework.org/tags" prefix="spring"%>  <!-- (1) --
↪>
<%@ taglib uri="http://www.springframework.org/tags/form" prefix="form"%>
<%@ taglib uri="http://www.springframework.org/security/tags" prefix="sec"%>
<%@ taglib uri="http://terasoluna.org/functions" prefix="f"%>
<%@ taglib uri="http://terasoluna.org/tags" prefix="t"%>
```

Sr. No.	Description
(1)	When message is to be output in JSP, it is output using Spring tag library; hence custom tag needs to be defined. `<%@taglib uri="http://www.springframework.org/tags" prefix="spring"%>` should be defined.

Note: For details on common jsp files to be included, refer to *Creating common JSP for include*.

JSP file for screen display

```
<spring:message code="title.admin.top" />  <!-- (2) -->
```

Sr. No.	Description
(2)	Output the message using <spring:message> which is a Spring tag library of JSP. In code attribute, set the key specified in properties. In this example, if locale is ja, “管理画面 Top” is output and for other locales, “Admin Top” is output.

Changing locale depending on screen operations dynamically

The method of dynamic changing the locale depending on screen operations etc. is effective in case of selecting a specific language irrespective of user terminal (browser) settings.

Following is an example of changing locale depending on screen operations.

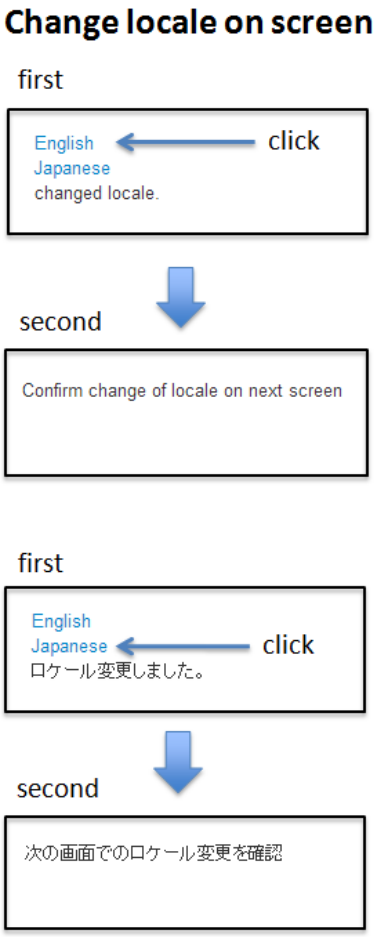
To use the language selected by a user, chooose `org.springframework.web.servlet.i18n.LocaleChangeIntercep`

`LocaleChangeInterceptor`is an interceptor to save the locale value specified by the request parameter using `org.springframework.web.servlet.LocaleResolver`.

Select the implementation class of `LocaleResolver` from the following table.

Table.4.42 **Types of LocaleResolver**

No	Implementation class	How to save locale
1.	`org.springframework.web.servlet.i18n.SessionLocaleResolver`	Save in server(using `HttpSession`)
2.	`org.springframework.web.servlet.i18n.CookieLocaleResolver`	Save in client(using `Cookie`)



Note: When `org.springframework.web.servlet.i18n.AcceptHeaderLocaleResolver` is used in `LocaleResolver`, locale cannot be changed dynamically using `org.springframework.web.servlet.i18n.LocaleChangeInterceptor`.

How to define LocaleChangeInterceptor

If switching the locale using the request parameter, use the `LocaleChangeInterceptor`.

spring-mvc.xml

```
<mvc:interceptors>
  <mvc:interceptor>
    <mvc:mapping path="/**" />
```

```
    <mvc:exclude-mapping path="/resources/**" />
    <mvc:exclude-mapping path="/**/*.html" />
    <bean
      class="org.springframework.web.servlet.i18n.LocaleChangeInterceptor">  <!--␣
↪(1) -->
    </bean>
    <!-- omitted -->
  </mvc:interceptor>
</mvc:interceptors>
```

Sr. No.	Description
(1)	Define `org.springframework.web.servlet.i18n.LocaleChangeInterceptor` in interceptor of Spring MVC.

Note: How to change the name of request parameter to specify locale

```
<bean
    class="org.springframework.web.servlet.i18n.LocaleChangeInterceptor">
    <property name="paramName" value="lang"/>  <!-- (2) -->
</bean>
```

Sr. No.	Description
(2)	In `paramName`property, specify the name of request parameter. In this example, it is "request URL?lang=xx". **When paramName property is omitted, "locale" gets set.** With "request URL?locale=xx", it becomes *enabled*.

How to define SessionLocaleResolver

If saving the locale in the server side, use the `SessionLocaleResolver`.

spring-mvc.xml

```
<bean id="localeResolver" class="org.springframework.web.servlet.i18n.
↪SessionLocaleResolver">  <!-- (1) -->
    <property name="defaultLocale" value="en"/>  <!-- (2) -->
</bean>
```

Sr. No.	Description
(1)	Define id attribute of bean tag in "localeResolver" and specify the class wherein `org.springframework.web.servlet.LocaleResolver` is implemented. In this example, `org.springframework.web.servlet.i18n.SessionLocaleResolver` that stores locale in session is specified. id attribute of bean tag should be set as "localeResolver". By performing these settings, `SessionLocaleResolver` will be used at the `LocaleChangeInterceptor`.
(2)	Specify Locale in `defaultLocale` property. When Locale cannot be fetched from the session, setup value of `value` becomes valid. **Note:** When `defaultLocale` property is omitted, Locale set in the user terminal (browser) becomes valid.

How to define CookieLocaleResolver

If saving the locale in the client side, use the `CookieLocaleResolver`.

spring-mvc.xml

```
<bean id="localeResolver" class="org.springframework.web.servlet.i18n.
↪CookieLocaleResolver">  <!-- (1) -->
      <property name="defaultLocale" value="en"/>  <!-- (2) -->
      <property name="cookieName" value="localeCookie"/>  <!-- (3) -->
</bean>
```

Sr. No.	Description
(1)	In id attribute "localeResolver" of bean tag, specify `org.springframework.web.servlet.i18n.CookieLocaleResolver`. id attribute of bean tag should be set as "localeResolver". By performing these settings, `CookieLocaleResolver` will be used at the `LocaleChangeInterceptor`.
(2)	Specify Locale in `defaultLocale` property. When Locale cannot be fetched from Cookie, setup value of `value` becomes valid. **Note:** When `defaultLocale` property is omitted, Locale configured in the user terminal (browser) becomes valid.
(3)	The value specified in `cookieName` property is used as cookie name. If not specified, the value of `org.springframework.web.servlet.i18n.CookieLocaleResolver.LOCALE`is used as default. **It is recommended to change not to tell the user explicitly Spring Framework is used.**

Messages settings

See the example below for messages settings.

application-messages.properties

```
i.xx.yy.0001 = changed locale
i.xx.yy.0002 = Confirm change of locale at next screen
```

application-messages_ja.properties

```
i.xx.yy.0001 = Locale を変更しました。
i.xx.yy.0002 = 次の画面での Locale 変更を確認
```

Implementation of JSP

See the example of implementation of JSP below.

JSP file for screen display

```
<a href='${pageContext.request.contextPath}?locale=en'>English</a>  <!-- (1) -->
<a href='${pageContext.request.contextPath}?locale=ja'>Japanese</a>
<spring:message code="i.xx.yy.0001" />
```

Sr. No.	Description
(1)	Submit the request parameter to switch the locale. Request parameter name is specified in `paramName` property of `LocaleChangeInterceptor`. (In the example above, the default parameter name is used) In the above example, it is changed to English locale in English link and to Japanese locale in Japanese link. Hereafter, the selected locale is enabled. As “en” properties file does not exist, English locale is read from properties file by default.

Tip:

- Spring tag library should be defined in common jsp files to be included.
- For details on common jsp files to be included, refer to *Creating common JSP for include*.

Measures when Internationalization is not applied

Since `LocaleChangeInterceptor` is an interceptor which is called while executing `Controller` process of Spring MVC, it should be noted that internationalization is not applied when the transition is not done via `Controller`.

For example, when a JSP file is to be specified directly in the settings for transition to error screen, `Controller` is not used in the transition to error screen. In such a case, in order to internationalize error screen, a Controller must be created for transition to error screen and it must be set so as to enable use of `LocaleChangeInterceptor`, by using it in transition to error screen.

Note: Similarly, when a transition is to be performed by specifying JSP directly, Tiles is not applied since it does not pass through `ViewResolver` used by *Screen Layout using Tiles*.

Implementation of *Authorization* is shown below with an example, for the setting method.

Example for transition to error screen wherein LocaleChangeInterceptor is not applied

- spring-security.xml

```
<sec:http>
    <!-- omitted -->
    <sec:access-denied-handler
        error-page="/WEB-INF/views/common/error/accessDeniedError.jsp" /> <!-- (1)␣
↪-->
    <!-- omitted -->
</sec:http>
```

Sr. No.	Description
(1)	Specify error screen for authorization error in `error-page` attribute of `<sec:access-denied-handler>` tag, by JSP.

Example for transition to error screen wherein LocaleChangeInterceptor is applied

- spring-security.xml

```
<sec:http>
    <!-- omitted -->
    <sec:access-denied-handler
        error-page="/common/error/accessDeniedError" /> <!-- (1) -->
    <!-- omitted -->
</sec:http>
```

Sr. No.	Description
(1)	Specify a path for transition to error screen for authorization error, in `error-page` attribute of `<sec:access-denied-handler>` tag.

- Controller class

```
@Controller
@RequestMapping("/common/error") // (1)
public class ErrorController {

    @RequestMapping("accessDeniedError") // (1)
    public String accessDeniedError() {
        return "common/error/accessDeniedError"; // (2)
    }

}
```

Sr. No.	Description
(1)	Define request mapping for transition to error screen.
(2)	Return View name of error screen for transition.

Warning: Generally, it is recommended not to use `<mvc:view-controller>` since transition to error screen is likely to occur by GET request as well as POST request.

For notes while using `<mvc:view-controller>`, refer *HTML response*.

4.8 Codelist

4.8.1 Overview

A codelist is a pair comprising of “Code values (Value) and their display names (Label)”.

It is used for mapping code values with the labels to be displayed on screen such as selectbox.

Common library provides the following functionalities:

- Functionality to read and cache the codelist defined in xml file or DB at the time of launching the application.
- Functionality to refer to the codelist from JSP or Java class
- Functionality to perform input validation using codelist

Moreover, it also supports

- internationalization of codelist
- reloading of cached codelist

Note: As per standard specifications, only the codelist defined in DB is reloadable.

Following four types of codelists are implemented in common library.

Table.4.44 **Types of codelist**

Types	Contents	Reloadable
`org.terasoluna.gfw.common.codelist.SimpleMapCodeList`	Use the hard-coded contents from xml file.	NO
`org.terasoluna.gfw.common.codelist.NumberRangeCodeList`	Use when creating number range list.	NO
`org.terasoluna.gfw.common.codelist.JdbcCodeList`	Use the codelist by fetching the code from DB using SQL.	YES
`org.terasoluna.gfw.common.codelist.EnumCodeList`	Use when creating the codelist from constant defined in `Enum` class.	NO
`org.terasoluna.gfw.common.codelist.i18n.SimpleI18nCodeList`	Use the codelist corresponding to java.util.Locale.	NO

Common library provides `org.terasoluna.gfw.common.codelist.CodeList` for the interface of above codelists.

Codelist class diagram provided in common library is as follows:

4.8.2 How to use

This section describes settings for various codelists and their implementation methods.

- *Using SimpleMapCodeList*
- *Using NumberRangeCodeList*
- *Using JdbcCodeList*
- *How to use EnumCodeList*
- *How to use SimpleI18nCodeList*
- *Display code name corresponding to code value*
- *Input validation of code value using codelist*

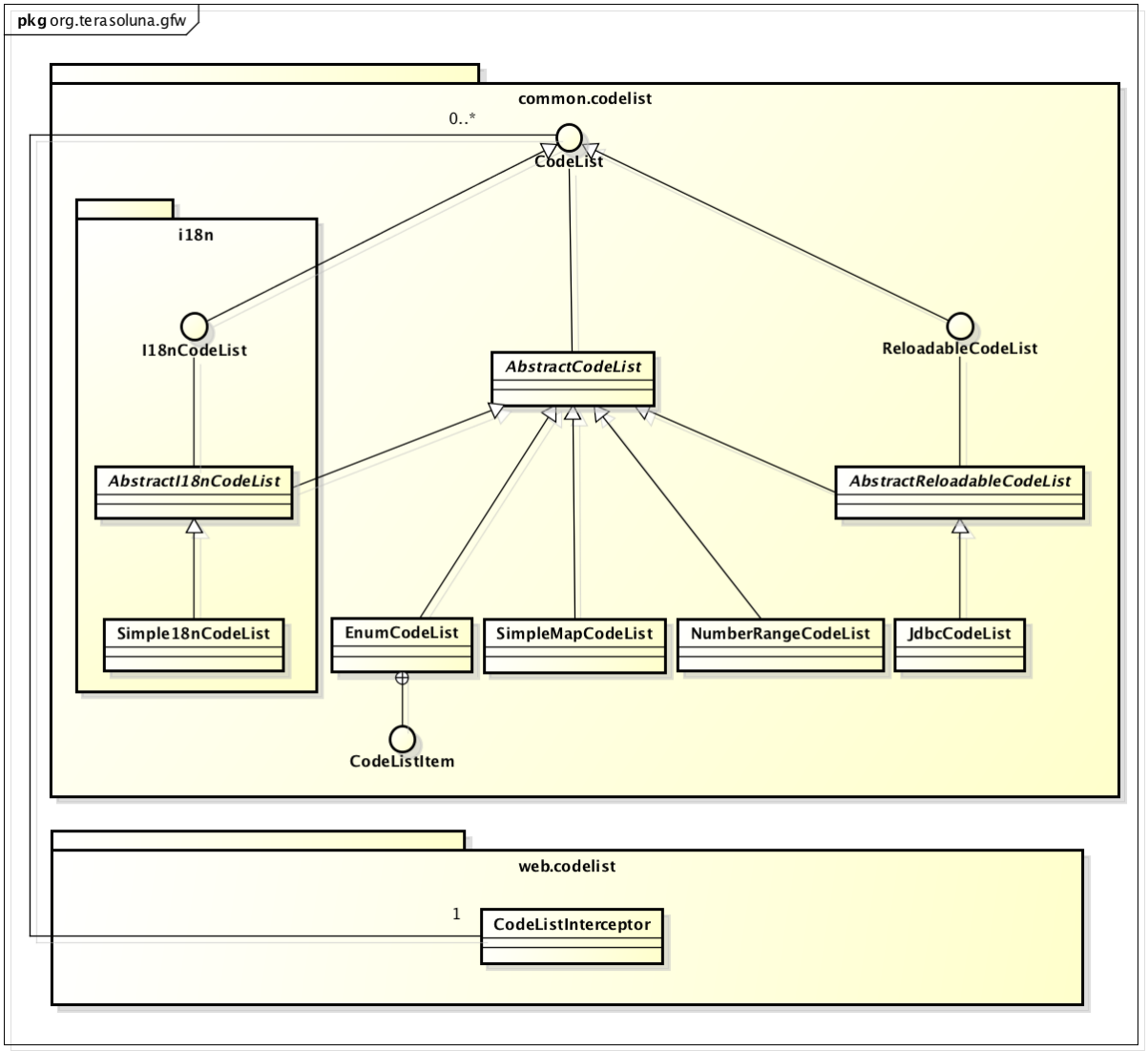


Figure.4.17 **Picture - Image of codelist class diagram**

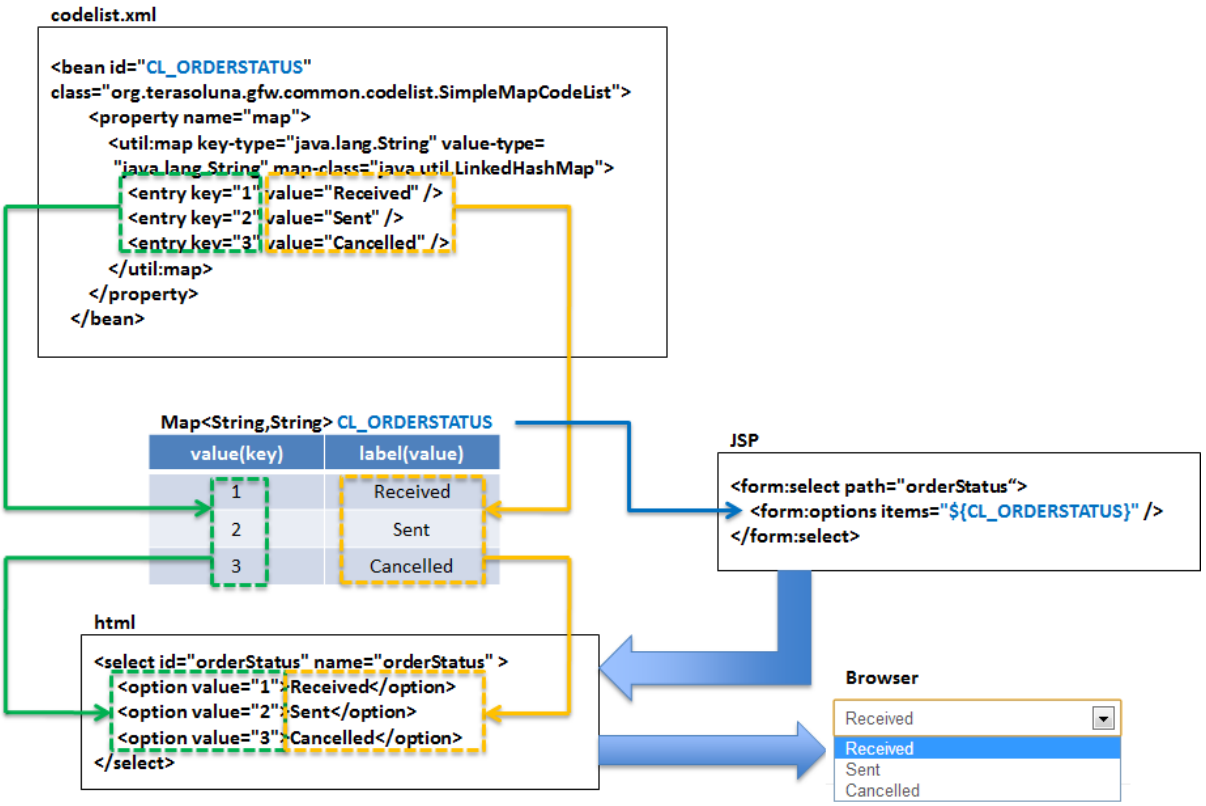
Using SimpleMapCodeList

`org.terasoluna.gfw.common.codelist.SimpleMapCodeList` reads the code values defined in xml file at the time of launching the application and uses them as is.

SimpleMapCodeList image

Example of codelist settings

Definition of Bean definition file(xxx-codelist.xml)



It is recommended to create a bean definition file for codelist.

```
<bean id="CL_ORDERSTATUS" class="org.terasoluna.gfw.common.codelist.
↪SimpleMapCodeList"> <!-- (1) -->
    <property name="map">
        <util:map>
            <entry key="1" value="Received" /> <!-- (2) -->
            <entry key="2" value="Sent" />
            <entry key="3" value="Cancelled" />
        </util:map>
    </property>
</bean>
```

Sr. No.	Description
(1)	Define a bean of SimpleMapCodeList class. beanID should have the name matching with the ID pattern of `org.terasoluna.gfw.web.codelist.CodeListInterceptor` described later.
(2)	Define Key, Value pairs of Map. When map-class attribute is omitted, it is registered in `java.util.LinkedHashMap`; hence in the above example, “Name and value” are stored in Map in the order of registration.

Definition of Bean definition file(xxx-domain.xml)

Once the bean definition file for codelist is created, it should be imported to already existing bean definition file.

```
<import resource="classpath:META-INF/spring/projectName-codelist.xml" /> <!-- (3)
↪-->
<context:component-scan base-package="com.example.domain" />

<!-- omitted -->
```

Sr. No.	Description
(3)	Import bean definition file for codelist. Resource information of import is necessary during component-scan; hence import should be set above `<context:component-scan base-package="com.example.domain" />`.

Using codelist in JSP

By using the interceptor of common library, codelist can be set automatically in request scope and can be easily referred from JSP.

Definition of Bean definition file(spring-mvc.xml)

```
<mvc:interceptors>
  <mvc:interceptor>
    <mvc:mapping path="/**" /> <!-- (1) -->
    <bean
      class="org.terasoluna.gfw.web.codelist.CodeListInterceptor"> <!-- (2) -->
      <property name="codeListIdPattern" value="CL_.+" /> <!-- (3) -->
    </bean>
  </mvc:interceptor>

  <!-- omitted -->

</mvc:interceptors>
```

Sr. No.	Description
(1)	Set the applicable path.
(2)	Define a bean of CodeListInterceptor class.
(3)	Set the beanID pattern of codelist which is automatically set in the request scope. In pattern, regular expression used in `java.util.regex.Pattern` should be set. In the above example, only the data in which id is defined in “CL_XXX” format is targeted. In that case, bean definition wherein id does not start with “CL_” should not be imported. beanID defined in “CL_” can be used in JSP since it is set in the request scope. `codeListIdPattern` property can be omitted. If omitting `codeListIdPattern` property, all of `CodeLists` (all beans which implements `org.terasoluna.gfw.common.codelist.CodeList`) are available in JSP.

Example of implementing the codelist in jsp

```
<form:select path="orderStatus">
  <form:option value="" label="--Select--" /> <!-- (4) -->
  <form:options items="${CL_ORDERSTATUS}" /> <!-- (5) -->
```

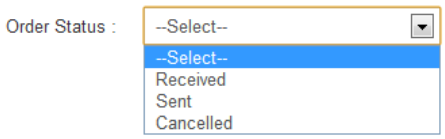
```
</form:select>
```

Sr. No.	Description
(4)	When setting dummy value at the top of the selectbox, null characters should be specified in the value.
(5)	Specify the beanID for which codelist is defined.

Output HTML

```
<select id="orderStatus" name="orderStatus">
   <option value="">"--Select--</option>
   <option value="1">Received</option>
   <option value="2">Sent</option>
   <option value="3">Cancelled</option>
</select>
```

Output screen



Using codelist in Java class

When using the codelist in Java class, inject the codelist by setting `javax.inject.Inject` annotation and `javax.inject.Named` annotation. Specify the codelist name in `@Named` annotation.

```
import javax.inject.Named;

import org.terasoluna.gfw.common.codelist.CodeList;

public class OrderServiceImpl implements OrderService {

    @Inject
    @Named("CL_ORDERSTATUS")
    CodeList orderStatusCodeList; // (1)
```

```
    public boolean existOrderStatus(String target) {
        return orderStatusCodeList.asMap().containsKey(target); // (2)
    }
}
```

Sr. No.	Description
(1)	Inject the codelist with beanID “CL_ORDERSTATUS”.
(2)	Fetch the codelist in `java.util.Map` format using CodeList#asMap method.

Using NumberRangeCodeList

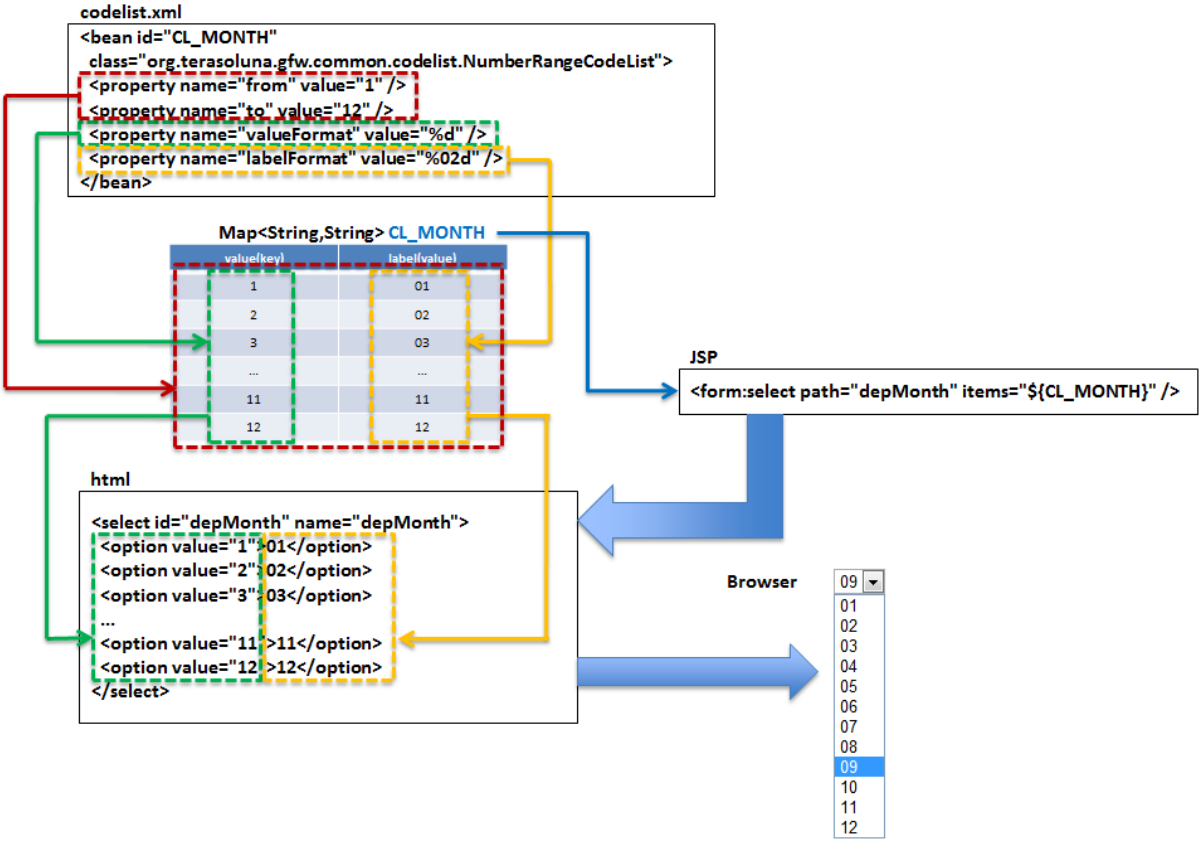
`org.terasoluna.gfw.common.codelist.NumberRangeCodeList` is a codelist that creates the list of numeric values of specified range at the time of launching the application. It is assumed that this codelist will mainly be used in the selectboxes having only numbers i.e. selectbox for month, date etc.

Image of NumberRangeCodeList

Tip: NumberRangeCodeList supports only Arabic numbers and does not support Chinese and Roman numbers. Chinese and Roman numbers can be supported by using JdbcCodeList and SimpleMapCodeList.

NumberRangeCodeList has the following features:

1. In order to set From value < To value, the values increased in accordance with the interval are set in From-To range in ascending order.
2. In order to set To value < From value, the values decreased in accordance with the interval are set in To-From range in descending order.
3. Increment (decrement) can be changed by setting intervals.



The information here describes how to configure the ascending `NumberRangeCodeList`. For how to create the descending `NumberRangeCodeList` or change interval, refer to "*Variations of NumberRangeCodeList*". |

Example of codelist settings

Example of setting From value < To value is shown below.

Definition of Bean definition file(xxx-codelist.xml)

```
<bean id="CL_MONTH"
    class="org.terasoluna.gfw.common.codelist.NumberRangeCodeList"> <!-- (1) -->
    <property name="from" value="1" /> <!-- (2) -->
    <property name="to" value="12" /> <!-- (3) -->
    <property name="valueFormat" value="%d" /> <!-- (4) -->
    <property name="labelFormat" value="%02d" /> <!-- (5) -->
    <property name="interval" value="1" /> <!-- (6) -->
</bean>
```

Sr. No.	Description
(1)	Define a bean of NumberRangeCodeList.
(2)	Specify the range start value. When omitted, “0” is set as range start value.
(3)	Specify the range end value. It cannot be blank.
(4)	Specify the format of the code value. Format used should be `java.lang.String.format`. When omitted, “%s” is set.
(5)	Specify the format of the code name. Format used should be `java.lang.String.format`. When omitted, “%s” is set.
(6)	Set the increment value. When omitted, “1” is set.

Using codelist in JSP

For details on settings shown below, refer to *Using codelist in JSP* described earlier.

Example of jsp implementation

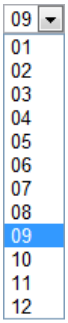
```
<form:select path="depMonth" items="${CL_MONTH}" />
```

Output HTML

```
<select id="depMonth" name="depMonth">
  <option value="1">01</option>
  <option value="2">02</option>
  <option value="3">03</option>
  <option value="4">04</option>
```

```
  <option value="5">05</option>
  <option value="6">06</option>
  <option value="7">07</option>
  <option value="8">08</option>
  <option value="9">09</option>
  <option value="10">10</option>
  <option value="11">11</option>
  <option value="12">12</option>
</select>
```

Output screen



Using codelist in Java class

For details on settings shown below, refer to *Using codelist in Java class* described earlier.

Using JdbcCodeList

`org.terasoluna.gfw.common.codelist.JdbcCodeList` is a class for creating codelist by fetching values from DB at the time of starting the application.

Since `JdbcCodeList` creates a cache while starting the application, no delay occurs during DB access when you want to display a list.

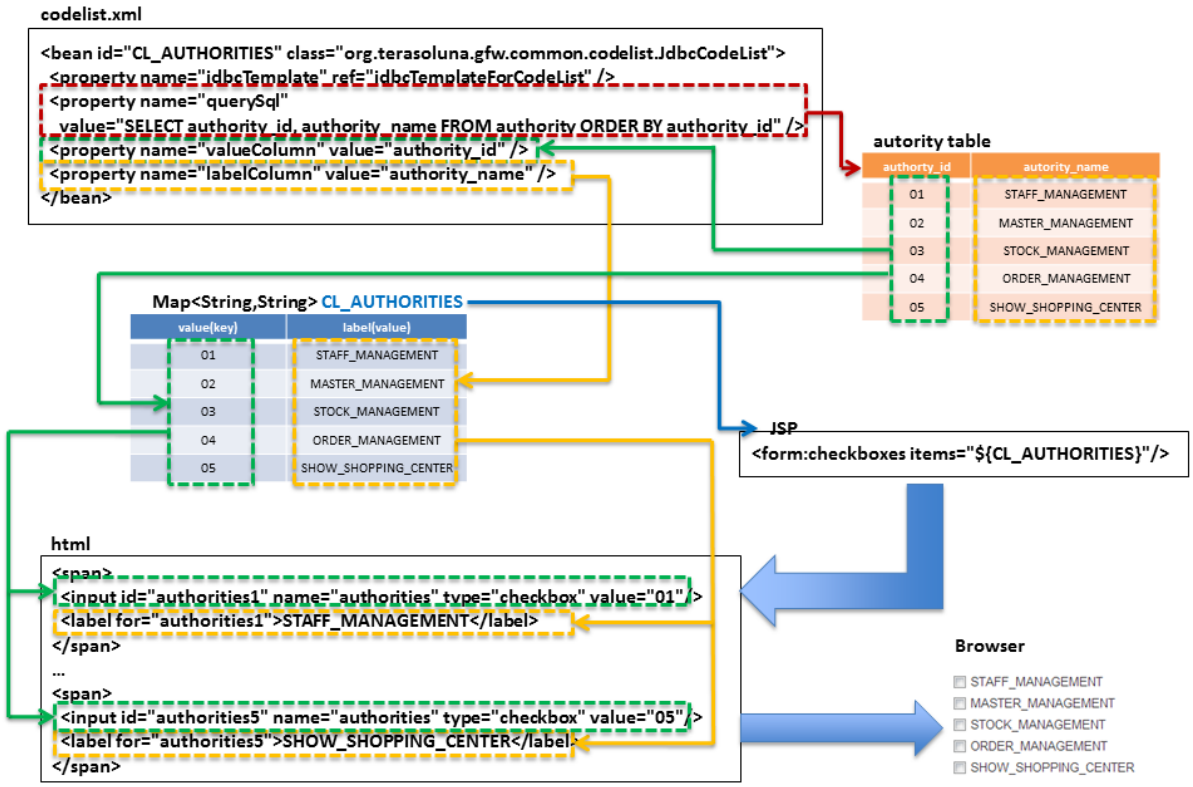
If you want to reduce the read time for the startup, it is preferable to set an upper limit on the number of acquisitions.

`JdbcCodeList` consists of a field which sets the `org.springframework.jdbc.core.JdbcTemplate`.

If an upper limit is set for the `fetchSize` of `JdbcTemplate`, the records only till the upper limit are read at the startup.

The fetched values can be changed dynamically by reloading. For details, refer to *When reloading the codelist*.

JdbcCodeList image



Example of codelist settings

Definition of Table(authority)

authority_id	authority_name
01	STAFF_MANAGEMENT
02	MASTER_MANAGEMENT
03	STOCK_MANAGEMENT
04	ORDER_MANAGEMENT
05	SHOW_SHOPPING_CENTER

Definition of Bean definition file(xxx-codelist.xml)

```
<bean id="jdbcTemplateForCodeList" class="org.springframework.jdbc.core.
↪JdbcTemplate" > <!-- (1) -->
    <property name="dataSource" ref="dataSource" />
    <property name="fetchSize" value="${codelist.jdbc.fetchSize:1000}" /> <!-- (2)␣
↪-->
</bean>

<bean id="AbstractJdbcCodeList"
    class="org.terasoluna.gfw.common.codelist.JdbcCodeList" abstract="true"> <!--␣
↪(3) -->
    <property name="jdbcTemplate" ref="jdbcTemplateForCodeList" /> <!-- (4) -->
</bean>

<bean id="CL_AUTHORITIES" parent="AbstractJdbcCodeList" > <!-- (5) -->
    <property name="querySql"
        value="SELECT authority_id, authority_name FROM authority ORDER BY␣
↪authority_id" /> <!-- (6) -->
    <property name="valueColumn" value="authority_id" /> <!-- (7) -->
    <property name="labelColumn" value="authority_name" /> <!-- (8) -->
</bean>
```

Sr. No.	Description
(1)	Define a bean for `org.springframework.jdbc.core.JdbcTemplate` class. It is necessary for independently setting the fetchSize.
(2)	Set the fetchSize. An appropriate value must be set since FetchSize is set to Fetch All by default. When `fetchSize` is set to "fetch all" and when the records that are required to be read by `JdbcCodeList` are large, process efficiency while fetching a list from DB is likely to reduce resulting in prolonged startup time of application.
(3)	Define a common bean of JdbcCodeList. Common parts of another `JdbcCodeList` are specified. Therefore, the bean is defined in parent class for bean definition of basic `JdbcCodeList`. An instance cannot be created for this bean by setting abstract attribute to true.
(4)	Specify `jdbcTemplate` set in (1). `JdbcTemplate` which specifies `fetchSize` is stored in `JdbcCodeList`.
(5)	Bean definition of JdbcCodeList By setting Bean defined in (3) as parent class in parent attribute, `JdbcCodeList` which specifies `fetchSize` is set. In this bean definition, only the query related settings are carried out and the required CodeList is created.
(6)	Write an SQL to be fetched in querySql property. At that time, **always specify "ORDER BY" clause and determine the order**. If "ORDER BY" is not specified, the order gets changed while fetching the records.
(7)	Set the value corresponding to the Key of Map in valueColumn property. In this example, authority_id is set.

次のページに続く

Table. 4.45 – 前のページからの続き

Sr. No.	Description
(8)	Set the value corresponding to Value of Map in labelColumn property. In this example, authority_name is set.

Using codelist in JSP

For details on settings shown below, refer to *Using codelist in JSP* described earlier.

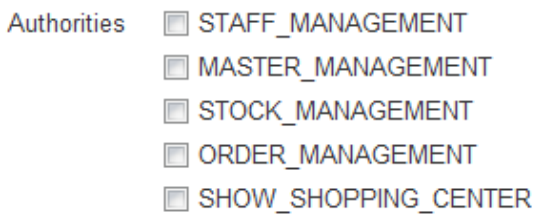
Example of jsp implementation

```
<form:checkboxes items="${CL_AUTHORITIES}"/>
```

Output HTML

```
<span>
  <input id="authorities1" name="authorities" type="checkbox" value="01"/>
  <label for="authorities1">STAFF_MANAGEMENT</label>
</span>
<span>
  <input id="authorities2" name="authorities" type="checkbox" value="02"/>
  <label for="authorities2">MASTER_MANAGEMENT</label>
</span>
<span>
  <input id="authorities3" name="authorities" type="checkbox" value="03"/>
  <label for="authorities3">STOCK_MANAGEMENT</label>
</span>
<span>
  <input id="authorities4" name="authorities" type="checkbox" value="04"/>
  <label for="authorities4">ORDER_MANAGEMENT</label>
</span>
<span>
  <input id="authorities5" name="authorities" type="checkbox" value="05"/>
  <label for="authorities5">SHOW_SHOPPING_CENTER</label>
</span>
```

Output screen



Using codelist in Java class

For details on settings shown below, refer to *Using codelist in Java class* described earlier.

How to use EnumCodeList

`org.terasoluna.gfw.common.codelist.EnumCodeList` is a class for creating codelist from constant defined in `Enum` class.

Note: In case of handling codelist in applications that match with the following conditions, it should be analyzed if the codelist label can be stored in `Enum` class using `EnumCodeList` . By storing codelist label in `Enum` class, the information and operations linked with code values can be aggregated in `Enum` class.

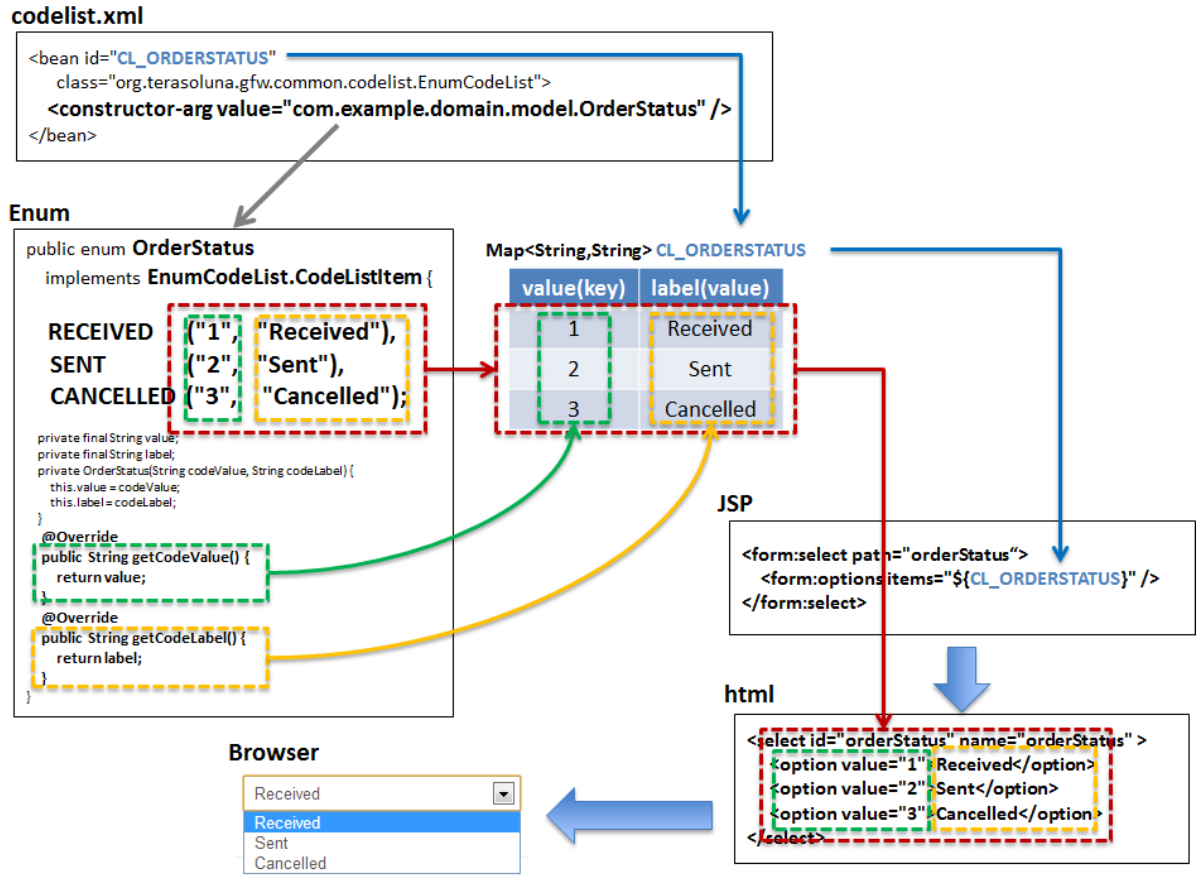
- It is necessary to store the code values in `Enum` class (i.e. the process needs to be performed considering code values in Java logic)
- Internationalization (multilingualization) of UI is not required

---

Image of using `EnumCodeList` is shown below.

Note: In `EnumCodeList`, `org.terasoluna.gfw.common.codelist.EnumCodeList.CodeListItem` interface is provided to fetch the information (code values and labels) required for creating codelist from `Enum` class.

In case of using `EnumCodeList`, `EnumCodeList.CodeListItem` interface should be implemented in `Enum` class to be created.



Example of codelist settings

Creating Enum class

In case of using `EnumCodeList`, create `Enum` class that implements `EnumCodeList.CodeListItem` interface. Example is shown below.

```
package com.example.domain.model;

import org.terasoluna.gfw.common.codelist.EnumCodeList;

public enum OrderStatus
    // (1)
    implements EnumCodeList.CodeListItem {

    // (2)
    RECEIVED  ("1", "Received"),
    SENT      ("2", "Sent"),
    CANCELLED ("3","Cancelled");

    // (3)
    private final String value;
    private final String label;
```

```
    // (4)
    private OrderStatus(String codeValue, String codeLabel) {
        this.value = codeValue;
        this.label = codeLabel;
    }

    // (5)
    @Override
    public String getCodeValue() {
        return value;
    }

    // (6)
    @Override
    public String getCodeLabel() {
        return label;
    }

}
```

Sr. No.	Description
(1)	In `Enum` class to be used as codelist, implement the `org.terasoluna.gfw.common.codelist.EnumCode` interface provided by common library. In `EnumCodeList.CodeListItem` interface, following methods are defined to fetch the information (code values and labels) required for creating a codelist. • `getCodeValue()` method to fetch code values • `getCodeLabel()` method to fetch labels
(2)	Define constants. When creating constants, specify the information (code values and labels) required for creating a codelist. In above example, following 3 constants are defined. • `RECEIVED` (code value="`1`", label="`Received`") • `SENT` (code value="`2`", label="`Sent`") • `CANCELLED` (code value="`3`", label="`Cancelled`") **Note:** Sorting order of codelist when using `EnumCodeList` will be the order of defining constants.
(3)	Create a property to store the information (code values and labels) required for creating a codelist.
(4)	Create a constructor to receive the information (code values and labels) required for creating a codelist.
(5)	Return the code values storing constants. This method is defined in `EnumCodeList.CodeListItem` interface, and it is called when `EnumCodeList` fetches code value from a constant.
(6)	Return the label storing constants. This method is defined in `EnumCodeList.CodeListItem` interface, and it is called when `EnumCodeList` fetches label from a constant.

Definition of bean definition file (xxx-codelist.xml)

`EnumCodeList` is defined in bean definition file for codelist. Example of definition is shown below.

```
<bean id="CL_ORDERSTATUS"
      class="org.terasoluna.gfw.common.codelist.EnumCodeList"> <!-- (7) -->
    <constructor-arg value="com.example.domain.model.OrderStatus" /> <!-- (8) -->
</bean>
```

Sr. No.	Description
(7)	Specify `EnumCodeList` class as codelist implementation class.
(8)	Specify FQCN of `Enum` class that implements `EnumCodeList.CodeListItem` interface in constructor of `EnumCodeList` class.

Using codelist in JSP

For details on how to use codelist in JSP, refer to *Using codelist in JSP* described earlier.

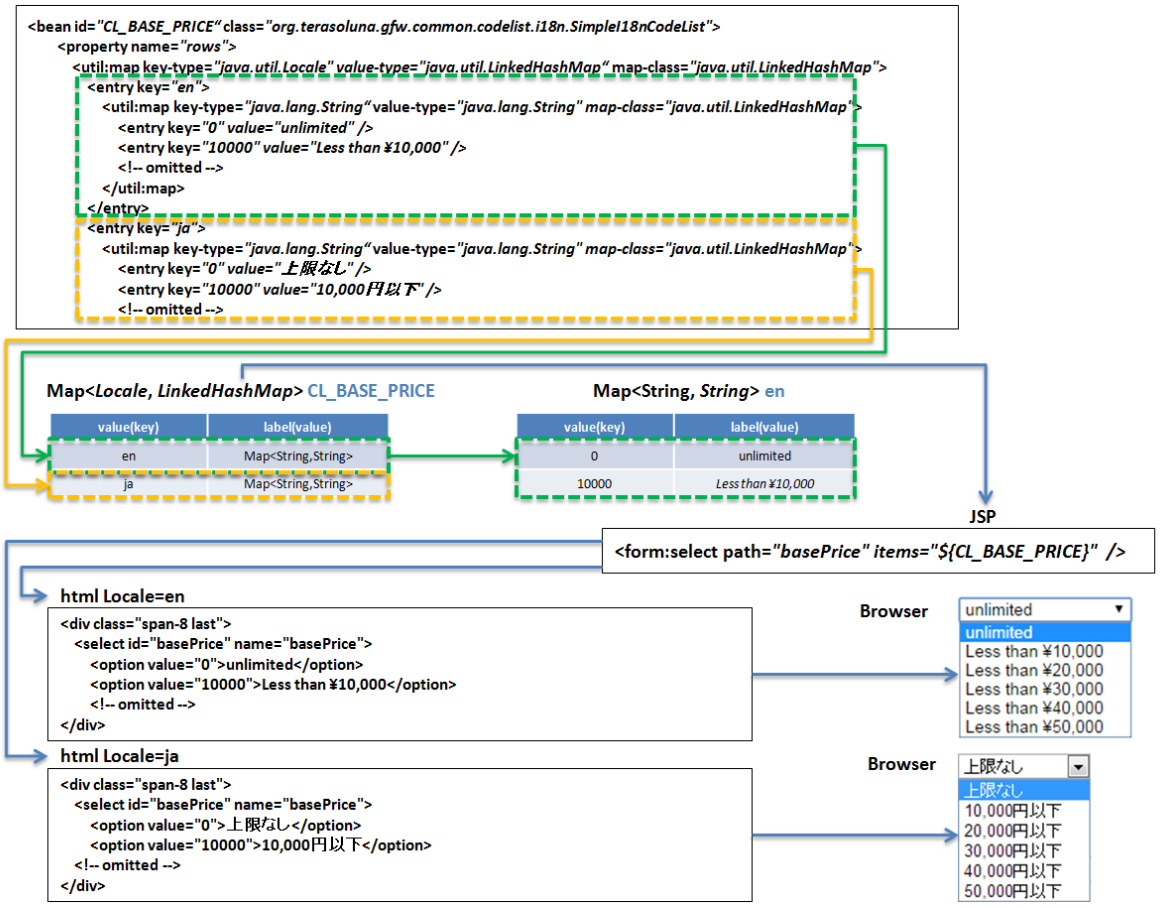
Using codelist in Java class

For details on how to use codelist in Java class, refer to *Using codelist in Java class* described earlier.

How to use SimpleI18nCodeList

`org.terasoluna.gfw.common.codelist.i18n.SimpleI18nCodeList` is a codelist supporting internationalization. By setting the codelist for each locale, the codelist corresponding to locale can be returned.

SimpleI18nCodeList image



Example of codelist settings

It is easier to understand if you consider `SimpleI18nCodeList` as two dimensional table wherein row is `Locale`, column contains code values and cell details are labels.

The table would be as follows in case of a selectbox for selecting charges.

row=Locale, column=Code	0	10000	20000	30000	40000	50000
en	unlimited	Less than \10,000	Less than \20,000	Less than \30,000	Less than \40,000	Less than \50,000
ja	上限なし	10,000 円以下	20,000 円以下	30,000 円以下	40,000 円以下	50,000 円以下

For creating a codelist table that supports internationalization, `SimpleI18nCodeList` has been set in following 3 ways.

- Set `CodeList` for each locale by rows.
- Set `java.util.Map`(key = code value, value = label) for each locale by rows.
- Set `java.util.Map`(key = locale, value = label) for each code value by columns.

It is recommended that you set the codelist using “Set `CodeList` for each locale by rows.” method.

The way of setting the `CodeList` for each locale by rows considering the above example of selectbox for selecting charges, is mentioned below. For other setting methods, refer to *Setting SimpleI18nCodeList*.

Definition of Bean definition file (xxx-codelist.xml)

```
<bean id="CL_I18N_PRICE"
    class="org.terasoluna.gfw.common.codelist.i18n.SimpleI18nCodeList">
    <property name="rowsByCodeList"> <!-- (1) -->
        <util:map>
            <entry key="en" value-ref="CL_PRICE_EN" />
            <entry key="ja" value-ref="CL_PRICE_JA" />
        </util:map>
    </property>
</bean>
```

Sr. No.	Description
(1)	Set Map wherein key is `java.lang.Locale`, in rowsByCodeList properties. In Map, specify locale in key and a reference link to codelist class corresponding to locale in value-ref. For Map values, refer to codelist class corresponding to each locale.

Definition of Bean definition file(xxx-codelist.xml) when creating SimpleMapCodeList for each locale

```
<bean id="CL_I18N_PRICE"
    class="org.terasoluna.gfw.common.codelist.i18n.SimpleI18nCodeList">
    <property name="rowsByCodeList">
        <util:map>
            <entry key="en" value-ref="CL_PRICE_EN" />
            <entry key="ja" value-ref="CL_PRICE_JA" />
        </util:map>
    </property>
</bean>

<bean id="CL_PRICE_EN" class="org.terasoluna.gfw.common.codelist.SimpleMapCodeList
↪"> <!-- (2) -->
    <property name="map">
```

```
        <util:map>
            <entry key="0" value="unlimited" />
            <entry key="10000" value="Less than \\10,000" />
            <entry key="20000" value="Less than \\20,000" />
            <entry key="30000" value="Less than \\30,000" />
            <entry key="40000" value="Less than \\40,000" />
            <entry key="50000" value="Less than \\50,000" />
        </util:map>
    </property>
</bean>

<bean id="CL_PRICE_JA" class="org.terasoluna.gfw.common.codelist.SimpleMapCodeList
↪">  <!-- (3) -->
    <property name="map">
        <util:map>
            <entry key="0" value="上限なし" />
            <entry key="10000" value="10,000円以下" />
            <entry key="20000" value="20,000円以下" />
            <entry key="30000" value="30,000円以下" />
            <entry key="40000" value="40,000円以下" />
            <entry key="50000" value="50,000円以下" />
        </util:map>
    </property>
</bean>
```

Sr. No.	Description
(2)	For bean definition `CL_PRICE_EN` where locale is “en”, codelist class is set in `SimpleMapCodeList`.
(3)	For bean definition `CL_PRICE_JA` where locale is “ja”, codelist class is set in `SimpleMapCodeList`.

Definition of Bean definition file(xxx-codelist.xml) when creating JdbcCodeList for each locale

```
<bean id="CL_I18N_PRICE"
    class="org.terasoluna.gfw.common.codelist.i18n.SimpleI18nCodeList">
    <property name="rowsByCodeList">
        <util:map>
            <entry key="en" value-ref="CL_PRICE_EN" />
            <entry key="ja" value-ref="CL_PRICE_JA" />
```

```
        </util:map>
    </property>
</bean>

<bean id="CL_PRICE_EN" parent="AbstractJdbcCodeList">  <!-- (4) -->
    <property name="querySql"
        value="SELECT code, label FROM price WHERE locale = 'en' ORDER BY code" />
    <property name="valueColumn" value="code" />
    <property name="labelColumn" value="label" />
</bean>

<bean id="CL_PRICE_JA" parent="AbstractJdbcCodeList">  <!-- (5) -->
    <property name="querySql"
        value="SELECT code, label FROM price WHERE locale = 'ja' ORDER BY code" />
    <property name="valueColumn" value="code" />
    <property name="labelColumn" value="label" />
</bean>
```

Sr. No.	Description
(4)	For bean definition `CL_PRICE_EN` where locale is “en”, codelist class is set in `JdbcCodeList`.
(5)	For bean definition `CL_PRICE_JA` where locale is “ja”, codelist class is set in `JdbcCodeList`.

Insert the following data in Table Definition (price table).

locale	code	label
en	0	unlimited
en	10000	Less than \10,000
en	20000	Less than \20,000
en	30000	Less than \30,000

次のページに続く

Table. 4.46 – 前のページからの続き

locale	code	label
en	40000	Less than \40,000
en	50000	Less than \50,000
ja	0	上限なし
ja	10000	10,000 円以下
ja	20000	20,000 円以下
ja	30000	30,000 円以下
ja	40000	40,000 円以下
ja	50000	50,000 円以下

Warning: Currently `SimpleI18nCodeList` does not support reloadable functionality. It should be noted that even if `JdbcCodeList` (reloadable CodeList) referred by `SimpleI18nCodeList` is reloaded, it does not get reflected in `SimpleI18nCodeList`. In order to make it reloadable, it should be implemented independently. For implementation method, refer to *Customizing the codelist independently*.

Using codelist in JSP

Description of basic settings is omitted since it is same as *Using codelist in JSP* described earlier.

Definition of Bean definition file(spring-mvc.xml)

```
<mvc:interceptors>
  <mvc:interceptor>
    <mvc:mapping path="/**" />
    <bean
      class="org.terasoluna.gfw.web.codelist.CodeListInterceptor">
      <property name="codeListIdPattern" value="CL_.+" />
      <property name="fallbackTo" value="en" />  <!-- (1) -->
    </bean>
  </mvc:interceptor>

  <!-- omitted -->

</mvc:interceptors>
```

Sr. No.	Description
(1)	When request locale is not defined in codelist, codelist is fetched using the locale set in fallbackTo property. When fallbackTo property is not set, default JVM locale is used as fallbackTo property. When codelist cannot be fetched even after using the locale set in fallbackTo property, WARN log is output and empty Map is returned.

Example of jsp implementation

```
<form:select path="basePrice" items="${CL_I18N_PRICE}" />
```

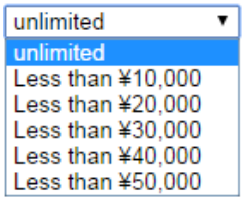
Output HTML lang=en

```
<select id="basePrice" name="basePrice">
  <option value="0">unlimited</option>
  <option value="1">Less than \\10,000</option>
  <option value="2">Less than \\20,000</option>
  <option value="3">Less than \\30,000</option>
  <option value="4">Less than \\40,000</option>
  <option value="5">Less than \\50,000</option>
</select>
```

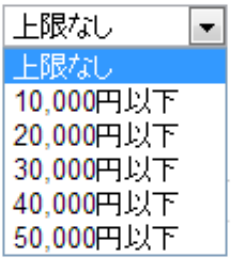
Output HTML lang=ja

```
<select id="basePrice" name="basePrice">
  <option value="0">上限なし</option>
  <option value="1">10,000 円以下</option>
  <option value="2">20,000 円以下</option>
  <option value="3">30,000 円以下</option>
  <option value="4">40,000 円以下</option>
  <option value="5">50,000 円以下</option>
</select>
```

Output screen lang=en



Output screen lang=ja



Using codelist in Java class

Description of basic settings is omitted since it is same as *Using codelist in Java class* described earlier.

```
@RequestMapping("orders")
@Controller
public class OrderController {

    @Inject
    @Named("CL_I18N_PRICE")
    I18nCodeList priceCodeList;

    // ...

    @RequestMapping(method = RequestMethod.POST, params = "confirm")
    public String confirm(OrderForm form, Locale locale) {
        // ...
        String priceMassage = getPriceMessage(form.getPriceCode(), locale);
        // ...
    }

    private String getPriceMessage(String targetPrice, Locale locale) {
         return priceCodeList.asMap(locale).get(targetPrice);  // (1)
    }

}
```

Sr. No.	Description
(1)	Map of locale corresponding to I18nCodeList#asMap(Locale) can be fetched.

Display code name corresponding to code value

When it is necessary to refer the codelist in JSP, it can be referred same as `java.util.Map` interface.

For details, see the below example.

Example of jsp implementation

```
Order Status : ${f:h(CL_ORDERSTATUS[orderForm.orderStatus])}
```

Sr. No.	Description
(1)	Get a codelist that has been converted to the `java.util.Map` from the request scope (In this example, `"CL_ORDERSTATUS"` used as codelist). The codelist has been referred with the beanID of codelist. Then specify a code value as a key of the `Map` interface which displays a corresponding code name (In this example, `orderStatus` value is used as a key).

Input validation of code value using codelist

When checking whether the input value is the code value defined in codelist, `org.terasoluna.gfw.common.codelist.ExistInCodeList` annotation for BeanValidation is provided in common library.

For details on BeanValidation and message output method, refer to *Input Validation*.

For input validation using `@ExistInCodeList` annotation, it is necessary to carry out "*Definition of error messages*" for `@ExistInCodeList` .

When project is created by Blank project , the following message is defined in `ValidationMessages.properties` file directly under `xxx-web/src/main/resources`. Please change the message to fit the application requirements.

```
org.terasoluna.gfw.common.codelist.ExistInCodeList.message = Does not exist in
↪{codeListId}
```

Note: In the terasoluna-gfw-common 5.0.0.RELEASE or later, the format of message property key has been changed to standard format of Bean Validation (FQCN of annotation + `.message`).

Version	Property key of message
version 5.0.0.RELEASE or later	`org.terasoluna.gfw.common.codelist.ExistInCodeList.m`
version 1.0.x.RELEASE	`org.terasoluna.gfw.common.codelist.ExistInCodeList`

For migrating to the version 5.0.0.RELEASE or later from the version 1.0.x.RELEASE, when message is changed to fit the application requirements, the property key should be changed.

Note: From terasoluna-gfw-common 1.0.2.RELEASE, `ValidationMessages.properties`wherein `@ExistInCodeList`message is defined, is not included in jar file. This is to fix the "Bug in which message is not displayed if multiple ValidationMessages.properties exist".

For migrating to version 1.0.2.RELEASE or later from version 1.0.1.RELEASE or prior, if the message defined in `ValidationMessages.properties`included in jar of terasoluna-gfw-common, is used, it is necessary to define the message by creating `ValidationMessages.properties`.

Example of @ExistInCodeList settings

See below the example of input validation method using codelist.

Definition of Bean definition file(xxx-codelist.xml)

```
<bean id="CL_GENDER" class="org.terasoluna.gfw.common.codelist.SimpleMapCodeList">
    <property name="map">
        <map>
            <entry key="M" value="Male" />
            <entry key="F" value="Female" />
        </map>
    </property>
</bean>
```

Form object

```
public class Person {
    @ExistInCodeList(codeListId = "CL_GENDER")  // (1)
    private String gender;

    // getter and setter omitted
}
```

Sr. No.	Description
(1)	Set `@ExistInCodeList` annotation for the field for which input is to be validated, and specify the target codelist in codeListId.

As a result of above settings, when characters other than M, F are stored in `gender`, the system throws an error.

Tip: `@ExistInCodeList` input validation supports only the implementation class (`String` etc) of `CharSequence` interface or `Character` type. Therefore, even if the fields with `@ExistInCodeList`may contain integer values, they should be defined as `String`data type. (such as Year/Month/Day)

Also, because `@ExistInCodeList`does not correspond to the values in the collection,it is necessary to device the implementation in order to correspond `@ExistInCodeList`annotation to multiple selectable screen items (check boxes, multiple selection drop down, etc.) For details, please refer *How to check the values in the collection using Bean Validation's annotation* .

4.8.3 How to extend

When reloading the codelist

Codelist provided in common library is read at the time of launching the application and it is never updated subsequently. However, in some cases, when the master data of the codelist is updated, the codelist also needs to be updated.

Example: Updating the codelist when DB master is updated using JdbcCodeList.

Common library provides `org.terasoluna.gfw.common.codelist.ReloadableCodeList` interface. The class implementing the above interface, implements refresh method. Codelist can be updated by calling this refresh method. JdbcCodeList implements ReloadableCodeList interface; hence it is possible to update the codelist.

Codelist can be updated in following two ways.

1. By using Task Scheduler
2. By calling refresh method in Controller (Service) class

This guideline recommends the method to reload the codelist periodically using Spring Task Scheduler.

However, when it is necessary to arbitrarily refresh the codelist, it is appropriate to call refresh method in Controller class.

Note: For the codelist having ReloadableCodeList interface, refer to *List of codelist types*.

Using Task Scheduler

Example for setting the Task Scheduler is shown below.

Definition of Bean definition file(xxx-codelist.xml)

```
<task:scheduler id="taskScheduler" pool-size="10"/>  <!-- (1) -->

<task:scheduled-tasks scheduler="taskScheduler">  <!-- (2) -->
    <task:scheduled ref="CL_AUTHORITIES" method="refresh" cron="${cron.codelist.
↪refreshTime}"/>  <!-- (3) -->
</task:scheduled-tasks>

<bean id="CL_AUTHORITIES" parent="AbstractJdbcCodeList">
    <property name="querySql"
        value="SELECT authority_id, authority_name FROM authority ORDER BY␣
↪authority_id" />
    <property name="valueColumn" value="authority_id" />
    <property name="labelColumn" value="authority_name" />
</bean>
```

Sr. No.	Description
(1)	Specify the thread pool size in pool-size attribute of `<task:scheduler>` element. When pool-size attribute is not specified, the value is set to "1".
(2)	Define `<task:scheduled-tasks>` element and set `<task:scheduler>` ID in scheduler attribute.
(3)	Define `<task:scheduled>` element. Specify refresh method in method attribute. In cron attribute, the value should be mentioned in `org.springframework.scheduling.support.CronSequenceGenerator` supported format. Reload timing for cron attribute may change with development environment and commercial environment; hence it is recommended to fetch the codelist from property file or environment variable. **Example of setting cron attribute** Specify in "Seconds Minutes Hours Month Year Day". execution every second "* * * * * *" execution every hour "0 0 * * * *" execution every hour 9:00-17:00 on weekdays "0 0 9-17 * * MON-FRI" For details, refer to JavaDoc. http://docs.spring.io/spring/docs/4.3.5.RELEASE/javadoc-api/org/springframework/scheduling/support/CronSequenceGenerator.html

Calling refresh method in Controller (Service) class

See the example below for directly calling refresh method of JdbcCodeList in Service class.

Definition of Bean definition file(xxx-codelist.xml)

```
<bean id="CL_AUTHORITIES" parent="AbstractJdbcCodeList">
    <property name="querySql"
```

```
        value="SELECT authority_id, authority_name FROM authority ORDER BY␣
↪authority_id" />
    <property name="valueColumn" value="authority_id" />
    <property name="labelColumn" value="authority_name" />
</bean>
```

Controller class

```
@Controller
@RequestMapping(value = "codelist")
public class CodeListContoller {

    @Inject
    CodeListService codeListService; // (1)

    @RequestMapping(method = RequestMethod.GET, params = "refresh")
    public String refreshJdbcCodeList() {
        codeListService.refresh(); // (2)
        return "codelist/jdbcCodeList";
    }
}
```

Sr. No.	Description
(1)	Inject the Service class that executes refresh method of ReloadableCodeList class.
(2)	Execute the refresh method of Service class that executes refresh method of ReloadableCodeList class.

Service class

The description below is given only for the implementation class. Description for interface class has been omitted.

```
@Service
public class CodeListServiceImpl implements CodeListService { // (3)

    @Inject
    @Named(value = "CL_AUTHORITIES") // (4)
    ReloadableCodeList codeListItem; // (5)

    @Override
    public void refresh() { // (6)
        codeListItem.refresh(); // (7)
    }
}
```

Sr. No.	Description
(3)	Implement `CodeListService` interface for `CodeListServiceImpl` class.
(4)	Specify the corresponding codelist using `@Named` annotation at the time of injecting the codelist. ID of the bean to be fetched should be specified in `value` attribute. Codelist of ID attribute "CL_AUTHORITIES" of bean tag defined in Bean definition file is injected.
(5)	ReloadableCodeList interface should be defined in field type. ReloadableCodeList interface should be implemented for Bean specified in (4).
(6)	refresh method defined in Service class is called from Controller class.
(7)	refresh method of codelist wherein ReloadableCodeList interface is implemented. Codelist is updated by executing refresh method.

Customizing the codelist independently

In order to create a codelist which does not fall under the 4 types provided by the common library, the existing codelist can be customized independently. Refer to the table below for the implementation method and type of codelist that can be created.

Sr. No.	Reloadable	Class to be inherited	Implementation location
(1)	Not required	`org.terasoluna.gfw.common.codelist.AbstractCodeList`	Override `asMap`
(2)	Required	`org.terasoluna.gfw.common.codelist.AbstractReloadableCodeList`	Override `retrieveMap`

The codelist can be customized by directly implementing `org.terasoluna.gfw.common.codelist.CodeList` and `org.terasoluna.gfw.common.codelist.ReloadableCodeList` interfaces; however extending the abstract class provided in common library minimizes the implementation efforts.

Actual example of independent customization is shown below. It illustrates a codelist for creating a list of current year and the next year. (Example: If current year is 2013, it is stored in codelist in the order of "2013, 2014".)

Codelist class

```
@Component("CL_YEAR") // (1)
public class DepYearCodeList extends AbstractCodeList { // (2)

    @Inject
    JodaTimeDateFactory dateFactory; // (3)

    @Override
    public Map<String, String> asMap() {  // (4)
        DateTime dateTime = dateFactory.newDateTime();
        DateTime nextYearDateTime = dateTime.plusYears(1);

        Map<String, String> depYearMap = new LinkedHashMap<String, String>();

        String thisYear = dateTime.toString("Y");
        String nextYear = nextYearDateTime.toString("Y");
        depYearMap.put(thisYear, thisYear);
        depYearMap.put(nextYear, nextYear);

        return Collections.unmodifiableMap(depYearMap);
    }
}
```

Sr. No.	Description
(1)	Register the codelist as a component using `@Component` annotation. By specifying `"CL_YEAR"` in Value, register the codelist as a component using the codelist intercept set in bean definition.
(2)	Inherit `org.terasoluna.gfw.common.codelist.AbstractCodeList`. When creating the list of current year and next year, reloading is not necessary since it is created dynamically by calculating from system date.
(3)	`org.terasoluna.gfw.common.date.jodatime.JodaTimeDateFactory` creating the Date class of system date is injected. Current year and next year can be fetched using `JodaTimeDateFactory`. Class that implements `JodaTimeDateFactory` interface should be set in advance in bean definition file.
(4)	Override `asMap()` method and create the list of current year and next year. Implementation differs with every created codelist.

Example of jsp implementation

```
<form:select path="mostRecentYear" items="${CL_YEAR}" /> <!-- (5) -->
```

Sr. No.	Description
(5)	`"CL_YEAR"` registered as component in items attribute should be specified in `${}` placeholder to fetch the corresponding codelist.

Output HTML

```
<select id="mostRecentYear" name="mostRecentYear">
   <option value="2013">2013</option>
```

```
   <option value="2014">2014</option>
</select>
```

Output screen



Note: Implementation should be made thread-safe at the time of customizing the reloadable CodeList independently.

4.8.4 Appendix

Setting SimpleI18nCodeList

Apart from the settings mentioned in *How to use SimpleI18nCodeList*, SimpleI18nCodeList can be set in following 2 ways. The respective setting methods are explained using the example of selectbox for selecting charges.

Set `java.util.Map` (key = code value, value = label) for each locale by rows

Definition of Bean definition file(xxx-codelist.xml)

```
<bean id="CL_I18N_PRICE"
    class="org.terasoluna.gfw.common.codelist.i18n.SimpleI18nCodeList">
    <property name="rows"> <!-- (1) -->
        <util:map>
            <entry key="en">
                <util:map>
                    <entry key="0" value="unlimited" />
                    <entry key="10000" value="Less than \\10,000" />
                    <entry key="20000" value="Less than \\20,000" />
                    <entry key="30000" value="Less than \\30,000" />
                    <entry key="40000" value="Less than \\40,000" />
                    <entry key="50000" value="Less than \\50,000" />
                </util:map>
            </entry>
            <entry key="ja">
                <util:map>
```

```
                    <entry key="0" value="unlimited" />
                    <entry key="10000" value="10,000 円以下" />
                    <entry key="20000" value="20,000 円以下" />
                    <entry key="30000" value="30,000 円以下" />
                    <entry key="40000" value="40,000 円以下" />
                    <entry key="50000" value="50,000 円以下" />
                </util:map>
            </entry>
        </util:map>
    </property>
</bean>
```

Sr. No.	Description
(1)	Set “Map of Map” for rows property. External Map key is `java.lang.Locale`. Internal Map key is a code value and value is a label corresponding to locale.

Set `java.util.Map`(key = locale, value = label) for each code value by columns

Definition of Bean definition file(xxx-codelist.xml)

```
<bean id="CL_I18N_PRICE"
    class="org.terasoluna.gfw.common.codelist.i18n.SimpleI18nCodeList">
    <property name="columns"> <!-- (1) -->
        <util:map>
            <entry key="0">
                <util:map>
                    <entry key="en" value="unlimited" />
                    <entry key="ja" value="上限なし" />
                </util:map>
            </entry>
            <entry key="10000">
                <util:map>
                    <entry key="en" value="Less than \\10,000" />
                    <entry key="ja" value="10,000 円以下" />
                </util:map>
            </entry>
            <entry key="20000">
                <util:map>
                    <entry key="en" value="Less than \\20,000" />
                    <entry key="ja" value="20,000 円以下" />
                </util:map>
```

```
            </entry>
            <entry key="30000">
                <util:map>
                    <entry key="en" value="Less than \\30,000" />
                    <entry key="ja" value="30,000円以下" />
                </util:map>
            </entry>
            <entry key="40000">
                <util:map>
                    <entry key="en" value="Less than \\40,000" />
                    <entry key="ja" value="40,000円以下" />
                </util:map>
            </entry>
            <entry key="50000">
                <util:map>
                    <entry key="en" value="Less than \\50,000" />
                    <entry key="ja" value="50,000円以下" />
                </util:map>
            </entry>
        </util:map>
    </property>
</bean>
```

Sr. No.	Description
(1)	Set “Map of Map” for columns property. External Map key is a code value. Internal Map key is `java.lang.Locale` and value is a label corresponding to locale.

Variations of NumberRangeCodeList

Create the Descending NumberRangeCodeList

Example of setting To value < From value is shown below.

Definition of Bean definition file(xxx-codelist.xml)

```
<bean id="CL_BIRTH_YEAR"
    class="org.terasoluna.gfw.common.codelist.NumberRangeCodeList">
    <property name="from" value="2013" /> <!-- (1) -->
    <property name="to" value="2000" /> <!-- (2) -->
</bean>
```

Sr. No.	Description
(1)	Specify the range start value. Specify a value greater than the one specified in “value” attribute of “to” property. As per this specification, display the values decreased in accordance with the interval in To-From range in descending order. Since interval is not set, default value 1 is applied.
(2)	Specify the range end value. In this example, since 2000 is specified as range end value; the value is reduced by 1 and stored in descending order from 2013 to 2000.

Example of jsp implementation

```
<form:select path="birthYear" items="${CL_BIRTH_YEAR}" />
```

Output HTML

```
<select id="birthYear" name="birthYear">
  <option value="2013">2013</option>
  <option value="2012">2012</option>
  <option value="2011">2011</option>
  <option value="2010">2010</option>
  <option value="2009">2009</option>
  <option value="2008">2008</option>
  <option value="2007">2007</option>
  <option value="2006">2006</option>
  <option value="2005">2005</option>
  <option value="2004">2004</option>
  <option value="2003">2003</option>
  <option value="2002">2002</option>
  <option value="2001">2001</option>
  <option value="2000">2000</option>
</select>
```

Output screen



Change interval of NumberRangeCodeList

Example of setting interval value is shown below.

Definition of Bean definition file(xxx-codelist.xml)

```
<bean id="CL_BULK_ORDER_QUANTITY_UNIT"
    class="org.terasoluna.gfw.common.codelist.NumberRangeCodeList">
    <property name="from" value="10" />
    <property name="to" value="50" />
    <property name="interval" value="10" /> <!-- (1) -->
</bean>
```

Sr. No.	Description
(1)	Specify increment (decrement) value. Then, store the values obtained upon increasing (decreasing) the interval value within From-To range as codelist. In the above example, the values are stored in the order of `10`,`20`,`30`,`40`,`50` in the codelist.

Example of jsp implementation

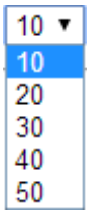
```
<form:select path="quantity" items="${CL_BULK_ORDER_QUANTITY_UNIT}" />
```

Output HTML

```
<select id="quantity" name="quantity">
    <option value="10">10</option>
    <option value="20">20</option>
```

```
    <option value="30">30</option>
    <option value="40">40</option>
    <option value="50">50</option>
</select>
```

Output screen



Note: If From-To value exceeds the specified range, then the value increased (decreased) in accordance with interval is not stored in the codelist.

i.e. in case of following definition,

```
<bean id="CL_BULK_ORDER_QUANTITY_UNIT"
    class="org.terasoluna.gfw.common.codelist.NumberRangeCodeList">
    <property name="from" value="10" />
    <property name="to" value="55" />
    <property name="interval" value="10" />
</bean>
```

5 values of `10`,`20`,`30`,`40`,`50` are stored in the codelist. The value of subsequent interval `60` and the range threshold value `55` are not stored in the codelist.

4.9 File Upload

4.9.1 Overview

This chapter explains how to upload files.

Files are uploaded using the File Upload functionality supported by Servlet 3.0 and classes provided by Spring Web.

Note: In this chapter, File Upload functionality supported by Servlet 3.0 is used; hence, Servlet version 3.0 or above is a prerequisite here.

Note: File Upload functionality of Servlet 3.0 may likely result into garbling of multi byte characters of file names or request parameters on some application server.

Application servers wherein occurrence of issues is confirmed at the version 5.3.0.RELEASE are as shown below.

- WebLogic 12.1.3
- JBoss EAP 7.0
- JBoss EAP 6.4.0.GA

Among these, the issues can be avoided by adding application server specific settings in JBoss EAP 7.0. For details, refer Precautions for using JBoss EAP 7.0.

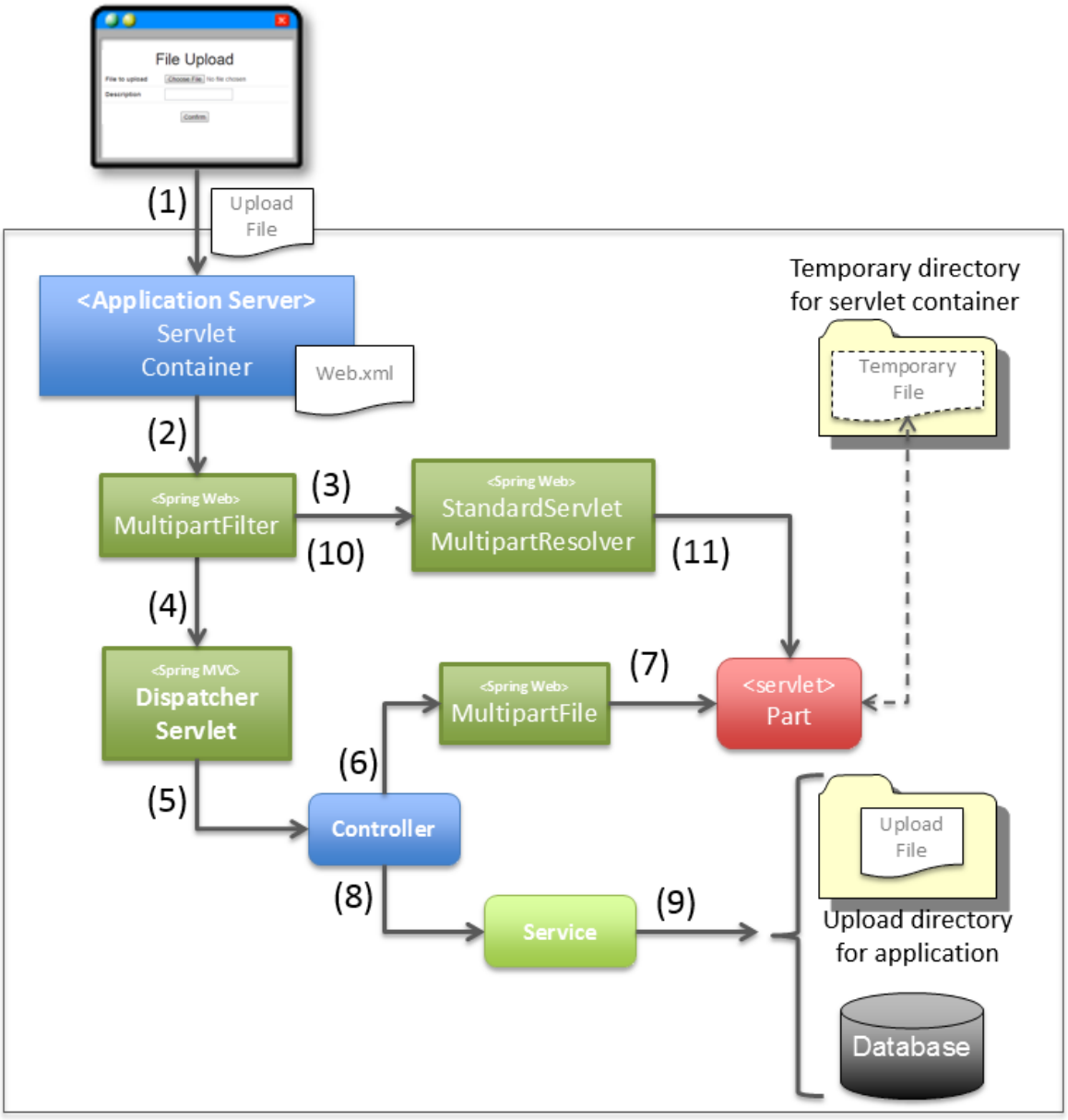
When the application server wherein other problems occur are used, the issues can be avoided by using Commons FileUpload. For setup methods for using Commons FileUpload, refer "*File upload using Commons FileUpload*".

Warning: If implementation of file upload of an application server to be used depends on implementation of Apache Commons FileUpload, security vulnerabilities reported in CVE-2014-0050 and CVE-2016-3092 may occur. Hence ensure that there are no such vulnerabilities in the application server to be used.

In case of using Tomcat, it is necessary to use version 7.0.52 or above for series 7.0, and version 8.0.3 or above for series 8.0.

Basic flow of upload process

Basic flow of uploading files using File Upload functionality supported by Servlet 3.0, and classes of Spring Web, is as shown below.



Sr. No.	Description
(1)	Select and upload the target files.

次のページに続く

Table. 4.47 – 前のページからの続き

Sr. No.	Description
(2)	Servlet container receives `multipart/form-data` request and calls `org.springframework.web.multipart.support.MultipartFilter`.
(3)	`MultipartFilter` calls the method of `org.springframework.web.multipart.support.StandardServletMultipartResolver` to enable File Upload functionality of Servlet 3.0 in Spring MVC. `StandardServletMultipartResolver` generates `org.springframework.web.multipart.MultipartFile` object that wraps the API (`javax.servlet.http.Part`) introduced through Servlet 3.0.
(4)	Apply a filter chain in `DispatcherServlet` from `MultipartFilter`.
(5)	`DispatcherServlet` calls handler method of Controller. `MultipartFile` object generated in (3) is bound to Controller argument or form object.
(6)	Controller calls a method of `MultipartFile` object and fetch contents of uploaded file and meta information (file name etc.).
(7)	`MultipartFile` calls a method of `Part` object introduced from Servlet 3.0, fetches contents of uploaded file and meta information (file name etc.) and returns to Controller.
(8)	Controller calls the Service method and executes upload process. It passes the contents and meta information (file name etc.) of the file retrieved from `MultipartFile` object as an argument of Service method.
(9)	Service stores contents of uploaded file and meta information (file name etc.) in the file or database.

次のページに続く

Table. 4.47 – 前のページからの続き

Sr. No.	Description
(10)	`MultipartFilter` calls `StandardServletMultipartResolver` and deletes temporary file used by file upload function of Servlet 3.0.
(11)	`StandardServletMultipartResolver` calls a method of `Part` object introducted from Servlet 3.0 and deletes the temporary file stored in the disc.

Note: Controller performs the process for `MultipartFile` object of Spring Web; hence implementation which is dependent on the File Upload API provided by Servlet 3.0 can be excluded.

About classes provided by Spring Web

Classes provided by Spring Web for uploading a file are as follows:

Sr. No.	Class name	Description
1.	org.springframework.web.multipart. MultipartFile	Interface indicating uploaded file. It plays a role in abstraction of file objects handled by the File Upload functionality to be used.
2.	org.springframework.web.multipart.support. StandardMultipartHttpServletRequest$ StandardMultipartFile	`MultipartFile` class of File Upload functionality introduced through Servlet 3.0. Process is delegated to the `Part` object introduced through Servlet 3.0.
3.	org.springframework.web.multipart. MultipartResolver	Interface that resolves the analysis method of `multipart/form-data` request. It plays a role in generating `MultipartFile` object corresponding to implementation of File Upload functionality.
4.	org.springframework.web.multipart.support. StandardServletMultipartResolver	`MultipartResolver` class for File Upload functionality introduced through Servlet 3.0.
5.	org.springframework.web.multipart.support. MultipartFilter	A class which generates MultipartFile by calling a class which implements MultipartResolver from DI container, at the time of multipart/form-data request. If this class is not used, a request parameter cannot be fetched in Servlet Filter process when maximum size allowed in file upload exceeds the limit. Therefore, it is recommended to use MultipartFilter in this guideline.

Tip: In this guideline, it is a prerequisite to use File Upload functionality implemented from Servlet 3.0. However, Spring Web also provides an implementation class for "Apache Commons FileUpload".

The difference in implementation of upload processes is absorbed by `MultipartResolver` and `MultipartFile` objects; hence it does not affect Controller implementation.

4.9.2 How to use

Application settings

Settings to enable Servlet 3.0 upload functionality

Perform the following settings to enable upload functionality of Servlet 3.0.

- `web.xml`

```
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.
↪com/xml/ns/javaee/web-app_3_0.xsd"
    version="3.0"> <!-- (1) (2) -->

    <servlet>
        <servlet-class>
            org.springframework.web.servlet.DispatcherServlet
        </servlet-class>
        <!-- omitted -->
        <multipart-config> <!-- (3) -->
            <max-file-size>5242880</max-file-size> <!-- (4) -->
            <max-request-size>27262976</max-request-size> <!-- (5) -->
             <file-size-threshold>0</file-size-threshold> <!-- (6) -->
        </multipart-config>
    </servlet>

    <!-- omitted -->

</web-app>
```

Sr. No.	Description
(1)	Specify the XSD file of Servlet 3.0 or above in `xsi:schemaLocation` attribute of `<web-app>` element.

次のページに続く

Table. 4.48 – 前のページからの続き

Sr. No.	Description
(2)	Specify version `3.0` or above in the `version` attribute of `<web-app>` element.
(3)	Add `<multipart-config>` element to `<servlet>` element of the Servlet using the File Upload functionality.
(4)	Specify the maximum file size of 1 upload-permissible file in bytes. If not specified, -1 (no limit) is set by default. If it exceeds the specified value, exception, `org.springframework.web.multipart.MultipartException` occurs. In the above example, a file size of 5MB is specified.
(5)	Specify the maximum Content-Length value of `multipart/form-data` request. If not specified, -1 (no limit) is set by default. If it exceeds the specified value, exception `org.springframework.web.multipart.MultipartException` occurs. Value to be set in this parameter should be calculated by the following formula. **(“maximum file size of 1 file to be uploaded” * “Number of files allowed to be uploaded simultaneously”) + “Data size of other form fields” + “Meta information size of multipart/form-data request”** In the above example, parameter value of 26MB is specified. Its breakup is, 25MB (5MB * 5 files) and 1MB (number of bytes of meta information + number of bytes of form fields).

次のページに続く

Table. 4.48 – 前のページからの続き

Sr. No.	Description
(6)	Specify the threshold value (number of bytes for 1 file) if the contents of uploaded file are to be saved as a temporary file. If this parameter is not specified explicitly, there are application servers wherein values specified for elements `<max-file-size>` and `<max-request-size>` are considered invalid; hence default value (0) is being specified explicitly.

Warning: In order to increase the resistance against Dos attack, `max-file-size` and `max-request-size` should be specified without fail.

For Dos attack, refer to *Dos attack with respect to upload functionality*.

Note:

Uploaded file is by default output as temporary file. However, its output can be controlled using the configuration value of `<file-size-threshold>` element, which is the child element of `<multipart-config>`.

```
<!-- omitted -->

<multipart-config>
    <!-- omitted -->
    <file-size-threshold>32768</file-size-threshold> <!-- (7) -->
</multipart-config>

<!-- omitted -->
```

Sr. No.	Description
(7)	Specify the threshold file size (number of bytes of 1 file) if contents of uploaded file are to be saved as a temporary file. If not specified, 0 is set. If uploaded file size exceeds the specified value, it is output as a temporary file to the disk and deleted when the request is completed. In the above example, 32KB is specified.

Warning: This parameter shows a trade-off relationship as indicated by the following points. Hence, **configuration value corresponding to system characteristics should be specified.**.

- Increasing the configuration value improves processing performance as, processing gets completed within available memory. However, there is a high possibility that `OutOfMemoryError` may occur due to Dos attack.
- If configuration value is reduced, memory utilization can be controlled to the minimum, thereby avoiding the possibility of `OutOfMemoryError` due to Dos attack

> etc. However, there is a high possibility of performance degradation since the frequency of disk IO generation is high.

To change output directory of temporary files, specify directory path in `<location>` element, which is the child element of `<multipart-config>`.

```
<!-- omitted -->

<multipart-config>
    <location>/tmp</location> <!-- (8) -->
    <!-- omitted -->
</multipart-config>

<!-- omitted -->
```

Sr. No.	Description
(8)	Specify the directory path for outputting temporary files. When omitted, they are output to the directory that stores temporary files of application server. In the above example, `/tmp` is specified.

> **Warning:** The directory specified in `<location>` element is the one used by the application server (servlet container) and **cannot be accessed from application.**
>
> When the files uploaded as application are to be saved as temporary files, they should be output to a directory other than the directory specified in `<location>` element.

Servlet Filter settings

The operation when the maximum size allowed in file upload exceeds the limit at the time of multipart/form-data request, varies depending on the application server. `MultipartException` generated when maximum size exceeds the limit depending on the application server is likely to be not detected and exception handling settings described later will be invalid.

Since this operation can be evaded by setting `MiltipartFilter`, `MiltipartFilter` setting is described

as a prerequisite in this guideline.

Setting example is given below.

- `web.xml`

```
<!-- (1) -->
<filter>
    <filter-name>MultipartFilter</filter-name>
    <filter-class>org.springframework.web.multipart.support.MultipartFilter</
↪filter-class>
</filter>
<!-- (2) -->
<filter-mapping>
    <filter-name>MultipartFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

Sr. No.	Description
(1)	Define `MultipartFilter` as the Servlet Filter.
(2)	Specify the URL pattern for applying `MultipartFilter`.

Warning: Precautions while using Spring Security

When security countermeasures are to be carried out by using Spring Security, they should be defined prior to `springSecurityFilterChain`. Further, when request parameters are accessed by a project-specific Servlet Filter, MultipartFilter should be defined before that Servlet Filter.

However, when defined before `springSecurityFilterChain`, unauthenticated or unauthorized users may be allowed to upload the file (create temporary file). Although a method to avoid this operation has been given in Spring Security Reference -Cross Site Request Forgery (CSRF)-, it is not recommended to be applied in this guideline since it poses a security risk.

Warning: Precautions when maximum size limit for file upload is exceeded

When allowable size limit for file upload has been exceeded, an 'Over the size limit" error may get detected before fetching a CSRF token in some of the application servers like WebLogic and CSRF token check is not performed.

Note: Default calling of MultipartResolver

If `MultipartFilter` is used, `org.springframework.web.multipart.support.StandardServletMult` is called by default. `StandardServletMultipartResolver` should be able to generates uploaded file as `org.springframework.web.multipart.MultipartFile` and receive as property of Controller argument and form object.

Settings for exception handling

Add the exception handling definition of `MultipartException` which occurs when a request for file or multipart with non-permissible size is sent.

`MultipartException` is an exception caused due to file size specified by the client; hence it is recommended to handle it as a client error (HTTP response code=4xx).

If exception handling is not added for individual exception, it is eventually treated as system error; hence make sure that it is defined without fail.

Settings for handling `MultipartException` differ depending upon whether `MultipartFilter` is used or not.

In case of using `MultipartFilter`, exception handling is carried out by using the `<error-page>` functionality of servlet container.

Example of settings is shown below.

- `web.xml`

```
<error-page>
    <!-- (1) -->
    <exception-type>org.springframework.web.multipart.MultipartException</
↪exception-type>
    <!-- (2) -->
    <location>/WEB-INF/views/common/error/fileUploadError.jsp</location>
</error-page>
```

Sr. No.	Description
(1)	Specify `MultipartException` as the exception class for handling.
(2)	Specify the file displayed when `MultipartException` occurs. In the above example, `"/WEB-INF/views/common/error/fileUploadError.jsp"` is specified.

- `fileUploadError.jsp`

```
<%-- (3) --%>
<% response.setStatus(HttpServletResponse.SC_BAD_REQUEST); %>
<!DOCTYPE html>
<html>

    <!-- omitted -->

</html>
```

Sr. No.	Description
(3)	Set HTTP status code by calling the API of `HttpServletResponse`. In the above request, `"400"` (Bad Request) is set. When not set explicitly, the HTTP status code is considered as `"500"` (Internal Server Error).

When not using `MultipartFilter`, carry out exception handling by using `SystemExceptionResolver`.
Example of settings is shown below.

- `spring-mvc.xml`

```
<bean class="org.terasoluna.gfw.web.exception.SystemExceptionResolver">
    <!-- omitted -->
    <property name="exceptionMappings">
        <map>
```

```
            <!-- omitted -->
            <!-- (4) -->
            <entry key="MultipartException"
                   value="common/error/fileUploadError" />

        </map>
    </property>
    <property name="statusCodes">
        <map>
            <!-- omitted -->
            <!-- (5) -->
            <entry key="common/error/fileUploadError" value="400" />
        </map>
    </property>
    <!-- omitted -->
</bean>
```

Sr. No.	Description
(4)	In `exceptionMappings` of `SystemExceptionResolver`, add the definition for View (JSP) which is displayed when `MultipartException` occurs. In the above example, `"common/error/fileUploadError"` is specified.
(5)	Add the definition of HTTP status code which is received as response when `MultipartException` occurs. In the above example, `"400"` (Bad Request) is specified. By specifying client error (HTTP response code = 4xx), the level of log which is output by the class (`HandlerExceptionResolverLoggingInterceptor`) provided by the exception handling functionality of common library is `WARN` and not `ERROR`.

Add exception code settings when setting an exception code for `MultipartException`.

Exception code is output to the log which is output using log output functionality of common library.

Exception code can also be referred from View (JSP).

For referring to exception code from View (JSP), refer to *Method to display system exception code on screen*.

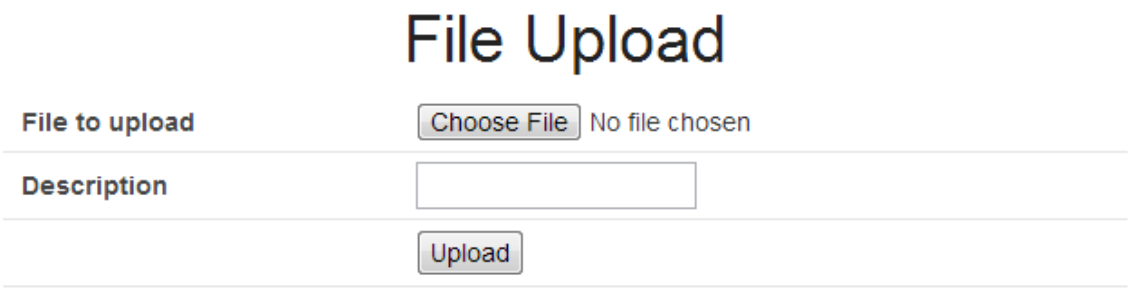
- applicationContext.xml

```
<bean id="exceptionCodeResolver"
    class="org.terasoluna.gfw.common.exception.
↪SimpleMappingExceptionCodeResolver">
    <property name="exceptionMappings">
        <map>
            <!-- (6) -->
            <entry key="MultipartException" value="e.xx.fw.6001" />
            <!-- omitted -->
        </map>
    </property>
    <property name="defaultExceptionCode" value="e.xx.fw.9001" />
    <!-- omitted -->
</bean>
```

Sr. No.	Description
(6)	In `exceptionMappings` of `SimpleMappingExceptionCodeResolver`, add the exception code to be applied when `MultipartException` occurs. In the above example, `"e.xx.fw.6001"` is specified. When it is not defined individually, exception code specified in `defaultExceptionCode` is applied.

Uploading a single file

The explanation about uploading a single file is given below.



There are 2 methods to upload a single file. One is by binding

`org.springframework.web.multipart.MultipartFile` object to the form object and the other is by receiving it directly as Controller argument. However, this guideline recommends the first method wherein it is received after it is bound with the form object.

The reason for this being, single field check of the uploaded file can be performed using Bean Validation.

How to receive a single file by binding it to form object is explained below.

Implementing form

```
public class FileUploadForm implements Serializable {

    // omitted

    private MultipartFile file; // (1)

    @NotNull
    @Size(min = 0, max = 100)
    private String description;

    // omitted getter/setter methods.

}
```

Sr. No.	Description
(1)	Define properties of `org.springframework.web.multipart.MultipartFile` in form object.

Implementing JSP

```
<form:form
  action="${pageContext.request.contextPath}/article/upload" method="post"
  modelAttribute="fileUploadForm" enctype="multipart/form-data"> <!-- (1) (2)␣
↪-->
  <table>
    <tr>
      <th width="35%">File to upload</th>
      <td width="65%">
        <form:input type="file" path="file" /> <!-- (3) -->
        <form:errors path="file" />
      </td>
    </tr>
    <tr>
```

```
      <th width="35%">Description</th>
      <td width="65%">
        <form:input path="description" />
        <form:errors  path="description" />
      </td>
    </tr>
    <tr>
      <td> </td>
      <td><form:button>Upload</form:button></td>
    </tr>
  </table>
</form:form>
```

Sr. No.	Description
(1)	Specify "multipart/form-data" in the enctype attribute of <form:form> element.
(2)	Specify attribute name of form object in the modelAttribute of <form:form> element. In the above example, "fileUploadForm" is specified.
(3)	Specify "file" in type attribute of <form:input> element and specify MultipartFile property name in path attribute. In the above example, the uploaded file is stored in "file" property of FileUploadForm object.

Implementing Controller

```
@RequestMapping("article")
@Controller
public class ArticleController {

    @Value("${upload.allowableFileSize}")
    private int uploadAllowableFileSize;

    // omitted

    // (1)
    @ModelAttribute
    public FileUploadForm setFileUploadForm() {
        return new FileUploadForm();
    }
```

```
    // (2)
    @RequestMapping(value = "upload", method = RequestMethod.GET, params =
↪"form")
    public String uploadForm() {
        return "article/uploadForm";
    }

    // (3)
    @RequestMapping(value = "upload", method = RequestMethod.POST)
    public String upload(@Validated FileUploadForm form,
            BindingResult result, RedirectAttributes redirectAttributes) {

        if (result.hasErrors()) {
            return "article/uploadForm";
        }

        MultipartFile uploadFile = form.getFile();

        // (4)
        if (!StringUtils.hasLength(uploadFile.getOriginalFilename())) {
            result.rejectValue(uploadFile.getName(), "e.xx.at.6002");
            return "article/uploadForm";
        }

        // (5)
        if (uploadFile.isEmpty()) {
            result.rejectValue(uploadFile.getName(), "e.xx.at.6003");
            return "article/uploadForm";
        }

        // (6)
        if (uploadAllowableFileSize < uploadFile.getSize()) {
            result.rejectValue(uploadFile.getName(), "e.xx.at.6004",
                    new Object[] { uploadAllowableFileSize }, null);
            return "article/uploadForm";
        }

        // (7)
        // omit processing of upload.

        // (8)
        redirectAttributes.addFlashAttribute(ResultMessages.success().add(
                "i.xx.at.0001"));

        // (9)
        return "redirect:/article/upload?complete";
    }

    @RequestMapping(value = "upload", method = RequestMethod.GET, params =
↪"complete")
```

```
    public String uploadComplete() {
        return "article/uploadComplete";
    }

        // omitted

}
```

Sr. No.	Description
(1)	Method of storing the form object for file upload in `Model`. In the above example, the attribute name for storing form object in `Model` is `"fileUploadForm"`.
(2)	Handler method for displaying upload screen.
(3)	Handler method for uploading files.
(4)	It is checked whether the files for upload are selected. To check if the files are selected, call `MultipartFile#getOriginalFilename` method and decide on the basis of whether file name is specified or not. In the above example, input validation error is thrown if the files are not selected.
(5)	It is checked whether an empty file is selected. To check if the selected file is not empty, call `MultipartFile#isEmpty` method to check for presence of contents. In the above example, input validation error is thrown if an empty file is selected.
(6)	It is checked whether the file size is within allowable range. To check the size of selected file, call `MultipartFile#getSize` method and check whether the size is within the allowable range. In the above example, input validation error is thrown if the file size exceeds the allowable range.

次のページに続く

Table. 4.49 – 前のページからの続き

Sr. No.	Description
(7)	Implement upload process. The above example does not cover any specific implementation; however process to store the file on a shared disk or database is performed.
(8)	As per the requirement, the processing result message notifying about successful upload is stored.
(9)	Once upload is complete, redirect to upload completion screen.

Note: Preventing duplicate upload

When uploading files, it is recommended to perform transaction token check and screen transition based on PRG pattern. With this, upload of same files caused due to double submission can be prevented.

For more details on how to prevent double submission, refer to *Double Submit Protection*.

Note: About MultipartFile

Methods to operate the uploaded file are provided in MultipartFile. For details on using each method, refer to JavaDoc of MultipartFile class.

Bean Validation of file upload

In the above implementation example, uploaded file is validated as a Controller process. However, here the uploaded file is validated using Bean Validation.

For validation details, refer to *Input Validation*.

Note: It is recommended to use Bean Validation since this makes maintenance of Controller processes easier.

Implementing validation to verify that the file is selected

```
// (1)
@Target({ ElementType.METHOD, ElementType.FIELD, ElementType.ANNOTATION_TYPE }
↪)
@Retention(RetentionPolicy.RUNTIME)
@Constraint(validatedBy = UploadFileRequiredValidator.class)
public @interface UploadFileRequired {
            String message() default "{com.examples.upload.UploadFileRequired.
↪message}";
    Class<?>[] groups() default {};
    Class<? extends Payload>[] payload() default {};

    @Target({ ElementType.METHOD, ElementType.FIELD, ElementType.ANNOTATION_
↪TYPE })
    @Retention(RetentionPolicy.RUNTIME)
    @Documented
    @interface List {
```

```
        UploadFileRequired[] value();
    }

}
```

```
// (2)
public class UploadFileRequiredValidator implements
    ConstraintValidator<UploadFileRequired, MultipartFile> {

    @Override
    public void initialize(UploadFileRequired constraint) {
    }

    @Override
    public boolean isValid(MultipartFile multipartFile,
        ConstraintValidatorContext context) {
        return multipartFile != null &&
            StringUtils.hasLength(multipartFile.getOriginalFilename());
    }

}
```

Sr. No.	Description
(1)	Create annotation to verify that the file is selected.
(2)	Create implementation class to verify that the file is selected.

Implementing validation to verify that the file is not empty

```
// (3)
@Target({ ElementType.METHOD, ElementType.FIELD, ElementType.ANNOTATION_TYPE }
↪)
@Retention(RetentionPolicy.RUNTIME)
@Constraint(validatedBy = UploadFileNotEmptyValidator.class)
public @interface UploadFileNotEmpty {
    String message() default "{com.examples.upload.UploadFileNotEmpty.message}
↪";
    Class<?>[] groups() default {};
    Class<? extends Payload>[] payload() default {};

    @Target({ ElementType.METHOD, ElementType.FIELD, ElementType.ANNOTATION_
↪TYPE })
    @Retention(RetentionPolicy.RUNTIME)
```

```
    @Documented
    @interface List {
        UploadFileNotEmpty[] value();
    }

}
```

```
// (4)
public class UploadFileNotEmptyValidator implements
    ConstraintValidator<UploadFileNotEmpty, MultipartFile> {

    @Override
    public void initialize(UploadFileNotEmpty constraint) {
    }

    @Override
    public boolean isValid(MultipartFile multipartFile,
        ConstraintValidatorContext context) {
        if (multipartFile == null ||
            !StringUtils.hasLength(multipartFile.getOriginalFilename())) {
            return true;
        }
        return !multipartFile.isEmpty();
    }

}
```

Sr. No.	Description
(3)	Create annotation to verify that the file is not empty.
(4)	Create implementation class to verify that the file is not empty.

Implementing validation to verify that file size is within allowable range

```
// (5)
@Target({ ElementType.METHOD, ElementType.FIELD, ElementType.ANNOTATION_TYPE }
↪)
@Retention(RetentionPolicy.RUNTIME)
@Constraint(validatedBy = UploadFileMaxSizeValidator.class)
public @interface UploadFileMaxSize {
    String message() default "{com.examples.upload.UploadFileMaxSize.message}
↪";
    long value() default (1024 * 1024);
```

```
    Class<?>[] groups() default {};
    Class<? extends Payload>[] payload() default {};

    @Target({ ElementType.METHOD, ElementType.FIELD, ElementType.ANNOTATION_
↪TYPE })
    @Retention(RetentionPolicy.RUNTIME)
    @Documented
    @interface List {
        UploadFileMaxSize[] value();
    }

}
```

```
// (6)
public class UploadFileMaxSizeValidator implements
    ConstraintValidator<UploadFileMaxSize, MultipartFile> {

    private UploadFileMaxSize constraint;

    @Override
    public void initialize(UploadFileMaxSize constraint) {
        this.constraint = constraint;
    }

    @Override
    public boolean isValid(MultipartFile multipartFile,
        ConstraintValidatorContext context) {
        if (constraint.value() < 0 || multipartFile == null) {
            return true;
        }
        return multipartFile.getSize() <= constraint.value();
    }

}
```

Sr. No.	Description
(5)	Create annotation to verify that the file size is within allowable range.
(6)	Create implementation class to verify that the file size is within allowable range.

Implementing form

```
public class FileUploadForm implements Serializable {

    // omitted

    // (7)
    @UploadFileRequired
    @UploadFileNotEmpty
    @UploadFileMaxSize
    private MultipartFile file;

    @NotNull
    @Size(min = 0, max = 100)
    private String description;

    // omitted getter/setter methods.

}
```

Sr. No.	Description
(7)	Assign annotation to `MultipartFile` field for validating uploaded file.

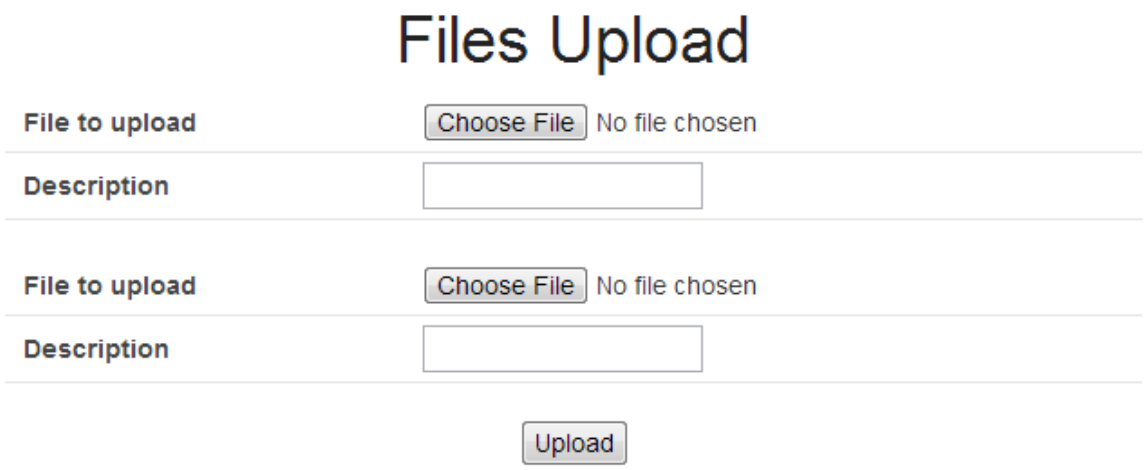
Implementing Controller

```
@RequestMapping(value = "upload", method = RequestMethod.POST)
public String uploadFile(@Validated FileUploadForm form,
        BindingResult result, RedirectAttributes redirectAttributes) {

    // (8)
    if (result.hasErrors()) {
        return "article/uploadForm";
    }

    MultipartFile uploadFile = form.getFile();

    // omit processing of upload.

    redirectAttributes.addFlashAttribute(ResultMessages.success().add(
            "i.xx.at.0001"));

    return "redirect:/article/upload";
}
```

Sr. No.	Description
(8)	Validation result of uploaded file is stored in `BindingResult`.

Uploading multiple files

This section explains about simultaneously uploading multiple files.



In order to upload multiple files simultaneously, it is necessary to receive `org.springframework.web.multipart.MultipartFile` object by binding it to the form object.

The explanation that has already been covered under single file upload has been omitted to avoid duplication.

Implementing form

```
// (1)
public class FileUploadForm implements Serializable {

    // omitted

    @UploadFileRequired
    @UploadFileNotEmpty
    @UploadFileMaxSize
    private MultipartFile file;

    @NotNull
    @Size(min = 0, max = 100)
    private String description;
```

```
    // omitted getter/setter methods.

}
```

```
public class FilesUploadForm implements Serializable {

    // omitted

    @Valid // (2)
    private List<FileUploadForm> fileUploadForms; // (3)

    // omitted getter/setter methods.

}
```

Sr. No.	Description
(1)	Class that maintains the information of each file (uploaded file itself and related form fields). In the above example, form object that was originally created to explain about single file upload, is re-used.
(2)	To carry out input validation through Bean Validation for the object maintained in list, assign `@Valid` annotation.
(3)	Define the object that maintains information of each file (uploaded file itself and related form fields) as List property.

Note: When only files are to be uploaded, `MultipartFile` object can also be defined as List property; however, for input validation of uploaded files using Bean Validation, there is better compatibility if the object that maintains information of each file, is defined as List property.

Implementing JSP

```
<form:form
  action="${pageContext.request.contextPath}/article/uploadFiles" method="post
↪"
  modelAttribute="filesUploadForm" enctype="multipart/form-data">
  <table>
```

```
    <tr>
      <th width="35%">File to upload</th>
      <td width="65%">
        <form:input type="file" path="fileUploadForms[0].file" /> <!-- (1) -->
        <form:errors path="fileUploadForms[0].file" />
      </td>
    </tr>
    <tr>
      <th width="35%">Description</th>
      <td width="65%">
        <form:input path="fileUploadForms[0].description" />
        <form:errors  path="fileUploadForms[0].description" />
      </td>
    </tr>
  </table>
  <table>
    <tr>
      <th width="35%">File to upload</th>
      <td width="65%">
        <form:input type="file" path="fileUploadForms[1].file" /> <!-- (1) -->
        <form:errors path="fileUploadForms[1].file" />
      </td>
    </tr>
    <tr>
      <th width="35%">Description</th>
      <td width="65%">
        <form:input path="fileUploadForms[1].description" />
        <form:errors path="fileUploadForms[1].description" />
      </td>
    </tr>
  </table>
  <div>
    <form:button>Upload</form:button>
  </div>
</form:form>
```

Sr. No.	Description
(1)	Specify the binding position of the uploaded file in List. Specify the binding position within List in `[]`. Start position begins with `0`.

Implementing Controller

```
@RequestMapping(value = "uploadFiles", method = RequestMethod.POST)
public String uploadFiles(@Validated FilesUploadForm form,
        BindingResult result, RedirectAttributes redirectAttributes) {
```

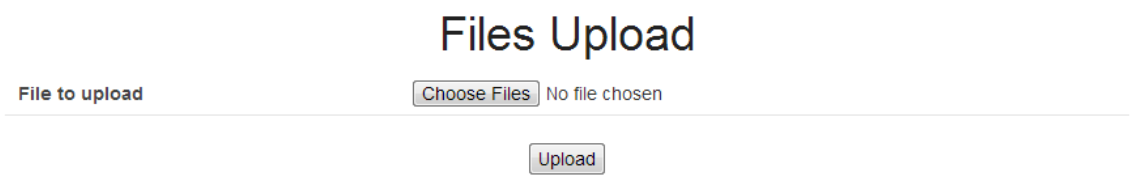
```
    if (result.hasErrors()) {
        return "article/uploadForm";
    }

    // (1)
    for (FileUploadForm fileUploadForm : form.getFileUploadForms()) {

        MultipartFile uploadFile = fileUploadForm.getFile();

        // omit processing of upload.

    }

    redirectAttributes.addFlashAttribute(ResultMessages.success().add(
            "i.xx.at.0001"));

    return "redirect:/article/upload?complete";
}
```

Table.4.50 :header-rows: 1 :widths: 10 90

Sr. No.	Description
(1)	Fetch `MultipartFile` from the object that maintains information of each file (uploaded file itself and related form fields) and implement upload process. The above example does not cover any specific implementation; however process to store the file on a shared disk or database is performed.

Uploading multiple files using the “multiple” attribute of HTML5

The method to simultaneously upload multiple files using “multiple” attribute of input tag supported by HTML5, is explained below.



The explanation that has already been covered under single file upload and multiple file upload has been omitted.

Implementing form

When uploading multiple files simultaneously using “multiple” attribute of HTML5 input tag, it is necessary to receive collection of `org.springframework.web.multipart.MultipartFile` object by binding it to form object.

```
// (1)
public class FilesUploadForm implements Serializable {

    // omitted

    // (2)
    @UploadFileNotEmpty
    private List<MultipartFile> files;

    // omitted getter/setter methods.

}
```

Sr. No.	Description
(1)	Form object that maintains the multiple uploaded files.
(2)	Declare `MultipartFile` class as list. In the above example, the annotation to verify that the file is not empty, is specified as input validation. Principally, a file size check or other mandatory checks are also required; however, they have been omitted in the above example.

Implementing Validator

When carrying out input validation for multiple `MultipartFile` objects stored in collection, it is necessary to implement Validator for Collection.

The section explains about creating Validator for Collection using the Validator created for single file.

```
// (1)
public class UploadFileNotEmptyForCollectionValidator implements
    ConstraintValidator<UploadFileNotEmpty, Collection<MultipartFile>> {

    // (2)
    private final UploadFileNotEmptyValidator validator =
        new UploadFileNotEmptyValidator();
```

```
    // (3)
    @Override
    public void initialize(UploadFileNotEmpty constraintAnnotation) {
        validator.initialize(constraintAnnotation);
    }

    // (4)
    @Override
    public boolean isValid(Collection<MultipartFile> values,
            ConstraintValidatorContext context) {
        for (MultipartFile file : values) {
            if (!validator.isValid(file, context)) {
                return false;
            }
        }
        return true;
    }

}
```

Sr. No.	Description
(1)	Class for performing implementation to verify that none of the files is empty. Specify `Collection<MultipartFile>` as the type of value to be verified.
(2)	In order to delegate the actual process to a Validator for single file, create an instance for that Validator.
(3)	Initialize the Validator. In the above example, Validator for single file that implements the actual process is initialized.
(4)	Verify that none of the file is empty. In the above example, each file is verified by calling the method of Validator for single file.

```
@Target({ ElementType.METHOD, ElementType.FIELD, ElementType.ANNOTATION_TYPE }
↪)
@Retention(RetentionPolicy.RUNTIME)
@Constraint(validatedBy =
    {UploadFileNotEmptyValidator.class,
```

```
    UploadFileNotEmptyForCollectionValidator.class}) // (5)
public @interface UploadFileNotEmpty {

    // omitted

}
```

Sr. No.	Description
(5)	Add the Validator class that carries out checks with respect to multiple files, to the annotation used for verification. Specify the class created in step (1) in the "validatedBy" attribute of `@Constraint` annotation. With this, the class created in step (1) is executed when validating the property with `@UploadFileNotEmpty` annotation.

Implementing JSP

```
<form:form
  action="${pageContext.request.contextPath}/article/uploadFiles" method="post
↪"
  modelAttribute="filesUploadForm2" enctype="multipart/form-data">
  <table>
    <tr>
      <th width="35%">File to upload</th>
      <td width="65%">
        <form:input type="file" path="files" multiple="multiple" /> <!-- (1) -
↪->
        <form:errors path="files" />
      </td>
    </tr>
  </table>
  <div>
    <form:button>Upload</form:button>
  </div>
</form:form>
```

Sr. No.	Description
(1)	In "path" attribute, specify "multiple" attribute by indicating property name of form object. By specifying "multiple" attribute, multiple files can be selected and uploaded using browser supporting HTML5.

Implementing Controller

```
@RequestMapping(value = "uploadFiles", method = RequestMethod.POST)
public String uploadFiles(@Validated FilesUploadForm form,
        BindingResult result, RedirectAttributes redirectAttributes) {
    if (result.hasErrors()) {
        return "article/uploadForm";
    }

    // (1)
    for (MultipartFile file : form.getFiles()) {

        // omit processing of upload.

    }

    redirectAttributes.addFlashAttribute(ResultMessages.success().add(
            "i.xx.at.0001"));

    return "redirect:/article/upload?complete";
}
```

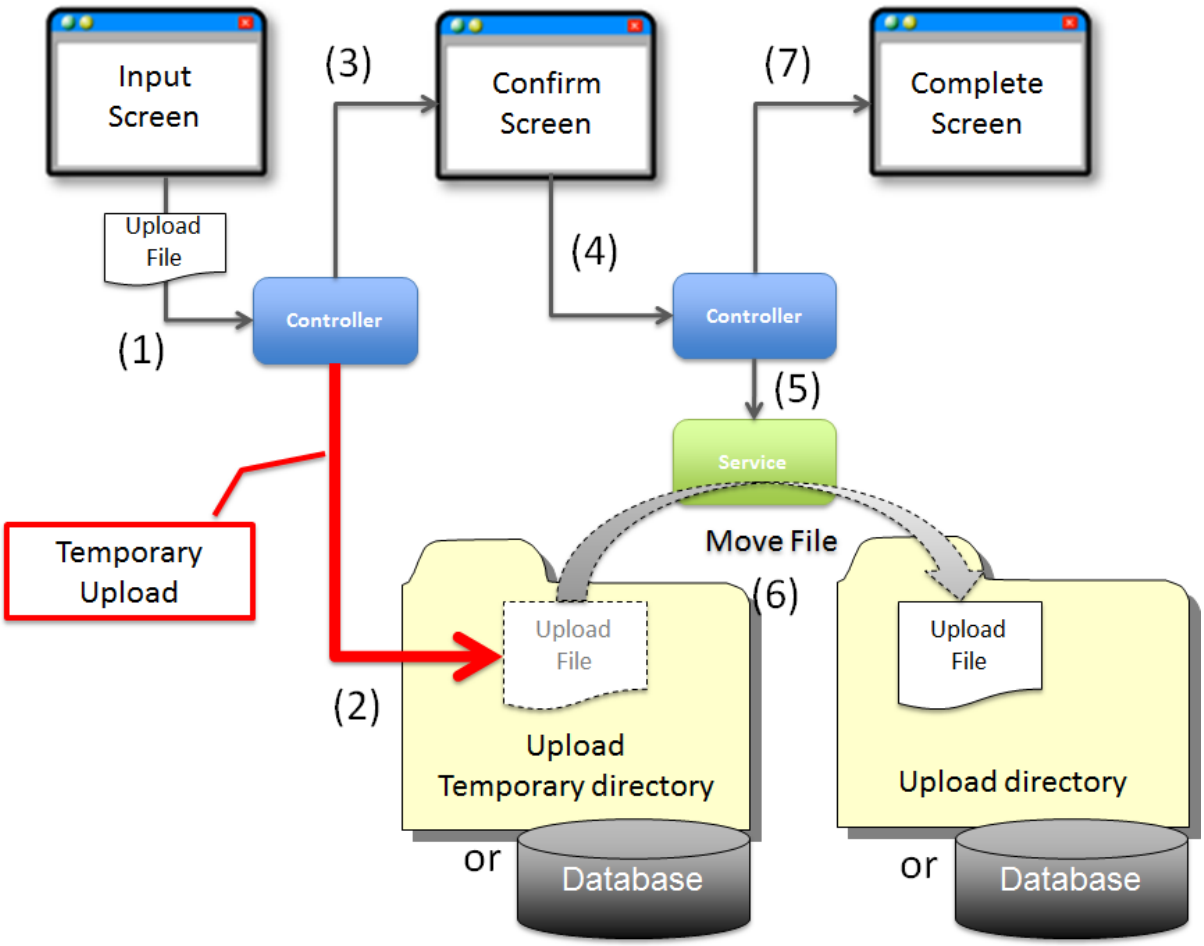
Sr. No.	Description
(1)	Implement upload process by fetching the list which stores `MultipartFile` objects from form object. The above example does not cover any specific implementation; however process to store the file on a shared disk or database is performed.

Temporary upload

Temporary upload is required when a file is to be uploaded midway through screen transitions like upload result confirmation screen etc.

Note: Contents of file stored in `MultipartFile` object may be deleted once the upload request is completed. Therefore, when the file contents are to be handled across requests, these contents and meta information (file name etc.) maintained in `MultipartFile` object need to be saved in a file or form.

The contents of file stored in `MultipartFile` object are deleted when step (3) of the following processing flow is completed.



Sr. No.	Description
(1)	On Input Screen, select the file to be uploaded and send a request for displaying Confirm Screen.
(2)	Controller temporarily saves contents of uploaded file in the temporary directory for application.
(3)	Controller returns View name of Confirm Screen and then displays the Confirm Screen.
(4)	On Confirm screen, send a request for executing the process.

次のページに続く

Table. 4.51 – 前のページからの続き

Sr. No.	Description
(5)	Controller calls Service method and executes process.
(6)	Service moves the temporary file saved in temporary directory to this directory or database.
(7)	Controller returns the View name which is required to display Complete Screen and then displays the Complete Screen.

Note: Temporary upload process is the responsibility of application layer; hence it is executed by Controller or Helper class.

Implementing Controller

Example for temporarily saving the uploaded file in a temporary directory, is shown below.

```
@Component
public class UploadHelper {

    // (2)
    @Value("${app.upload.temporaryDirectory}")
    private File uploadTemporaryDirectory;

    // (1)
    public String saveTemporaryFile(MultipartFile multipartFile)
        throws IOException {

        String uploadTemporaryFileId = UUID.randomUUID().toString();
        File uploadTemporaryFile =
            new File(uploadTemporaryDirectory, uploadTemporaryFileId);

        // (2)
        FileUtils.copyInputStreamToFile(multipartFile.getInputStream(),
                uploadTemporaryFile);

        return uploadTemporaryFileId;
    }

}
```

Sr. No.	Description
(1)	Create a method for executing temporary upload in Helper class. When there are multiple processes that perform file upload, it is recommended to have a common temporary upload process by creating a common Helper method.
(2)	Save the uploaded file as a temporary file. In the above example, contents of uploaded file are saved to a file by calling copyInputStreamToFile method of `org.apache.commons.io.FileUtils` class.

```
// omitted

@Inject
UploadHelper uploadHelper;

@RequestMapping(value = "upload", method = RequestMethod.POST, params =
↪"confirm")
public String uploadConfirm(@Validated FileUploadForm form,
        BindingResult result) throws IOException {

    if (result.hasErrors()) {
        return "article/uploadForm";
    }

    // (3)
    String uploadTemporaryFileId = uploadHelper.saveTemporaryFile(form
            .getFile());

    // (4)
    form.setUploadTemporaryFileId(uploadTemporaryFileId);
    form.setFileName(form.getFile().getOriginalFilename());

    return "article/uploadConfirm";
}
```

Sr. No.	Description
(3)	Call the Helper method to temporarily save the uploaded file. In the above example, ID by which the temporarily saved file is identified, is returned as the return value of Helper method.
(4)	Save the meta information of uploaded file (ID by which the file is identified, file name etc.) in form object. In the above example, name of the uploaded file and ID by which the temporarily saved file is identified, are stored in form object.

Note: Directory of temporary directories should be fetched from external properties as it may differ with the environment in which the application is deployed. For details on external properties, refer to *Properties Management*.

> **Warning:** In the above example, it is a file saved temporarily on the local disk of application server. However, when the application server is clustered, it needs to be saved in the database or on a shared disk. As a result, it is necessary to design a storage destination by considering even the non-functional requirements.
>
> Transaction management is necessary in case of saving the file to the database. As a result, the process to save it to the database will be delegated to Service method.

4.9.3 How to extend

Housekeeping of unnecessary files at the time of temporary upload

When uploading files using the temporary upload method, there is a possibility of unnecessary files piling up in temporary directory.
The cases are as follows:

- When there is interruption in screen operations after temporary upload
- When system error occurs during the screen operations after temporary upload
- When server stops during the screen operations after temporary upload etc ...

> **Warning:** A mechanism should be provided to delete unnecessary files as the disk may run out of space if such files are left to pile up.

This guideline explains about deleting unnecessary files using the "Task Scheduler" functionality provided by Spring Framework. For details on "Task Scheduler", refer to the official website "Task Execution and Scheduling".

Note: Although this guideline explains about how to use "Task Scheduler" functionality provided by Spring Framework; its usage is not mandatory. In an actual project, the infrastructure team may provide batch application (Shell application) to delete unnecessary files. In such cases, it is recommended to delete unnecessary files using the batch application created by infrastructure team.

Implementing component class to delete unnecessary files

Implement a component class to delete unnecessary files.

```
package com.examples.common.upload;

import java.io.File;
import java.util.Collection;
import java.util.Date;

import javax.inject.Inject;

import org.apache.commons.io.FileUtils;
import org.apache.commons.io.filefilter.FileFilterUtils;
import org.apache.commons.io.filefilter.IOFileFilter;
import org.springframework.beans.factory.annotation.Value;
import org.terasoluna.gfw.common.date.jodatime.JodaTimeDateFactory;

    // (1)
    public class UnnecessaryFilesCleaner {

    @Inject
    JodaTimeDateFactory dateFactory;

    @Value("${app.upload.temporaryFileSavedPeriodMinutes}")
    private int savedPeriodMinutes;

    @Value("${app.upload.temporaryDirectory}")
    private File targetDirectory;

    // (2)
    public void cleanup() {

        // calculate cutoff date.
        Date cutoffDate = dateFactory.newDateTime().minusMinutes(
                savedPeriodMinutes).toDate();

        // collect target files.
        IOFileFilter fileFilter = FileFilterUtils.ageFileFilter(cutoffDate);
        Collection<File> targetFiles = FileUtils.listFiles(targetDirectory,
                fileFilter, null);

        if (targetFiles.isEmpty()) {
            return;
        }

        // delete files.
        for (File targetFile : targetFiles) {
            FileUtils.deleteQuietly(targetFile);
        }
```

```
    }

}
```

Sr. No.	Description
(1)	Create component class to delete unnecessary files.
(2)	Implement the method to delete unnecessary files. In the above example, the files that have not been updated for a certain period of time from the last update, are treated as unnecessary files and are deleted.

Note: Directory path in which files to be deleted are stored or the time criteria for deletion etc. may differ depending upon the environment in which application is to be deployed. Hence they should be fetched from external properties. For details on external properties, refer to *Properties Management*.

Scheduling settings of the process for deleting unnecessary files

Carry out bean registration and task schedule settings for the POJO class that deletes unnecessary files.

- `applicationContext.xml`

```
<!-- omitted -->

<!-- (3) -->
<bean id="uploadTemporaryFileCleaner"
    class="com.examples.common.upload.UnnecessaryFilesCleaner" />

<!-- (4) -->
<task:scheduler id="fileCleanupTaskScheduler" />

<!-- (5) -->
<task:scheduled-tasks scheduler="fileCleanupTaskScheduler">
    <!-- (6)(7)(8) -->
    <task:scheduled ref="uploadTemporaryFileCleaner"
                    method="cleanup"
                    cron="${app.upload.temporaryFilesCleaner.cron}"/>
</task:scheduled-tasks>

<!-- omitted -->
```

Sr. No.	Description
(3)	POJO class that deletes unnecessary files should be registered in bean. In the above example, it is registered with "`uploadTemporaryFileCleaner`" ID.
(4)	Register the bean for task scheduler that executes the process to delete unnecessary files. In the above example, as pool-size attribute is omitted, this task scheduler executes the task in a single thread . When multiple tasks need to be executed simultaneously, some number should be specified in `pool-size` attribute.
(5)	Add the task to the task scheduler that deletes unnecessary files. In the above example, task is added to the task scheduler for which bean is registered in step (4).
(6)	In `ref` attribute, specify the bean that executes the process of deleting unnecessary files. In the above example, the bean registered in step (3) is specified.
(7)	In `method` attribute, specify the name of method executing the process of deleting unnecessary files. In the above example, cleanup method of bean registered in step (3) is specified.
(8)	In `cron` attribute, specify execution time of the process to delete unnecessary files. In the above example, cron definition is fetched from external properties.

Note: Specify the configuration value of `cron` attribute in "seconds minutes hour month year day" format.

Example:

- `0 */15 * * * *` : Executed in 0 minute, 15 minutes, 30 minutes and 45 minutes every hour.
- `0 0 * * * *` : Executed in 0 minute every hour.
- `0 0 9-17 * * MON-FRI` : Executed in 0 minute every hour from 9:00~17:00 on weekdays.

For details on specified value of cron, refer to CronSequenceGenerator - JavaDoc.

Execution time should be fetched from external properties as it may differ depending on the environment in which the application is to be deployed. For details on external properties, refer to *Properties Management*.

Tip: In the above example, cron is used as a trigger for executing tasks. However, other triggers namely fixed-delay and fixed-rate are also set by default and should be selectively used as per requirement.

When the default triggers do not satisfy the requirements, an independent trigger can be set by specifying the bean implementing `org.springframework.scheduling.Trigger` in trigger attribute.

4.9.4 Appendix

Security issues related to file upload

Following security issues need to be considered when providing File Upload functionality.

1. *Dos attack with respect to upload functionality*
2. *Attack by executing uploaded files on Web Server*
3. *Directory traversal attack*

Security measures are described below.

Dos attack with respect to upload functionality

Dos attack with respect to upload functionality is when load on the server is increased by continuously uploading large files, thereby crashing the server or reducing its response speed.

When there is no limit on the size of files to be uploaded and multipart request, the resistance to Dos attack becomes weak.
In order to enhance the resistance towards Dos attack, size limit needs to be set for a request, by using `<multipart-config>` element explained in *Application settings*.

Attack by executing uploaded files on Web Server

In this attack, the files on Web Server can be viewed/altered/deleted by uploading and executing the script files (php, asp, aspx, jsp etc.) that are executable on Web Server (Application Server).

With Web Server as a platform, another server present in the same network as the Web server, is also vulnerable to such attack.

Measures to be taken against this attack are as follows:

- To view the file contents through a process that displays the contents, without placing uploaded files in the public directory of Web Server (Application Server).
- To ensure that executable script file cannot be uploaded on Web server (Application Server) by restricting the extension of files that can be uploaded.

The attacks can be prevented by implementing either of the above measures; however it is always recommended to implement both the measures.

Directory traversal attack

Directory traversal attack is an attack that accesses the files on the server that originally should not been accessed, by accessing the file system using inputs which include strings like "../" etc.

For example, in the web application wherein the file uploaded by the user is placed in the predetermined directory on the server, the file with the name "../../../../somewhere/attack" is uploaded according to implementation method and the file is placed in a directory other than predetermined directory.

In that case, the file on the server is likely to get tampered by the file uploaded by the attacker.

A risk of directory traversal is likely in the file download function as well, along with file upload function.

For example, a type of attack can be considered wherein the attacker would obtain the details of "/etc/passwd" by entering "../../../../etc/passwd", in case of Web application wherein file is downloaded in accordance with the file name entered by the user.

Countermeasures for this attack are as below.

- When the uploaded file is to be stored on the server, it should be stored by using a different name, and the original file name and input value from user should not be used. It should be stored in a format which is not used for actual file access such as storing the correspondence relation with the file name on the server outside DB, for the original file name.
- When the file is to be accessed from the server, a request should be sent using an identifier for request instead of using actual file name and then converted to the file name on the server side. For example, identifiers “id01” and “id02” are used for actual file names “file_A” and “file_B” wherein if the client requests for “id01”, corresponding “file_A” on the server is accessed.

Tip: Another countermeasure can be taken into consideration wherein file path thus entered is normalised (”./” or ”../” etc, should be developed in a format which does not include strings with specific significance in the file system) and the access is given after checking whether the path matches (begins-with match) with the path already determined. However, if input value encoding and the variation in the path format according to OS is taken into consideration, it is difficult to determine whether the normalization has been done appropriately. Hence, it is preferable to avoid accessing the file system by using the input value from user.

File upload using Commons FileUpload

If File Upload functionality of Servlet 3.0 is only used partially on Application Server, it may likely result into garbling of multi byte characters of file names or request parameters.

For example: If File Upload functionality of Servlet 3.0 is used on WebLogic 12.1.3, it has been confirmed that multi byte characters of fields to be sent along with file are garbled. Note that it has been corrected in WebLogic 12.2.1.

This problem can be avoided using Commons FileUpload. Therefore, this guideline describes about file upload using Commons FileUpload as a temporary measure for the specific environment where problems are likely to occur. Using Commons FileUpload is not recommended where problems are not likely.

Perform the following settings when using Commons FileUpload.

xxx-web/pom.xml

```
<!-- (1) -->
<dependency>
    <groupId>commons-fileupload</groupId>
    <artifactId>commons-fileupload</artifactId>
</dependency>
```

Sr. No.	Description
(1)	Add dependency to `commons-fileupload`.

Note: In the above setting example, since it is assumed that the dependent library version is managed by the parent project terasoluna-gfw-parent , specifying the version in pom.xml is not necessary. The above dependent library used by terasoluna-gfw-parent is defined by Spring IO Platform.

Warning: In case of using Apache Commons FileUpload, security vulnerabilities reported in CVE-2014-0050 and CVE-2016-3092 are likely to occur. Confirm that there are no vulnerabilities in the version of Apache Commons FileUpload to be used.

When using Apache Commons FileUpload, version 1.3.2 or above should be used.

Note that, if a version managed by Spring IO Platform Athens-SR2.RELEASE which is in conformance with TERASOLUNA Server Framework for Java version 5.3.0.RELEASE is used, vulnerabilities reported in CVE-2014-0050 and CVE-2016-3092 do not occur. When Apache Commons FileUpload version is to be changed intentionally, a version wherein corresponding vulnerability has been addressed must be specified.

`xxx-web/src/main/resources/META-INF/spring/applicationContext.xml`

```
<!-- (1) -->
<bean id="filterMultipartResolver"
    class="org.springframework.web.multipart.commons.CommonsMultipartResolver">
    <property name="maxUploadSize" value="10240000" /><!-- (2) -->
</bean>

<!-- ... -->
```

Sr. No.	Description
(1)	Perform bean definition of `CommonsMultipartResolver` with `MultipartResolver` implemented using Commons FileUpload. Specify `"filterMultipartResolver"` in bean ID.
(2)	Set maximum size allowed in file upload. In case of Commons FileUpload, it should be noted that the maximum value is the entire size of request including header. Moreover, **as the default value is -1 (unlimited), make sure to set a value.** For other properties, refer to JavaDoc.

Warning: In case of using Commons Fileupload, `MultipartResolver` should be defined in `applicationContext.xml` and not in `spring-mvc.xml`. It should be deleted if defined in `spring-mvc.xml`.

`xxx-web/src/main/webapp/WEB-INF/web.xml`

```
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/
↪ns/javaee/web-app_3_0.xsd"
    version="3.0">

    <servlet>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-
↪class>
        <!-- omitted -->
        <!-- (1) -->
        <!-- <multipart-config>...</multipart-config> -->
    </servlet>

    <!-- (2) -->
    <filter>
```

```
        <filter-name>MultipartFilter</filter-name>
        <filter-class>org.springframework.web.multipart.support.MultipartFilter</
↪filter-class>
    </filter>
    <filter-mapping>
        <filter-name>MultipartFilter</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>

    <!-- omitted -->

</web-app>
```

Sr.No.	Description
(1)	When using Commons FileUpload, an upload function of Servlet 3.0 should be disabled. If `<multipart-config>` element is present in `DispatcherServlet` definition, make sure to delete the same.
(2)	When using Commons Fileupload, `MultipartFilter` must be defined to enable security countermeasures which use Spring Security. `MultipartFilter`mapping should be defined before defining springSecurityFilterChain (Servlet Filter of Spring Security).

Tip: `MultipartFilter`is a mechanism to perform the file upload process by fetching `MultipartResolver`registered with bean ID `"filterMultipartResolver"` from DI container (`applicationContext.xml`).

4.10 File Download

4.10.1 Overview

This chapter explains the functionality to download a file from server to client, using Spring.
It is recommended to use Spring MVC View for rendering the files.

> **Note:** It is not recommended to include file rendering logic in controller class.
>
> This is because, it deviates from the role of a controller. Moreover, a View can be easily changed when it is isolated from the controller.

Overview of File Download process is given below.

1. DispatchServlet sends file download request to the controller.
2. Controller fetches file display information.
3. Controller selects View.
4. File rendering is performed in View.

In order to perform file rendering in a Spring based Web application;
this guideline recommends implementation of custom view.
To implement custom view, `org.springframework.web.servlet.View` interface
is provided in Spring framework.

For PDF files

`org.springframework.web.servlet.view.document.AbstractPdfView` class of Spring is used as a subclass
when rendering PDF files using model information.

For Excel files

`org.springframework.web.servlet.view.document.AbstractXlsxView` class of Spring is used as a subclass
when rendering Excel files using model information.

For file formats other than those specified above, various types of View implementations are provided in Spring.

For technical details on View, refer to Spring Reference View technologies.

`org.terasoluna.gfw.web.download.AbstractFileDownloadView` provided by common library is the

abstract class to download arbitrary files.

Define this class as a subclass to render files in formats other than PDF or Excel.

Tip: A countermeasure for directory traversal attack may become necessary while offering a file download function. For directory traversal attack, refer *Directory traversal attack* .

4.10.2 How to use

Downloading PDF files

For rendering PDF files, it is necessary to create a class that

inherits `org.springframework.web.servlet.view.document.AbstractPdfView` provided by Spring.

The procedure to download a PDF file using controller is explained below.

Implementation of Custom View

Implementation of class that inherits AbstractPdfView

```
@Component  // (1)
public class SamplePdfView extends AbstractPdfView {  // (2)

  @Override
  protected void buildPdfDocument(Map<String, Object> model,
          Document document, PdfWriter writer, HttpServletRequest request,
          HttpServletResponse response) throws Exception {  // (3)

      document.add(new Paragraph((Date) model.get("serverTime")).toString());
```

```
  }
}
```

Sr. No.	Description
(1)	In this example, this class comes under the scope of component scanning by using `@Component` annotation. It will also come under the scope of `org.springframework.web.servlet.view.BeanNameViewResolver` which is described later.
(2)	Inherit `AbstractPdfView`.
(3)	Execute `buildPdfDocument` method.

`AbstractPdfView` uses iText for PDF rendering.

Therefore, it is necessary to add itext definition to pom.xml of Maven.

```
<dependencies>
    <!-- omitted -->
    <dependency>
        <groupId>com.lowagie</groupId>
        <artifactId>itext</artifactId>
        <exclusions>
            <exclusion>
                <artifactId>xml-apis</artifactId>
                <groupId>xml-apis</groupId>
            </exclusion>
            <exclusion>
                <artifactId>bctsp-jdk14</artifactId>
                <groupId>org.bouncycastle</groupId>
            </exclusion>
            <exclusion>
                <artifactId>jfreechart</artifactId>
                <groupId>jfree</groupId>
            </exclusion>
            <exclusion>
                <artifactId>dom4j</artifactId>
                <groupId>dom4j</groupId>
```

```
            </exclusion>
            <exclusion>
                <groupId>org.swinglabs</groupId>
                <artifactId>pdf-renderer</artifactId>
            </exclusion>
            <exclusion>
                <groupId>org.bouncycastle</groupId>
                <artifactId>bcprov-jdk14</artifactId>
            </exclusion>
        </exclusions>
    </dependency>
    <dependency>
        <groupId>org.bouncycastle</groupId>
        <artifactId>bcprov-jdk14</artifactId>
        <version>1.38</version>
    </dependency>
</dependencies>
```

Note: In the above setting example, since it is assumed that the dependent library version is managed by the parent project terasoluna-gfw-parent , specifying the version in pom.xml is not necessary. The above itext used by terasoluna-gfw-parent is defined by Spring IO Platform.

Definition of ViewResolver

`org.springframework.web.servlet.view.BeanNameViewResolver` is a class, that selects View to be executed using bean name stored in Spring context.

When using `BeanNameViewResolver`, it is recommended to define such that `BeanNameViewResolver` is executed before

- `ViewResolver` for JSP (`InternalResourceViewResolver`)
- `ViewResolver` for Tiles (`TilesViewResolver`)

which are generally used.

Note: Spring Framework provides various types of `ViewResolver` and it allows chaining of multiple `ViewResolver`. Therefore, some unintended View may get selected under certain conditions.

It is possible to avoid such a situation by setting appropriate priority order in `ViewResolver`. Method to set priority order differs depending on definition method of `ViewResolver`.

- When defining `ViewResolver` using `<mvc:view-resolvers>` element added from Spring Frame-

work 4.1, definition order of `ViewResolver` specified in child element will be the priority order. (executed sequentially from top)

- When specifying `ViewResolver`using `<bean>`element in a conventional way, set priority order in `order`property. (It is executed starting from smallest setting value).

bean definition file

```
<mvc:view-resolvers>
    <mvc:bean-name /> <!-- (1) (2) -->
    <mvc:jsp prefix="/WEB-INF/views/" />
</mvc:view-resolvers>
```

Sr. No.	Description
(1)	Define `BeanNameViewResolver` using `<mvc:bean-name>` element added from Spring Framework 4.1.
(2)	Define `<mvc:bean-name>`element right at the top so that it has a higher priority than the generally used `ViewResolver`(`ViewResolver`for JSP).

Tip: `<mvc:view-resolvers>` element is an XML element added from Spring Framework 4.1. If `<mvc:view-resolvers>` element is used, it is possible to define `ViewResolver` in a simple way.

Example of definition when `<bean>` element used in a conventional way is given below.

```
<bean class="org.springframework.web.servlet.view.BeanNameViewResolver">
    <property name="order" value="0"/>
</bean>

<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver
↪">
    <property name="prefix" value="/WEB-INF/views/" />
    <property name="suffix" value=".jsp" />
    <property name="order" value="1" />
</bean>
```

In `order`property, specify a value that is lesser than `InternalResourceViewResolver` to ensure that it

gets a high priority.

Specifying View in controller

With the help of `BeanNameViewResolver`, by returning "samplePDFView" in Controller,
a view named "samplePDFView" gets used from the BeanIDs stored in Spring Context.

Java source code

```
@RequestMapping(value = "home", params= "pdf", method = RequestMethod.GET)
public String homePdf(Model model) {
        model.addAttribute("serverTime", new Date());
        return "samplePdfView";    // (1)
}
```

Sr. No.	Description
(1)	With "samplePdfView" as the return value of method, `SamplePdfView` class stored in Spring context is executed.

Following PDF file can be opened after executing the above procedure.

Downloading Excel files

For rendering EXCEL files, it is necessary to create a class that
inherits `org.springframework.web.servlet.view.document.AbstractXlsxView` provided by Spring.
The procedure to download an EXCEL file using controller is explained below.

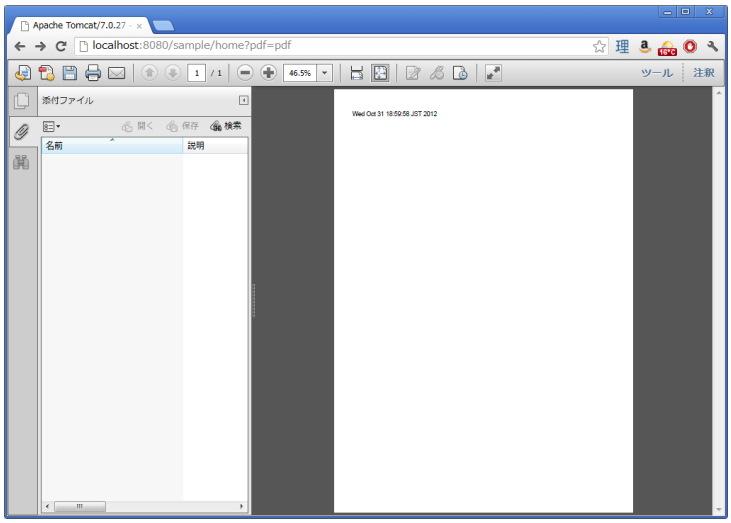


Figure.4.18 **Picture - FileDownload PDF**

Implementation of Custom View

Implementation of class that inherits AbstractXlsxView

```
@Component  // (1)
public class SampleExcelView extends AbstractXlsxView {  // (2)

    @Override
    protected void buildExcelDocument(Map<String, Object> model,
            Workbook workbook, HttpServletRequest request,
            HttpServletResponse response) throws Exception {  // (3)
        Sheet sheet;
        Cell cell;

        sheet = workbook.createSheet("Spring");
        sheet.setDefaultColumnWidth(12);

        // write a text at A1
        cell = getCell(sheet, 0, 0);
        setText(cell, "Spring-Excel test");

        cell = getCell(sheet, 2, 0);
        setText(cell, ((Date) model.get("serverTime")).toString());
    }

    private Cell getCell(Sheet sheet, int rowNumber, int cellNumber) {
        Row row = sheet.createRow(rowNumber);
        return row.createCell(cellNumber);
    }

    private void setText(Cell cell, String text) {
        cell.setCellValue(text);
```

```
    }
}
```

Sr. No.	Description
(1)	In this example, this class comes under the scope of component scanning by using `@Component` annotation. It will also come under the scope of `org.springframework.web.servlet.view.BeanNameViewResolver` which is described earlier.
(2)	Inherit `AbstractXlsxView`.
(3)	Execute `buildExcelDocument` method.

`AbstractXlsxView`uses Apache POI to render EXCEL file.

Therefore, it is necessary to add POI definition to the pom.xml file of Maven.

```
<dependencies>
    <!-- omitted -->
    <dependency>
        <groupId>org.apache.poi</groupId>
        <artifactId>poi-ooxml</artifactId>
        <exclusions>
            <exclusion>
                <groupId>stax</groupId>
                <artifactId>stax-api</artifactId>
            </exclusion>
        </exclusions>
    </dependency>
</dependencies>
```

Note: Since stax-api on which poi-ooxml is dependent, is provided as a standard from SE, the library is not required. Also, since a conflict is likely in the library, `<exclusions>` element should be added and

the relevant library should not be added in the application.

Note: In the above setting example, since it is assumed that the dependent library version is managed by the parent project terasoluna-gfw-parent , specifying the version in pom.xml is not necessary. The above dependent library used by terasoluna-gfw-parent is defined by Spring IO Platform.

Also, `AbstractExcelView` uses @Deprecated annotation from Spring Framework 4.2. Hence, it is recommended to use `AbstractXlsxView`in the same way even if you want to use a xls file. For details, refer AbstractExcelView - JavaDoc.

Definition of ViewResolver

Settings are same as that for PDF file rendering. For details, refer to *Definition of ViewResolver*.

Specifying View in controller

With the help of `BeanNameViewResolver`, by returning "sampleExcelView" in Controller,
a view named "sampleExcelView" gets used from the BeanIDs stored in Spring Context.

Java source

```
@RequestMapping(value = "home", params= "excel", method = RequestMethod.GET)
public String homeExcel(Model model) {
        model.addAttribute("serverTime", new Date());
        return "sampleExcelView";  // (1)
}
```

Sr. No.	Description
(1)	With "sampleExcelView"as the return value of method, `SampleExcelView` class stored in Spring context is executed.

EXCEL file can be opened as shown below after executing the above procedures.

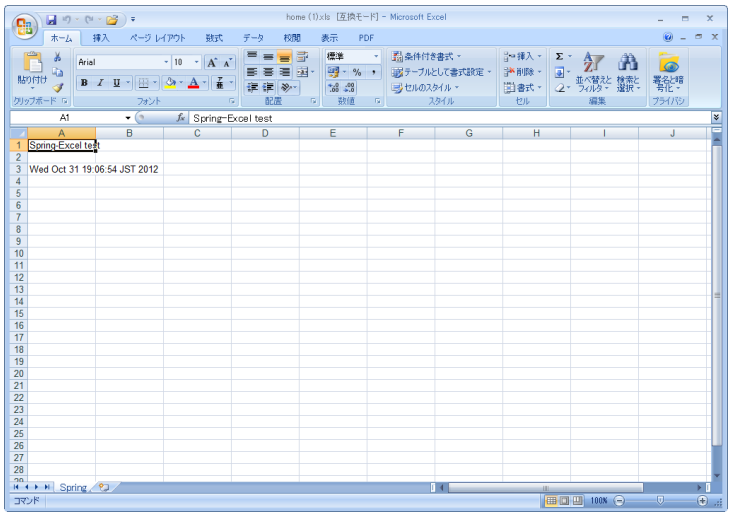


Figure.4.19 **Picture - FileDownload EXCEL**

Downloading arbitrary files

To download files in formats other than PDF or EXCEL,

class that inherits `org.terasoluna.gfw.web.download.AbstractFileDownloadView` provided by common library can be implemented.

Following steps should be implemented in `AbstractFileDownloadView` to render files in other format.

1. Fetch InputStream in order to write to the response body.
2. Set information in HTTP header.

How to implement file download using controller is explained below.

Implementation of Custom View

The example of text file download is given below.

Implementation of class that inherits AbstractFileDownloadView

```
@Component  // (1)
public class TextFileDownloadView extends AbstractFileDownloadView {  // (2)

    @Override
    protected InputStream getInputStream(Map<String, Object> model,
            HttpServletRequest request) throws IOException {  // (3)
```

```
        Resource resource = new ClassPathResource("abc.txt");
        return resource.getInputStream();
    }

    @Override
    protected void addResponseHeader(Map<String, Object> model,
            HttpServletRequest request, HttpServletResponse response) {  // (4)
        response.setHeader("Content-Disposition",
                "attachment; filename=abc.txt");
        response.setContentType("text/plain");

    }
}
```

Sr. No.	Description
(1)	In this example, this class comes under the scope of component scanning by using `@Component` annotation. It will also come under the scope of `org.springframework.web.servlet.view.BeanNameViewResolver` which is described earlier.
(2)	Inherit `AbstractFileDownloadView`.
(3)	Execute `getInputStream` method. `InputStream` to be downloaded should be returned.
(4)	Execute `addResponseHeader method`. Set Content-Disposition or ContentType as per the file to be downloaded.

Definition of ViewResolver

Settings are same as that of PDF file rendering. For details, refer to *Definition of ViewResolver*.

Specifying View in controller

With the help of `BeanNameViewResolver`, by returning “textFileDownloadView” in Controller,

a view named “textFileDownloadView” gets used from the BeanIDs stored in Spring Context.

Java source

```
@RequestMapping(value = "download", method = RequestMethod.GET)
public String download() {
    return "textFileDownloadView"; // (1)
}
```

Sr. No.	Description
(1)	With “textFileDownloadView”as the return value of method, `TextFileDownloadView` class stored in Spring context is executed.

Tip: As described above, Model information can be rendered in various types of Views using Spring. Spring supports rendering engine such as Jasper Reports and returns various types of views. For details, refer to the official Spring website Spring reference.

4.11 Screen Layout using Tiles

4.11.1 Overview

When developing a Web application with common layouts such as header, footer and side menu, maintaining the layouts becomes complicated if the common parts are coded in all JSPs.

For example, if the header design needs to be modified, the same modifications must be done for all JSPs.

In JSP development, when the same layout is used in many screens, it is recommended to use Apache Tiles (hereafter referred to as Tiles).

Reasons for using Tiles are as follows:

1. To eliminate layout errors by designer
2. To reduce redundant codes
3. To change oversized layouts easily

Tiles can combine different JSPs by defining an integrated screen layout.

As a result, the need to describe extra code in each JSP file is eliminated, thereby facilitating developer operations.

For example, if multiple screens have the following layout structure,

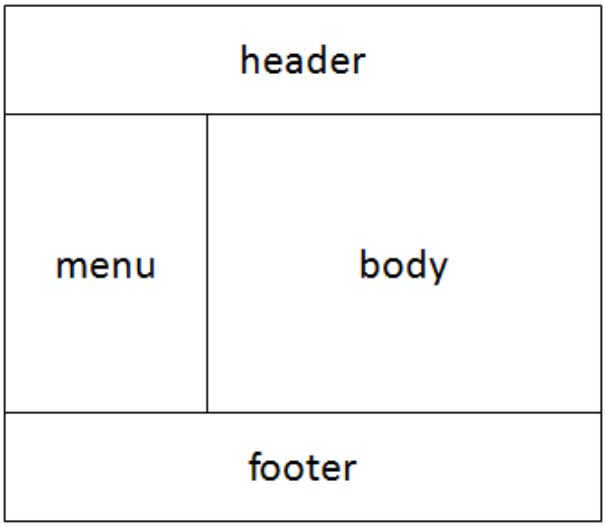


Figure.4.20 **Picture - Image of screen layout**

By using Tiles, one can focus only on creating the body without having to include and specify the sizes of header, menu and footer, in all the screens with the same layout.

Actual JSP file is as follows:

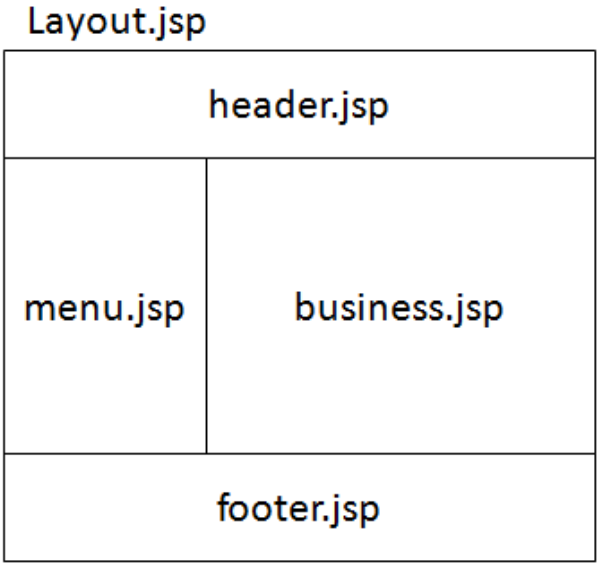


Figure.4.21 **Picture - Image of layout jsp**

Therefore, after configuring the screen layout using Tiles, only the JSP file corresponding to business process (business.jsp) may be created for each screen.

Note: In some cases, it is better to avoid using Tiles. For example, using Tiles in an error screen is not recommended due to the following reasons.

- If an error occurs due to Tiles during error screen display, analyzing the errors becomes difficult. (In case of double failure)
- Tiles Template is not necessarily always used to display screens in the JSP set by the <error-pages> tag of web.xml.

4.11.2 How to use

pom.xml setting

To use Tiles in Maven, following dependency should be added to pom.xml.

```
<dependency>
    <groupId>org.terasoluna.gfw</groupId>
    <artifactId>terasoluna-gfw-recommended-web-dependencies</artifactId><!-- (1) -->
    <type>pom</type><!-- (2) -->
</dependency>
```

Sr. No.	Description
(1)	Add terasoluna-gfw-recommended-web-dependencies defined for the group of web related libraries, to dependency.
(2)	Dependencies such as terasoluna-gfw-recommended-web-dependencies are defined only in pom file; hence `<type>pom</type>` needs to be specified.

Note: In the above setting example, since it is assumed that the dependent library version is managed by the parent project terasoluna-gfw-parent , specifying the version in pom.xml is not necessary.

Integration of Spring MVC and Tiles

It is advisable to use `org.springframework.web.servlet.view.tiles3.TilesViewResolver` for integrating Spring MVC and Tiles.

Implementation of Spring MVC Controller (returning View name) need not be changed.

How to configure is shown below.

Defining Bean (ViewResolver, TilesConfigurer)

- spring-mvc.xml

```
<mvc:view-resolvers>
    <mvc:tiles /> <!-- (1) -->
    <mvc:jsp prefix="/WEB-INF/views/" /> <!-- (2) -->
</mvc:view-resolvers>

<!-- (3) -->
<mvc:tiles-configurer>
    <mvc:definitions location="/WEB-INF/tiles/tiles-definitions.xml" />
</mvc:tiles-configurer>
```

Sr. No.	Description
(1)	Define `TilesViewResolver` using `<mvc:tiles>` element added from Spring Framework 4.1. By defining it above `<mvc:jsp>` element, first resolve `View` by referring to Tiles definition file (`tiles-definitions.xml`). If View name returned from Controller matches with `name` attribute pattern of `definition` element in Tiles definition file, `View` is resolved by `TilesViewResolver`.
(2)	Define `InternalResourceViewResolver` for JSP using `<mvc:jsp>` element added from Spring Framework 4.1. By defining it below `<mvc:tiles>` element, resolve `View` using "`InternalResourceViewResolver` for JSP" only for the View names that could not be resolved using `TilesViewResolver`. If a JSP file corresponding to View name exists under `/WEB-INF/views/` , `View` is resolved by `InternalResourceViewResolver` for JSP.
(3)	Read Tiles definition file using `<mvc:tiles-configurer>` element added from Spring Framework 4.1. Specify Tiles definition file in `location` attribute of `<mvc:definitions>` element.

Tip: `<mvc:view-resolvers>` element is an XML element added from Spring Framework 4.1. If `<mvc:view-resolvers>` element is used, it is possible to define `ViewResolver` in a simple way.

Example of definition when `<bean>` element is used in a conventional way is given below.

```
<bean id="tilesViewResolver"
    class="org.springframework.web.servlet.view.tiles3.TilesViewResolver">
    <property name="order" value="1" />
</bean>

<bean id="tilesConfigurer"
    class="org.springframework.web.servlet.view.tiles3.TilesConfigurer">
```

```
    <property name="definitions">
        <list>
            <value>/WEB-INF/tiles/tiles-definitions.xml</value>
        </list>
    </property>
</bean>

<bean id="viewResolver"
    class="org.springframework.web.servlet.view.
↪InternalResourceViewResolver">
    <property name="prefix" value="/WEB-INF/views/" />
    <property name="suffix" value=".jsp" />
    <property name="order" value="2" />
</bean>
```

In `order`property, specify a value that is lesser than `InternalResourceViewResolver` to ensure that it gets a high priority.

Tiles Definition

- tiles-definitions.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE tiles-definitions PUBLIC
   "-//Apache Software Foundation//DTD Tiles Configuration 3.0//EN"
   "http://tiles.apache.org/dtds/tiles-config_3_0.dtd"> <!-- (1) -->

<tiles-definitions>
    <definition name="layouts"
        template="/WEB-INF/views/layout/template.jsp"> <!-- (2) -->
        <put-attribute name="header"
            value="/WEB-INF/views/layout/header.jsp" /> <!-- (3) -->
        <put-attribute name="footer"
            value="/WEB-INF/views/layout/footer.jsp" /> <!-- (4) -->
    </definition>

    <definition name="*/*" extends="layouts"> <!-- (5) -->
        <put-attribute name="title" value="title.{1}.{2}" /> <!-- (6) -->
        <put-attribute name="body" value="/WEB-INF/views/{1}/{2}.jsp" /> <!--␣
↪(7) -->
    </definition>
</tiles-definitions>
```

Sr. No.	Description
(1)	Define dtd of tiles.
(2)	Define the parent layout structure. In 'template' attribute, specify the jsp file where layout is defined.
(3)	Specify the jsp file that defines header.
(4)	Specify the jsp file that defines footer.
(5)	Layout definition which is called when it is same as name pattern at the time of 'create' request. Extended layouts definition is also applied.
(6)	Specify title. Fetch the value from properties incorporated in spring-mvc. (In the following description, it is set in application-messages.properties.) {1},{2} correspond to the 1st and 2nd "*" of "*/*" request.
(7)	Design the location of jsp file that defines the body such that, request path matches with {1} and JSP name matches with {2}. With this, the efforts to describe definition for each request can be saved.

Note: For the screens where Tiles is not to be applied (error screen etc.), it is necessary to set a file structure that excludes use of Tiles. In Blank project, /WEB-INF/views/common/error/xxxError.jsp format is used so that InternalResourceViewResolver can be used (and so that it does not change to the "*/*" format) on error screen.

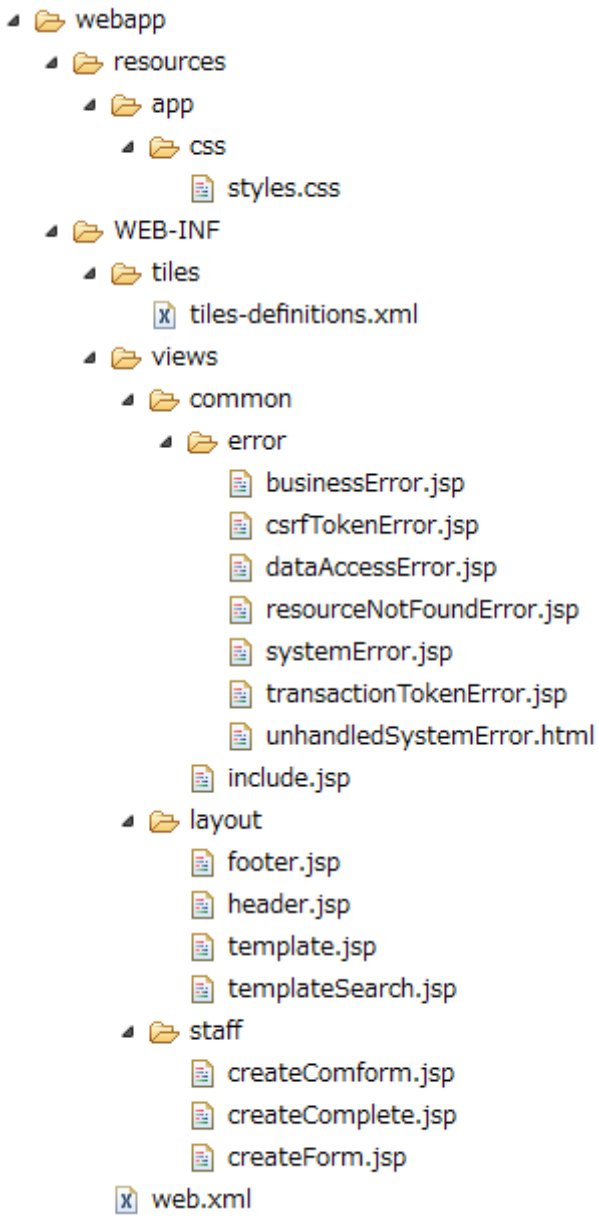
- *application-messages.properties*

```
title.staff.createForm = Create Staff Information
```

Note: For details on message properties file, refer to *Message Management*.

Following is the file structure when Tiles is set.

- tiles File Path



Custom tag settings

Custom tag (TLD) needs to be set to use Tiles.

- /WEB-INF/views/common/include.jsp

```
<%@ page session="false"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt"%>
<%@ taglib uri="http://www.springframework.org/tags" prefix="spring"%>
<%@ taglib uri="http://www.springframework.org/tags/form" prefix="form"%>
<%@ taglib uri="http://www.springframework.org/security/tags" prefix="sec"%>
<%@ taglib uri="http://terasoluna.org/functions" prefix="f"%>
<%@ taglib uri="http://terasoluna.org/tags" prefix="t"%>
<%@ taglib uri="http://tiles.apache.org/tags-tiles" prefix="tiles"%> <!-- (1)␣
↪-->
<%@ taglib uri="http://tiles.apache.org/tags-tiles-extras" prefix="tilesx"%>
↪<!-- (2) -->
```

Sr. No.	Description
(1)	Add a custom tag (TLD) definition for Tiles.
(2)	Add a custom tag (TLD) definition for Tiles-extras.

For details about custom tags of Tiles, refer to Here.

Tip:

Tiles version-2 had one taglib, but tiles-extras taglib is added from version-3.

useAttribute tag which was available in tiles taglib in version-2 is moved to tiles-extras taglib from version-3, hence should be careful while using.

e.g.) *<tiles:useAttribute>* : version 2 -> *<tilesx:useAttribute>* : version 3

- web.xml

```
<jsp-config>
    <jsp-property-group>
        <url-pattern>*.jsp</url-pattern>
        <el-ignored>false</el-ignored>
        <page-encoding>UTF-8</page-encoding>
        <scripting-invalid>false</scripting-invalid>
        <include-prelude>/WEB-INF/views/common/include.jsp</include-prelude>
↪<!-- (1) -->
    </jsp-property-group>
</jsp-config>
```

Sr. No.	Description
(1)	Based on web.xml settings, when jsp file (~.jsp) is to be read, include.jsp can be read in advance.

Note: Custom tag can also be set in template.jsp. However, it is recommended to create custom tag definition in common jsp include file. For details, refer to *Creating common JSP for include*.

Creating layout

Create jsp (template) that forms frame of a layout and jsp to be embedded in the layout.

- template.jsp

```
<!DOCTYPE html>
<!--[if lt IE 7]> <html class="no-js lt-ie9 lt-ie8 lt-ie7"> <![endif]-->
<!--[if IE 7]>    <html class="no-js lt-ie9 lt-ie8"> <![endif]-->
<!--[if IE 8]>    <html class="no-js lt-ie9"> <![endif]-->
<!--[if gt IE 8]><!-->
<html class="no-js">
<!--<![endif]-->
<head>
<meta charset="utf-8" />
<meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1" />
<meta name="viewport" content="width=device-width" />
<link rel="stylesheet"
    href="${pageContext.request.contextPath}/resources/app/css/styles.css"
    type="text/css" media="screen, projection">
<script type="text/javascript">

</script> <!-- (1) -->
<c:set var="titleKey"> <!-- (2) -->
    <tiles:insertAttribute name="title" ignore="true" />
</c:set>
<title><spring:message code="${titleKey}" text="Create Staff Information" /></
↪title><!-- (3) -->
</head>
<body>
    <div id="header">
        <tiles:insertAttribute name="header" /> <!-- (4) -->
    </div>
    <div id="body">
        <tiles:insertAttribute name="body" /> <!-- (5) -->
    </div>
    <div id="footer">
```

```
        <tiles:insertAttribute name="footer" /> <!-- (6) -->
    </div>
</body>
</html>
```

Sr. No.	Description
(1)	Mention the common contents that need to be described, above step (1).
(2)	Fetch the value of `title` specified in step (6) of tiles-definitions.xml and set it to `titleKey`.
(3)	Set title. When `titleKey` cannot be fetched, display the title defined in text attribute.
(4)	Read the “header” defined in tiles-definitions.xml.
(5)	Read the “body” defined in tiles-definitions.xml.
(6)	Read the “footer” defined in tiles-definitions.xml.

- header.jsp

```
<h1>
    <a href="${pageContext.request.contextPath}">Staff Management
        System</a>
</h1>
```

- createForm.jsp(example of body section)

  The developer is able to focus only on the body section and describe the same without having to mention the extra source code for header and footer.

```
<h2>Create Staff Information</h2>
<table>
    <tr>
        <td>Staff First Name</td>
```

```
        <td><input type="text" /></td>
    </tr>
    <tr>
        <td>Staff Family Name</td>
        <td><input type="text" /></td>
    </tr>
    <tr>
        <td rowspan="5">Staff Authorities</td>
        <td><input type="checkbox" name="sa" value="01" /> Staff
            Management</td>
    </tr>
    <tr>
        <td><input type="checkbox" name="sa" value="02" /> Master
            Management</td>
    </tr>
    <tr>
        <td><input type="checkbox" name="sa" value="03" /> Stock
            Management</td>
    </tr>
    <tr>
        <td><input type="checkbox" name="sa" value="04" /> Order
            Management</td>
    </tr>
    <tr>
        <td><input type="checkbox" name="sa" value="05" /> Show Shopping
            Management</td>
    </tr>
</table>

<input type="submit" value="cancel" />
<input type="submit" value="confirm" />
```

- footer.jsp

```
<p style="text-align: center; background: #e5eCf9;">Copyright &copy;
    20XX CompanyName</p>
```

Note: Refer *Screen footer copyright* for Copyright described in footer.

Creating Controller

While creating Controller, when the request is `<contextPath>/staff/create?form`, perform the settings such that "staff/createForm" is returned from the Controller.

- StaffCreateController.java

```
@RequestMapping(value = "create", method = RequestMethod.GET, params = "form")
public String createForm() {
    return "staff/createForm"; // (1)
}
```

Sr. No.	Description
(1)	With staff as {1} and createForm as {2}, fetch the title name from properties and identify the JSP.

Creating screen

When `<contextPath>/staff/create?form` is called in request, Tiles construct the layout and create screen, as shown below.

```
<definition name="layouts"
    template="/WEB-INF/views/layout/template.jsp"> <!-- (1) -->
    <put-attribute name="header"
        value="/WEB-INF/views/layout/header.jsp" /> <!-- (2) -->
    <put-attribute name="footer"
        value="/WEB-INF/views/layout/footer.jsp" /> <!-- (3) -->
</definition>

<definition name="*/*" extends="layouts">
  <put-attribute name="title" value="title.{1}.{2}" /> <!-- (4) -->
  <put-attribute name="body"
    value="/WEB-INF/views/{1}/{2}.jsp" /> <!-- (5) -->
</definition>
```

Sr. No.	Description
(1)	In case of corresponding request, “layouts” which is a parent layout is called and template is set to /WEB-INF/views/layout/template.jsp.
(2)	WEB-INF/views/layout/header.jsp is set in `header` within the template /WEB-INF/views/layout/template.jsp.
(3)	/WEB-INF/views/layout/footer.jsp is set in `footer` within the template /WEB-INF/views/layout/template.jsp.
(4)	With `title.staff.createForm` as key, fetch the value from properties incorporated in spring-mvc where staff is {1} and createForm is {2}.
(5)	/WEB-INF/views/staff/createForm.jsp is set in `body` within template/WEB-INF/views/layout/template.jsp with staff as {1} and createForm as {2}.

As a result, it is output to the browser by combining header.jsp, createForm.jsp and footer.jsp in the above template.jsp.

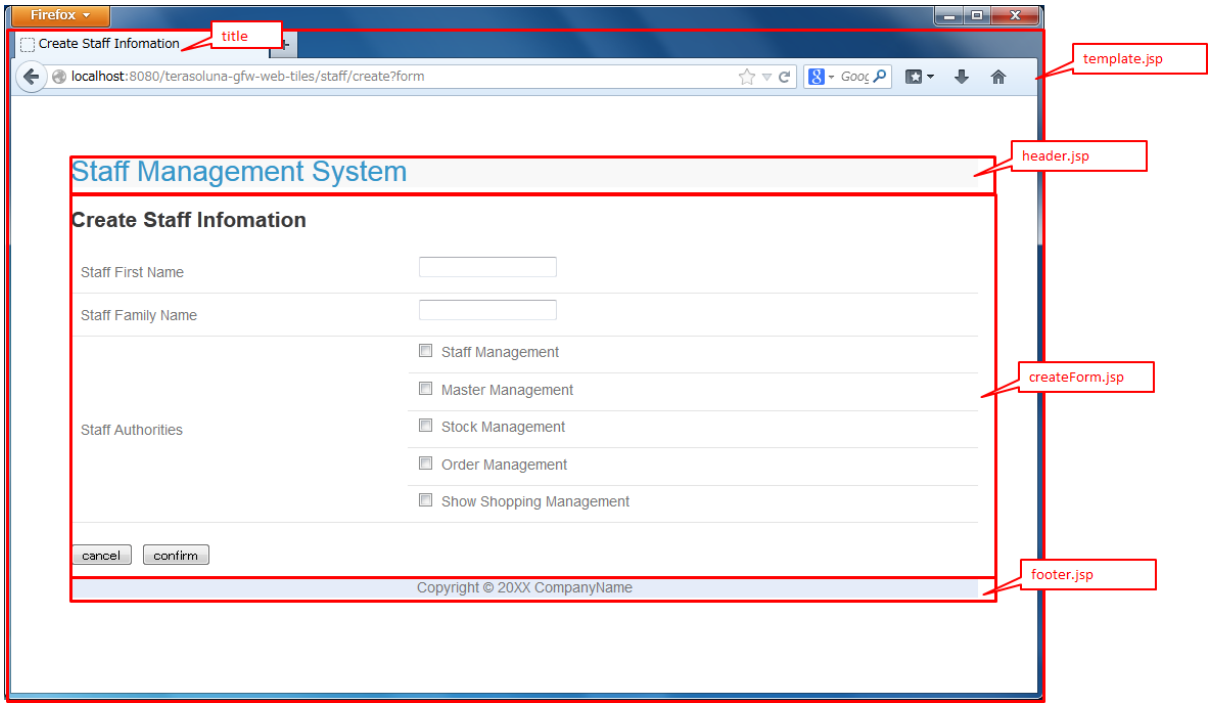
4.11.3 How to extend

Setting multiple layouts

When creating actual business application, display layout may be divided depending on business process contents.

This time, it is assumed that the staff search functionality menu is required to be displayed on left side of the screen.

Configuration is shown below based on *How to use*.



Tiles Definition

- tiles-definitions.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE tiles-definitions PUBLIC
   "-//Apache Software Foundation//DTD Tiles Configuration 3.0//EN"
   "http://tiles.apache.org/dtds/tiles-config_3_0.dtd">

<tiles-definitions>
    <definition name="layoutsOfSearch"
        template="/WEB-INF/views/layout/templateSearch.jsp"> <!-- (1) -->
        <put-attribute name="header"
            value="/WEB-INF/views/layout/header.jsp" />
        <put-attribute name="menu"
            value="/WEB-INF/views/layout/menu.jsp" />
        <put-attribute name="footer"
            value="/WEB-INF/views/layout/footer.jsp" />
    </definition>

    <definition name="*/search*" extends="layoutsOfSearch"> <!-- (2) -->
        <put-attribute name="title" value="title.{1}.search{2}" /> <!-- (3) -
↪->
        <put-attribute name="body" value="/WEB-INF/views/{1}/search{2}.jsp" /
↪> <!-- (4) -->
    </definition>

    <definition name="layouts"
        template="/WEB-INF/views/layout/template.jsp">
```

```
        <put-attribute name="header"
            value="/WEB-INF/views/layout/header.jsp" />
        <put-attribute name="footer"
            value="/WEB-INF/views/layout/footer.jsp" />
    </definition>

    <definition name="*/*" extends="layouts">
        <put-attribute name="title" value="title.{1}.{2}" />
        <put-attribute name="body" value="/WEB-INF/views/{1}/{2}.jsp" />
    </definition>
</tiles-definitions>
```

Sr. No.	Description
(1)	Define the parent layout structure to be added. When using a different layout, ensure that name attribute of definition tag does not duplicate with the existing layout definition i.e. “layouts”.
(2)	Layout definition called when the layout to be added is same as the name pattern at the time of ‘create’ request. This layout definition is read when the request corresponds to <contextPath>/*/search*. Extended layout definition “layoutsOfSearch” is also applied.
(3)	Specify the title to be used in the layout to be added. Fetch the value from properties incorporated in spring-mvc. (In the following description, it is set to application-messages.properties.) {1} is the 1st “*” of “*/search*” of request. It is necessary that {2} starts with “search” as it corresponds to the “search*” of “*/search*” request.
(4)	Place the jsp file in which the body is defined such that, the request path matches with {1} and JSP file name beginning with “search”, matches with {2}. The value of ‘value’ attribute needs to be changed according to the configuration of JSP file location.

Note: When multiple requests correspond to name attribute patterns of definition tag, the verification is

done sequentially from the top and the very first pattern that matches with the request is applied. In the above case, as the request for staff search screen corresponds to multiple patterns, the layout is defined at the top.

- *application-messages.properties*

```
title.staff.createForm = Create Staff Information
title.staff.searchStaff = Search Staff Information # (1)
```

Sr. No.	Description
(1)	Message to be added. "staff" is the 1st "*" of "*/search*" request. As "searchStaff" corresponds to "search*" part of "*/search*" request, it is necessary that it begins with "search".

Creating layout

Create the jsp (template) that forms the frame of the layout and jsp to be embedded in layout.

- templateSearch.jsp

```
<!DOCTYPE html>
<!--[if lt IE 7]> <html class="no-js lt-ie9 lt-ie8 lt-ie7"> <![endif]-->
<!--[if IE 7]>    <html class="no-js lt-ie9 lt-ie8"> <![endif]-->
<!--[if IE 8]>    <html class="no-js lt-ie9"> <![endif]-->
<!--[if gt IE 8]><!-->
<html class="no-js">
<!--<![endif]-->
<head>
<meta charset="utf-8" />
<meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1" />
<meta name="viewport" content="width=device-width" />
<link rel="stylesheet"
    href="${pageContext.request.contextPath}/resources/app/css/styles.css"
    type="text/css" media="screen, projection">
<script type="text/javascript">

</script>
<c:set var="titleKey">
    <tiles:insertAttribute name="title" ignore="true" />
</c:set>
<title><spring:message code="${titleKey}" text="Search Staff Information" /></
↪title>
</head>
<body>
    <div id="header">
```

```
        <tiles:insertAttribute name="header" />
    </div>
    <div id="menu">
        <tiles:insertAttribute name="menu" /> <!-- (1) -->
    </div>
    <div id="body">
        <tiles:insertAttribute name="body" />
    </div>
    <div id="footer">
        <tiles:insertAttribute name="footer" />
    </div>
</body>
</html>
```

Sr. No.	Description
(1)	Read the “menu” defined in tiles-definitions.xml. Rest is same as *How to use*.

- styles.css

```
div#menu { /* (1) */
    float: left;
    width: 20%;
}

div#searchBody { /* (2) */
    float: right;
    width: 80%;
}

div#footer { /* (3) */
    clear: both;
}
```

Sr. No.	Description
(1)	Set the Menu style. Here, Menu Screen is left aligned using float:left and is displayed with 20% width.
(2)	Set the Body style. Here, the Business Screen is right aligned using float:right and displayed with 80% width. Name is specified as searchBody. This is because duplication in existing layout and name can have an impact on the existing layout style.
(3)	Set the Footer style. Float effect of menu and body is initialized. By this, the footer is displayed below menu and body.

- header.jsp

  Same as *How to use*.

- menu.jsp

```
<table>
    <tr>
        <td><a href="${pageContext.request.contextPath}/staff/create?form">
↪Create Staff Information</a></td>
    </tr>
    <tr>
        <td><a href="${pageContext.request.contextPath}/staff/search">Search␣
↪Staff Information</a></td>
    </tr>
</table>
```

- searchStaff.jsp (example of body section)

```
<h2>Search Staff Information</h2>
<table>
    <tr>
        <td>Staff First Name</td>
        <td><input type="text" /></td>
    </tr>
    <tr>
        <td>Staff Family Name</td>
        <td><input type="text" /></td>
```

```
    </tr>
    <tr>
        <td rowspan="5">Staff Authorities</td>
        <td><input type="checkbox" name="sa" value="01" /> Staff
            Management</td>
    </tr>
    <tr>
        <td><input type="checkbox" name="sa" value="02" /> Master
            Management</td>
    </tr>
    <tr>
        <td><input type="checkbox" name="sa" value="03" /> Stock
            Management</td>
    </tr>
    <tr>
        <td><input type="checkbox" name="sa" value="04" /> Order
            Management</td>
    </tr>
    <tr>
        <td><input type="checkbox" name="sa" value="05" /> Show Shopping
            Management</td>
    </tr>
</table>

<input type="submit" value="Search" />
```

- footer.jsp

  Same as *How to use*.

Creating Controller

While creating Controller, if the request is `<contextPath>/staff/search`, set such that "staff/searchStaff" is returned from the Controller.

- StaffSearchController.java

```
@RequestMapping(value = "search", method = RequestMethod.GET)
public String createForm() {
    return "staff/searchStaff"; // (1)
}
```

Sr. No.	Description
(1)	With staff as {1} and searchStaff as {2}, fetch the title name from properties and identify the JSP.

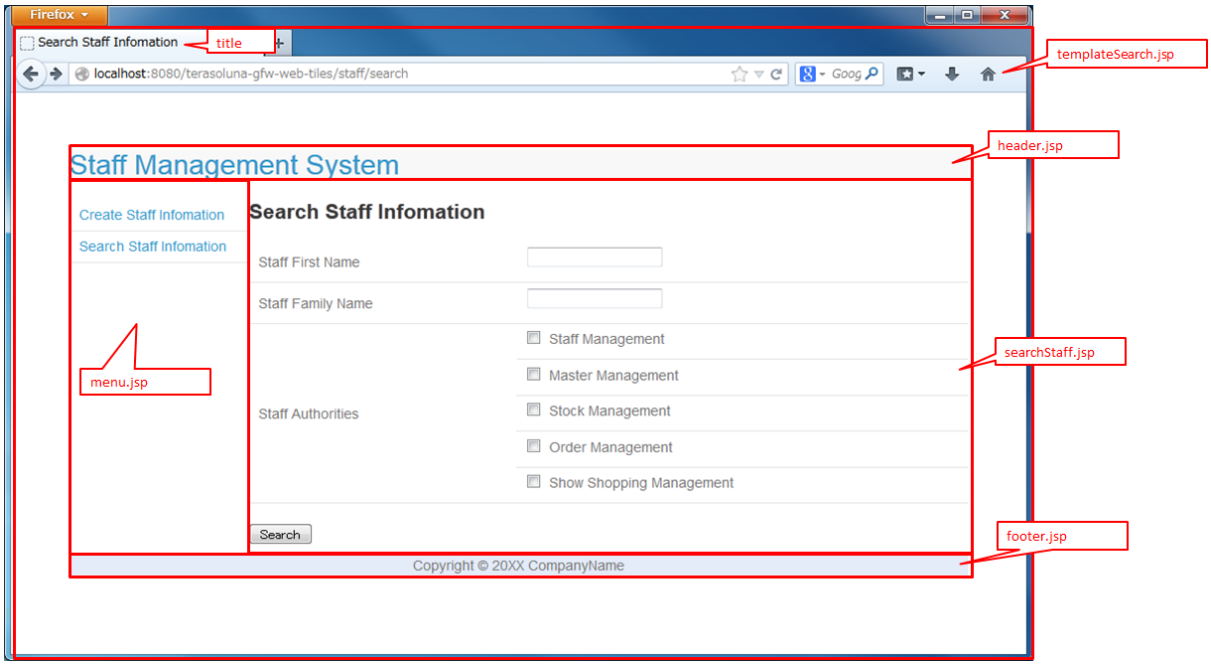
Creating screen

When <contextPath>/staff/search is called in request, screen is generated through another layout as shown below.

```
<definition name="layoutsOfSearch"
    template="/WEB-INF/views/layout/templateSearch.jsp"> <!-- (1) -->
    <put-attribute name="header"
        value="/WEB-INF/views/layout/header.jsp" /> <!-- (2) -->
    <put-attribute name="menu"
        value="/WEB-INF/views/layout/menu.jsp" /> <!-- (3) -->
    <put-attribute name="footer"
        value="/WEB-INF/views/layout/footer.jsp" /> <!-- (4) -->
</definition>

<definition name="*/search*" extends="layoutsOfSearch"> <!-- (5) -->
    <put-attribute name="title" value="title.{1}.search{2}" /> <!-- (6) -->
    <put-attribute name="body" value="/WEB-INF/views/{1}/search{2}.jsp" /> <!-
↪- (7) -->
</definition>
```

Sr. No.	Description
(1)	In case of a corresponding request, “layoutsOfSearch” which is a parent layout is called and template is set in /WEB-INF/views/layout/templateSearch.jsp.
(2)	WEB-INF/views/layout/header.jsp is set in `header` within the template /WEB-INF/views/layout/templateSearch.jsp.
(3)	/WEB-INF/views/layout/menu.jsp is set in `menu` within the template /WEB-INF/views/layout/templateSearch.jsp.
(4)	/WEB-INF/views/layout/footer.jsp is set in `footer` within the template /WEB-INF/views/layout/templateSearch.jsp.
(5)	This layout definition is read when the request corresponds to <contextPath>/*/search*. In that case, “layoutsOfSearch” which is a parent layout is also read.
(6)	With `title.staff.searchStaff` as key, fetch the value from properties incorporated in spring-mvc, where staff is {1} and searchStaff is “search{2}”.
(7)	/WEB-INF/views/staff/searchStaff.jsp is set in `body` within the template/WEB-INF/views/layout/templateSearch.jsp where staff is {1} and searchStaff is “search{2}”.

As a result, it is output to the browser by combining header.jsp, menu.jsp, searchStaff.jsp and footer.jsp in the above templateSearch.jsp file.



4.12 JSP Tag Libraries and EL Functions offered by common library

4.12.1 Overview

Below are the JSP Tag Libraries and EL Functions offered by common library as an ability to support the JSP implementation.

JSP Tag Library

Below are the JSP Tag Libraries offered by common library.

Sr.No	Tag name	Overview
1.	*<t:pagination>*	Outputs the pagination link.
2.	*<t:messagesPanel>*	Outputs the result message.
3.	*<t:transaction>*	Outputs the transaction token as hidden item.

EL Functions

Below are the EL Functions offered by common library.

XSS counter measures

Sr.No	Function Name	Overview
1.	*f:h()*	Converts the specified object into a string and escape the HTML special characters from converted string.
2.	*f:js()*	Escape the JavaScript special characters from the specified string.
3.	*f:hjs()*	After escaping the JavaScript special characters from the specified string, escape the HTML special characters. (short function of `f:h(f:js())`)

URL related

Sr.No	Function Name	Overview
4.	*f:query()*	Generates UTF-8 URL encoded query string from the specified object.
5.	*f:u()*	Performs UTF-8 URL encoding on specified string.

DOM related

Sr.No	Function Name	Overview
6.	*f:link()*	Generates a hyperlink (`<a>` tag) for jumping to specified URL.
7.	*f:br()*	Converts the new line character into ` ` tag from the specified string.

Utility

Sr.No	Function Name	Overview
8.	*f:cut()*	Extracts specified number of characters from the specified string.

4.12.2 How to use

The use of JSP Tag Library and EL function offered by common library explained below. The appropriate Hyperlink is placed at appropriate location if detail description explained in other chapters.

<t:pagination>

The `<t:pagination>` tag is a JSP Tag Library to output the pagination link by referring the information stored in page search results (`org.springframework.data.domain.Page`).

For detail description of pagination and how to use this tag, Refer the following section [*Pagination*].

- For pagination link, [*Display of pagination link*]
- For parameter values of this tag, [*Parameters of JSP tag library*]
- For basic implementation of the JSP using this tag, [*Display of Pagination link*]
- For the layout of how to change the pagination link, [*Implementation of JSP (layout change)*]

<t:messagesPanel>

The `<t:messagesPanel>` tag is a JSP Tag Library to output the processing result message, (Such as `org.terasoluna.gfw.common.message.ResultMessage` or message having exception).

Refer the following section [*Message Management*] for how to use this tag.

- For how to display messages using this tag, [*Display of result messages*]
- For parameter values of this tag, [*Changing attribute of <t:messagesPanel> tag*]

<t:transaction>

The `<t:transaction>` tag is a JSP Tag Library to output the transaction token as hidden item (`<input type="hidden">"`).

Refer the following section [*Double Submit Protection*] for the transaction token check feature and how to use this tag.

- For transaction token check feature, [*Using transaction token check*]
- For how to use this tag, [*How to use transaction token check in View (JSP)*]

Note: This tag is used for sending a transaction token to the server while using standard HTML `<form>` tag.

No need to use this tag if spring framework offered `<form:form>` tag (JSP Tag Library) has been used because `org.terasoluna.gfw.web.token.transaction.TransactionTokenRequestDataValueProcessor` offered by the common library has been already mechanized to handle a transaction token.

---

f:h()

The `f:h()` is an EL Function which converts the specified object into a string and escape the HTML special characters from converted string.

Refer [*Output Escaping*] for the specification of HTML special characters and escaping.

f:h() function specification

Argument

Sr.No	Type	Description
1.	`java.lang.Object`	Object that contain HTML special characters

Note: Specified objects,

- In case of array, `java.util.Arrays#toString` method
- In case of not array, `toString` method of specified object

is called for the string conversions.

Return value

Sr.No	Type	Description
1.	`java.lang.String`	String after HTML escaping If the object specified in argument is `null` , returns the empty string(`""`).

f:h() how to use

For the information about how to use `f:h()` function, refer [*Example of escaping output value using f:h() function*].

f:js()

The `f:js()` is an EL Function which escape the JavaScript special characters from the specified string argument.

Refer [*JavaScript Escaping*] for the specification of JavaScript special characters and escaping.

f:js() function specification

Argument

Sr.No	Type	Description
1.	`java.lang.String`	String that contain JavaScript special characters

Return value

Sr.No	Type	Description
1.	`java.lang.String`	String after JavaScript escaping If the string specified in argument is `null` , it returns the empty string(`""`).

f:js() how to use

For the information about how to use `f:js()` function, refer [*Example of escaping output value using f:js() function*].

f:hjs()

The `f:hjs()` is an EL Function which escapes the HTML special characters after escaping the JavaScript special characters from the specified string argument, (short function of `f:h(f:js())`).

- For how to use, refer [*Event handler Escaping*].
- Refer [*JavaScript Escaping*] for the specification of JavaScript special characters and escaping.
- Refer [*Output Escaping*] for the specification of HTML special characters and escaping.

f:hjs() function specification

Argument

Sr.No	Type	Description
1.	`java.lang.String`	String that contain HTML special characters or JavaScript special characters

Return value

Sr.No	Type	Description
1.	`java.lang.String`	String after JavaScript and HTML escaping. If the string specified in argument is `null` , it returns the empty string(`""`).

f:hjs() how to use

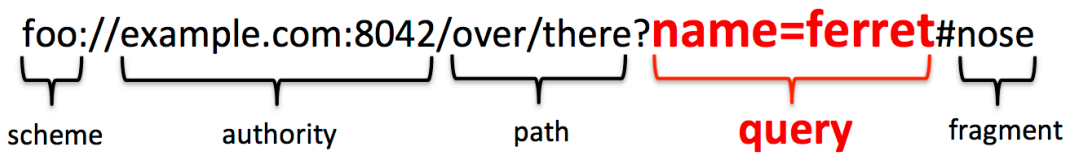
For the information about how to use `f:hjs()` function, refer [*Example of escaping output value using f:hjs() function*].

f:query()

The `f:query()` is an EL Function which generates the query string from java.util.Map object or JavaBean (form object) that is specified in the argument. Parameter names and parameter values in the query string are URL encoded in UTF-8.

URL encoding specification explained below.

In this function, the parameter name and parameter value of the query string are encoded on RFC 3986 basis. In RFC 3986, the part of query string is defined as follows.



- query = *(pchar / "/" / "?")
- pchar = unreserved / pct-encoded / sub-delims / ":" / "@"
- unreserved = ALPHA / DIGIT / "-" / "." / "_" / "~"
- sub-delims = "!" / "$" / "&" / "'" / "(" / ")" / "*" / "+" / "," / ";" / "="
- pct-encoded = "%" HEXDIG HEXDIG

In this function, one of the character that can be used as a query string,

- "=" (Separator character of the parameter name and parameter value)
- "&" (Separator character when dealing with multiple parameters)
- "+" (Character that represent a space when you submit HTML Form)

are encoded in the pct-encoded formatting string.

f:query() function specification

Argument

Sr.No	Type	Description
1.	`java.lang.Object`	Object from which the query string generated (JavaBean or `Map`) Property name will be a request parameter name if you have specified a JavaBean and key name will be a request parameter name if you specified the `Map`. Supported value types of JavaBean's property and value of `Map` are as follows: • Classes which implements `Iterable` interface • Array • Classes which implements `Map` interface • JavaBean • Simple Types (classes that can converted to `String` type the using `DefaultFormattingConversionService`) From the terasoluna-gfw-web 5.0.1.RELEASE, it has been improved to be able specify a nested structured JavaBean or `Map`.

Note: A simple type property value of the specified object is converted into string using the convert method of `org.springframework.format.support.DefaultFormattingConversionService`. Refer Spring Framework Reference Documentation(Spring Type Conversion)for the `ConversionService`.

Return value

Sr.No	Type	Description
1.	`java.lang.String`	Query string that is generated based on the specified object in the argument (URL encoded string in UTF-8) If the object specified in argument is other than the JavaBean or `Map`, it returns the empty string(`""`).

Note: Rules for conversion to the query string

`f:query()` converts an object so that the Spring Web MVC can handle it at binding process provided.

[Request parameter name]

Conditions	Conversion specification of parameter name	Conversion examples
Case that property type is an instance of `Iterable`	Property name + `[element position]`	`status[0]=accepting`
Case that property type is an instance of `Iterable` or Array and the value of the element is empty	Property name (Does not append `[element position]`)	`status=`
Case that property type is an instance of `Map`	Property name + `[Map's key name]`	`status[accepting]=AcceptOrder`
Case that property type(including element type in `Iterable`,Array and `Map`) is JavaBean	Value that combined a property name with `"."`(dot)	`mainContract.name=xxx` `subContracts[0].name=xxx`
Case that property type is a simple type	Property name	`userId=xxx`

Case that property value is `null`	_(underscore) + Property name	`_mainContract.name=`
		`_status[0]=`
		`_status[accepting]=`

[Request parameter value]

Conditions	Conversion specification of parameter value	Conversion examples
Case that property value is `null`	Empty string	`_userId=`
Case that property type is an instance of `Iterable` or Array and the value of the element is empty	Empty string	`status=`
Case that property value is not `null`	Value that can be converted to `String` type using the `DefaultFormattingConversionService`	`targetDate=20150801`

f:query() how to use

For the information about how to use `f:query()` function, refer [*Carrying forward search conditions using page link*]. Here, this function is used as to carry forward the search criteria while switching the pages using the pagination link. Further, function description and the specification also described here and that should be read.

f:u()

The `f:u()` is an EL Function which performs UTF-8 URL encoding on specified string argument.

This function is provided for performing URL encoding on those values which are going to be set as parameter values in the query string. For the URL encoding specification, refer [*f:query()*]

f:u() function specification

Argument

Sr.No	Type	Description
1.	`java.lang.String`	String that contain URL encoding required characters

Return value

Sr.No	Type	Description
1.	`java.lang.String`	String after URL encoding If the string specified in argument is `null` , it returns the empty string("").

f:u() how to use

```
<div id="url">
    <a href="https://search.yahoo.com/search?p=${f:u(bean.searchString)}">  <!--␣
↪(1) -->
        Go to Yahoo Search
    </a>
</div>
```

Sr.No	Description
(1)	In the above example, sets the URL-encoded value to the request parameters of the search site using this function.

f:link()

The `f:link()` is an EL Function which generates a hyperlink (<a> tag) for jumping to specified URL which is specified in the argument.

> **Warning:** Please note that, this function is not going to escaping the special characters nor performing the URL encoding.

f:link() function specification

Argument

Sr.No	Type	Description
1.	`java.lang.String`	Link of the URL string URL string should be HTTP or HTTPS schema of the URL format. (e.g : `http://hostname:80/terasoluna/global.ex?id=123`)

Return value

Sr.No	Type	Description
1.	`java.lang.String`	Generated Hyper link (<a> tag) based on string specified in the argument The string specified in arguments, • If the string specified in argument is `null` , the empty string(`""`) • If the string is not in the URL format of HTTP or HTTPS schema, input string without generating a hyperlink is return.

f:link() how to use

Implementation

```
<div id="link">
    ${f:link(bean.httpUrl)}  <!-- (1) -->
</div>
```

Output

```
<div id="link">
    <a href="http://terasoluna.org/">http://terasoluna.org/</a>  <!-- (2) -->
</div>
```

Sr.No	Description
(1)	Generated Hyper link from the URL string specified in the argument.
(2)	URL string specified in the argument set in the `href` attribute of `<a>` tag and link name of the hyper link.

Warning: When adding the request parameters to the URL, the value of the request parameters should be URL encoded. When adding the request parameters, the value of the request parameters should be URL encoded using appropriate `f:query()` function or `f:u()` function.

In addition, it has been described in the return value description, if the format of the URL string specified in the argument is not appropriate, it returns the input string value without generating a hyperlink. Therefore, if you want to use the input value from the user as a URL string in the argument, similar to string output process, the escaping process of the HTML special characters (*XSS Countermeasures*) are required.

f:br()

The `f:br()` is an EL Function which converts the new line character (`CRLF`, `LF`, `CR`) specified in the argument into `<br />` tag.

Tip: If you want to display a string containing new line code as a newline on browser, it is necessary to convert the new line code into ‘‘
 ‘‘ tag.

For example, if you want to display the string entered in the textarea (`<textarea>`) of the input screen as it is

on the confirmation screen or completion screen, it is advisable to use this function.

f:br() function specification

Argument

Sr.No	Type	Description
1.	`java.lang.String`	String that contain new line code

Return value

Sr.No	Type	Description
1.	`java.lang.String`	String after conversion If the string specified in argument is `null` , it returns the empty string(`""`).

f:br() how to use

```
<div id="text">
    ${f:br(f:h(bean.text))}">  <!-- (1) -->
</div>
```

Sr.No	Description
(1)	The newline displays on the browser by converting the new line character into ` ` tag from the specified string argument.

Note: When you display a string on the screen, there is a need to escape the HTML special character as [*XSS Countermeasures*].

if you are converting new line code into `<br />` tag using `f:br()` function, as in the above example, a string that has escaped the HTML special characters need to pass as an argument to `f:br()` function.

The string obtained by converting new line code into `<br />` tag using `f:br()` function passes as an argument to the `f:h()` function, the letter `"<br />"` get displayed on the browser hence be careful in order to call the function.

f:cut()

The `f:cut()` is an EL Function which extracts specified number of characters from the specified string.

f:cut() function specification

Argument

Sr.No	Type	Description
1.	`java.lang.String`	String from which extraction is done
2.	`int`	The number of characters that can extracted

Return value

Sr.No	Type	Description
1.	`java.lang.String`	The extracted string (String part that exceeds the specified number of characters has been destroyed) If the string specified in argument is `null` , it returns the empty string(`""`).

f:cut() how to use

```
<div id="cut">
    ${f:h(f:cut(bean.originText, 5))}  <!-- (1) -->
</div>
```

Sr.No	Description
(1)	In the above example, you can extract the first five characters of the string that was specified in the argument and displays on the screen.

Note: There is a need to escape the HTML special character as [*XSS Countermeasures*] while displaying the extracted string on the screen. In the above example, string is escaped by using `f:h()` function.

4.13 Ajax

4.13.1 Overview

This section explains how to implement applications that use Ajax.

Todo

TBD

Details regarding client side implementation etc. will follow in subsequent versions.

Ajax is the generic term used for group of techniques that perform the following asynchronous processes.

- Screen operations performed on the browser
- HTTP communication with the server triggered by a screen operation and reflecting back the communication results to the user interface

Ajax is often used to improve usability. This is because, screen operations can be continued during HTTP communication.
Typical examples of Ajax are (a) Providing suggestions while searching words and (b) Real time search of a search site.

4.13.2 How to use

Application settings

Application settings for Ajax are explained below.

Warning: DoS attack measures at the time of StAX(Streaming API for XML) use

If the StAX is used to parse the XML format data, protect DoS attack. For details, refer to CVE-2015-3192 - DoS Attack with XML Input.

Settings to enable the Ajax functionality in Spring MVC

Content-Type (such as "application/xml", "application/json" etc.) used in Ajax communication is set such that it can be handled by the handler method of Controller.

- spring-mvc.xml

```
<mvc:annotation-driven /> <!-- (1) -->
```

Sr. No.	Description
(1)	If <mvc:annotation-driven> element is specified, the functionality required for Ajax communication is enabled. Therefore, special settings are not necessary for Ajax communication.

Note: Functionalities required for Ajax communication specifically refer to the ones provided by org.springframework.http.converter.HttpMessageConverter class.

HttpMessageConverter performs the following roles.

- Creating Java object from data stored in the request body.
- Creating the data to be written to the response Body from Java object.

The HttpMessageConverter which is enabled by default on specifying <mvc:annotation-driven>, is as follows.

Sr. No.	Class name	Target Format	Description
1.	org.springframework.http.converter.json. MappingJackson2HttpMessageConverter	JSON	`HttpMessageConverter` to handle JSON as request body or response body. Jackson system is included in the blank project. Hence, it can be used in its default state.
2.	org.springframework.http.converter.xml. Jaxb2RootElementHttpMessageConverter	XML	`HttpMessageConverter` to handle XML as request body or response body. JAXB2.0 is included as standard from JavaSE6. Hence it can be used in its default state.

Note: Notice If you change from jackson version 1.xx to jackson version 2.xx, refer to *here*.

Warning: XXE (XML External Entity) Injection measures

When handling XML format data in Ajax communication, it is necessary to implement XXE(XML External Entity) Injection measure. Subsequent versions above terasoluna-gfw-web 1.0.1.RELEASE are Spring MVC (above 3.2.10.RELEASE) version dependent. As these Spring MVC versions implement XXE Injection measures, it is not necessary to implement them independently.

When using terasoluna-gfw-web 1.0.0.RELEASE, since it is dependent on the Spring MVC version (3.2.4.RELEASE) that does not implement XXE Injection, a class provided by Spring-oxm should be used.

- `spring-mvc.xml`

```
<!-- (1) -->
<bean id="xmlMarshaller" class="org.springframework.oxm.jaxb.
↪Jaxb2Marshaller">
    <property name="packagesToScan" value="com.examples.app" /> <!--␣
↪(2) -->
</bean>

<!-- ... -->

<mvc:annotation-driven>

    <mvc:message-converters>
        <!-- (3) -->
        <bean class="org.springframework.http.converter.xml.
↪MarshallingHttpMessageConverter">
            <property name="marshaller" ref="xmlMarshaller" /> <!--␣
↪(4) -->
            <property name="unmarshaller" ref="xmlMarshaller" /> <!--
↪ (5) -->
        </bean>
    </mvc:message-converters>

    <!-- ... -->

</mvc:annotation-driven>

<!-- ... -->
```

Sr. No.	Description
(1)	Perform the bean definition of `Jaxb2Marshaller` provided by Spring-oxm. `Jaxb2Marshaller` implements the XXE Injection measures in default state.
(2)	Specify the package name where the JAXB JavaBean (JavaBean assigned with `javax.xml.bind.annotation.XmlRootElement` annotation) is stored in the `packagesToScan` property. JAXB JavaBean stored under the specified package is scanned and registered for marshalling or unmarshalling the JavaBean. It is scanned in the same way as the base-package attribute of `<context:component-scan>`.

(3)	Add bean definition of `MarshallingHttpMessageConverter` to the `<mvc:message-converters>` element that is the child element of `<mvc:annotation-driven>`.
(4)	Specify the bean of `Jaxb2Marshaller` defined in (1) in `marshaller` property.
(5)	Specify the bean of `Jaxb2Marshaller` defined in (1) in `unmarshaller` property.

Adding Spring-oxm as dependent artifact.

- pom.xml

```
<!-- omitted -->

<!-- (1) -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-oxm</artifactId>
    <version>${org.springframework-version}</version> <!-- (2) -->
</dependency>

<!-- omitted -->
```

Sr. No.	Description
(1)	Add Spring-oxm as dependent artifact.
(2)	Spring version should be fetched from the placeholder (${org.springframework-version}) that controls the Spring version number defined in pom.xml of terasoluna-gfw-parent.

Implementing Controller

Prerequisites for the sample code explained hereafter, are as follows.

- Response data should be in JSON format.
- JQuery should be used at client side. It should be the latest version of 1.x series (1.10.2), which is used while writing this document.

Warning: Measures to circular reference

When you serialize a JavaBean in JSON or XML format using `HttpMessageConverter` and if property holds an object of cross reference relationship, the `StackOverflowError` and `OutOfMemoryError` occur due to circular reference, hence it is necessary to exercise caution.

In order to avoid a circular reference,

- `@com.fasterxml.jackson.annotation.JsonIgnore` annotation to exclude the property from serialization in case of serialized in JSON format using the Jackson
- `@javax.xml.bind.annotation.XmlTransient` annotation to exclude the property from serialization in case of serialized in XML format using the JAXB

can be added.

Below is the example of how to exclude specific field from serialization while serializing in JSON format using the Jackson.

```
public class Order {
    private String orderId;
    private List<OrderLine> orderLines;
    // ...
}
```

```
public class OrderLine {
    @JsonIgnore
    private Order order;
    private String itemCode;
    private int quantity;
    // ...
}
```

Sr. No.	Description
(1)	The `@JsonIgnore`annotation is added to exclude the property from serialization.

Fetching data

How to fetch data using Ajax is explained here.

Following example serves as the Ajax communication that returns a list matching with the search word.

- JavaBean for receiving request data

```
// (1)
public class SearchCriteria implements Serializable {

    // omitted

    private String freeWord; // (2)

    // omitted setter/getter

}
```

Sr. No.	Description
(1)	Create the JavaBean that receives request data.
(2)	Match property name with parameter name of request parameter.

- JavaBean for storing the data to be returned

```
// (3)
public class SearchResult implements Serializable {

    // omitted

    private List<XxxEntity> list;

    // omitted setter/getter

}
```

Sr. No.	Description
(3)	Create the JavaBean for storing the data to be returned.

- Controller

```
@RequestMapping(value = "search", method = RequestMethod.GET) // (4)
@ResponseBody // (5)
public SearchResult search(@Validated SearchCriteria criteria) { // (6)

    SearchResult searchResult = new SearchResult(); // (7)

    // (8)
    // omitted

    return searchResult; // (9)
}
```

Sr. No.	Description
(4)	Specify `RequestMethod.GET` in the method attribute of `@RequestMapping` annotation.
(5)	Assign `@org.springframework.web.bind.annotation.ResponseBody` annotation. By assigning this annotation, the returned object is marshalled in JSON format and set in response body.
(6)	Specify the JavaBean that receives request data, as an argument. If input validation is required, specify `@Validated`. For error handling of input validation, refer to "*Input error handling*". For details on input validation, refer to "*Input Validation*".
(7)	Create the JavaBean object to store the data to be returned.
(8)	Search data and store the search result in the object created in (7). In the above example, implementation is omitted.
(9)	Return the object to be marshalled in response body.

- HTML(JSP)

```
<!-- omitted -->

<meta name="contextPath" content="${pageContext.request.contextPath}" />

<!-- omitted -->
```

```
<!-- (10)  -->
<form id="searchForm">
  <input name="freeWord" type="text">
  <button onclick="return searchByFreeWord()">Search</button>
</form>
```

Sr. No.	Description
(10)	Form to enter the search condition. In the above example, it has a text box to enter the search condition and a search button.

```
<!-- (11) -->
<script type="text/javascript"
    src="${pageContext.request.contextPath}/resources/vendor/jquery/jquery-1.
↪10.2.js">
</script>
```

Sr. No.	Description
(11)	Read the JQuery JavaScript file. In the above example, request is sent to the `/resources/vendor/jquery/jquery-1.10.2.js` path, to read the JQuery JavaScript file.

Note: Refer to the settings below to read JQuery JavaScript file. Setting values provided in the blank project are as follows.

- `spring-mvc.xml`

```
<!-- (12) -->
<mvc:resources mapping="/resources/**"
    location="/resources/,classpath:META-INF/resources/"
    cache-period="#{60 * 60}" />
```

Sr. No.	Description
(12)	Settings for releasing resource files (JavaScript files, Stylesheet files, image files etc.). In the above setting example, when there is a request for path starting with `/resources/`, the files in `/resources/` directory of war file or the `/META-INF/resources/` directory of class path are sent as a response.

In the above settings, the JQuery JavaScript file needs to be placed under any one of the following paths.

- `/resources/vendor/jquery/jquery-1.10.2.js` in war file

  It is `src/main/webapp/resources/vendor/jquery/jquery-1.10.2.js` when indicated by the path in the project.

- `/META-INF/resources/vendor/jquery/jquery-1.10.2.js` in class path

  It is `src/main/resources/META-INF/resources/vendor/jquery/jquery-1.10.2.js` when indicated by the path in the project.

- JavaScript

```
var contextPath = $("meta[name='contextPath']").attr("content");

// (13)
function searchByFreeWord() {
    $.ajax(contextPath + "/ajax/search", {
        type : "GET",
        data : $("#searchForm").serialize(),
        dataType : "json", // (14)

    }).done(function(json) {
        // (15)
        // render search result
        // omitted
```

```
    }).fail(function(xhr) {
        // (16)
        // render error message
        // omitted

    });
    return false;
}
```

Sr. No.	Description
(13)	Ajax function that converts search criteria specified in the form to request parameter and sends the request for */ajax/search* using GET method. In the above example, clicking the button acts as the trigger for Ajax communication. However, by setting key down or key up of text box as the trigger, real time search can be performed.
(14)	Specify the data format to be received as a response. In the above example, as `"json"` is specified, `"application/json"` is set in Accept header.
(15)	Implement the process when Ajax communication ends normally (when Http status code is `"200"`). In the above example, implementation is omitted.
(16)	Implement the process when Ajax communication does not end normally (when Http status code is `"4xx"` and `"5xx"`). In the above example, implementation is omitted. For error process implementation, refer to *Posting form data*.

Tip: In the above example, by setting context path (`${pageContext.request.contextPath}`) of Web application in HTML‘‘<meta>‘‘ element. JSP code is deleted from JavaScript code.

Communication is as follows when “Search” button of Search form is clicked.

Main points are highlighted.

- Request data

```
GET /terasoluna-gfw-web-blank/ajax/search?freeWord= HTTP/1.1
Host: localhost:9999
Connection: keep-alive
Accept: application/json, text/javascript, */*; q=0.01
X-Requested-With: XMLHttpRequest
User-Agent: Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like␣
↪Gecko) Chrome/30.0.1599.101 Safari/537.36
Referer: http://localhost:9999/terasoluna-gfw-web-blank/ajax/xxe
Accept-Encoding: gzip,deflate,sdch
Accept-Language: en-US,en;q=0.8,ja;q=0.6
Cookie: JSESSIONID=3A486604D7DEE62032BA6C073FC6BE9F
```

- Response data

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
X-Track: a8fb8fefaaf64ee2bffc2b0f77050226
Content-Type: application/json;charset=UTF-8
Transfer-Encoding: chunked
Date: Fri, 25 Oct 2013 13:52:55 GMT

{"list":[]}
```

Posting form data

How to post form data and fetch processing result using Ajax, is explained here.

Following example is about the Ajax communication of receiving two numbers and returning the calculation result.

- JavaBean to receive form data

```
// (1)
public class CalculationParameters implements Serializable {

    // omitted

    private Integer number1;

    private Integer number2;

    // omitted setter/getter

}
```

Sr. No.	Description
(1)	Create the JavaBean for receiving form data.

- JavaBean that stores processing result

```
// (2)
public class CalculationResult implements Serializable {

    // omitted

    private int resultNumber;

    // omitted setter/getter

}
```

Sr. No.	Description
(2)	Create the JavaBean that stores processing result.

- Controller

```
@RequestMapping("xxx")
@Controller
public class XxxController {

    @RequestMapping(value = "plusForForm", method = RequestMethod.POST) // (3)
    @ResponseBody
    public CalculationResult plusForForm(
        @Validated CalculationParameters params) { // (4)
        CalculationResult result = new CalculationResult();
        int sum = params.getNumber1() + params.getNumber2();
        result.setResultNumber(sum); // (5)
        return result; // (6)
    }

    // omitted

}
```

Sr. No.	Description
(3)	Specify `RequestMethod.POST` in the method attribute of `@RequestMapping` annotation.
(4)	Specify the JavaBean for receiving form data as an argument. Specify `@Validated` when input validation is required. For handling input validation errors, refer to "*Input error handling*". For details on input validation, refer to "*Input Validation*".
(5)	Store the processing result in the object created for the same. In the above example, calculation result of the two numbers fetched from form object, is stored.
(6)	Return the object to perform marshalling in response body.

- HTML (JSP)

```
<!-- omitted -->

<meta name="contextPath" content="${pageContext.request.contextPath}" />

<sec:csrfMetaTags />

<!-- omitted -->

<!-- (7)  -->
<form id="calculationForm">
    <input name="number1" type="text">+
    <input name="number2" type="text">
    <button onclick="return plus()">=</button>
    <span id="calculationResult"></span> <!-- (8) -->
</form>
```

Sr. No.	Description
(7)	Form to enter the numerical value to be calculated.
(8)	Area to display calculation result. In the above example, calculation result is displayed when communication is successful and it is cleared when the communication fails.

- JavaScript

```
var contextPath = $("meta[name='contextPath']").attr("content");

// (9)
var csrfToken = $("meta[name='_csrf']").attr("content");
var csrfHeaderName = $("meta[name='_csrf_header']").attr("content");
$(document).ajaxSend(function(event, xhr, options) {
    xhr.setRequestHeader(csrfHeaderName, csrfToken);
});

// (10)
function plus() {
```

```
    $.ajax(contextPath + "/ajax/plusForForm", {
        type : "POST",
        data : $("#calculationForm").serialize(),
        dataType : "json"
    }).done(function(json) {
        $("#calculationResult").text(json.resultNumber);

    }).fail(function(xhr) {
        // (11)
        var messages = "";
        // (12)
        if(400 <= xhr.status && xhr.status <= 499){
            // (13)
            var contentType = xhr.getResponseHeader('Content-Type');
            if (contentType != null && contentType.indexOf("json") != -1) {
                // (14)
                json = $.parseJSON(xhr.responseText);
                $(json.errorResults).each(function(i, errorResult) {
                    messages += ("<div>" + errorResult.message + "</div>");
                });
            } else {
                // (15)
                messages = ("<div>" + xhr.statusText + "</div>");
            }
        }else{
            // (16)
            messages = ("<div>" + "System error occurred." + "</div>");
        }
        // (17)
        $("#calculationResult").html(messages);
    });

    return false;
}
```

Sr. No.	Description
(9)	To send the request using POST method, CSRF token needs to be set to HTTP header. In the above example, the header name and token value are set in the `<meta>` element of HTML and value is fetched by JavaScript. For details on CSRF measures, refer to *CSRF Countermeasures*.

次のページに続く

Table. 4.56 – 前のページからの続き

Sr. No.	Description
(10)	Ajax function that converts the numerical value specified in form, to request parameter and sends the request for `/ajax/plusForForm` using POST method. In the above example, clicking the button acts as the trigger for Ajax communication however, real time calculation can be implemented by setting lost focus of the text box as the trigger.
(11)	Implementation of error handling is shown below. For server side implementation of error handling, refer to *Input error handling*.
(12)	Determine the HTTP status code and type of error. HTTP status code is stored in the `status` field of XMLHttpRequest object.
(13)	Check whether the response data is in JSON format. In the above example, response data format is checked by referring to the value set in the Content-Type of response header. If the format is not checked and if it a format other than JSON, an error occurs while deserializing to JSON object. If error handling is performed easily at the server side, page may be returned in HTML format.
(14)	Deserialize the response data in JSON object. Response data is stored in the `responseText` field of XMLHttpRequest object. In the above example, error information is fetched from the deserialized JSON object and error message is created.
(15)	Perform the process when the response data is not in JSON format. In the above example, HTTP status text is stored in the error message. HTTP status text is stored in the `statusText` field of XMLHttpRequest object.

次のページに続く

Table. 4.56 – 前のページからの続き

Sr. No.	Description
(16)	Perform the process when there is a server error. In the above example, message notifying it as a system error is stored in error message.
(17)	Perform rendering process when there is an error. In the above example, error message is displayed in the area for displaying calculation result.

> **Warning:** In the above example, processes namely, Ajax communication, DOM operation (rendering) and error handling are performed by the same function. It is recommended to split and implement these processes.

Todo

TBD

Implementation at client side will be explained in detail, in subsequent versions.

> **Tip:** In the above example, JSP code is deleted from JavaScript code by setting CSRF token value and CSRF token header name, in the `<meta>` element of HTML using `<sec:csrfMetaTags />`. Please refer, *Coordination while using Ajax*.
>
> Please note that, CSRF token value and name of CSRF token header can also be fetched by using `${_csrf.token}` and `${_csrf.headerName}` respectively.

Following communication occurs when the "=" button of search form is clicked.
Main points are highlighted.

- Request data

```
POST /terasoluna-gfw-web-blank/ajax/plusForForm HTTP/1.1
Host: localhost:9999
Connection: keep-alive
Content-Length: 19
Accept: application/json, text/javascript, */*; q=0.01
Origin: http://localhost:9999
X-CSRF-TOKEN: a5dd1858-8a4f-4ecc-88bd-a326388ab5c9
X-Requested-With: XMLHttpRequest
User-Agent: Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like␣
↪Gecko) Chrome/30.0.1599.101 Safari/537.36
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
Referer: http://localhost:9999/terasoluna-gfw-web-blank/ajax/xxe
Accept-Encoding: gzip,deflate,sdch
Accept-Language: en-US,en;q=0.8,ja;q=0.6
```

```
Cookie: JSESSIONID=3A486604D7DEE62032BA6C073FC6BE9F

number1=1&number2=2
```

- Response data

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
X-Track: c2d5066d0fa946f584536775f07d1900
Content-Type: application/json;charset=UTF-8
Transfer-Encoding: chunked
Date: Fri, 25 Oct 2013 14:27:55 GMT

{"resultNumber":3}
```

- Response data in case of an input error

```
HTTP/1.1 400 Bad Request
Server: Apache-Coyote/1.1
X-Track: cecd7b4d746249178643b7110b0eaa74
Content-Type: application/json;charset=UTF-8
Transfer-Encoding: chunked
Date: Wed, 04 Dec 2013 15:06:01 GMT
Connection: close

{"errorResults":[{"code":"NotNull","message":"\"number2\"maynotbenull.",
↪"itemPath":"number2"},{"code":"NotNull","message":"\"number1\"maynotbenull.
↪","itemPath":"number1"}]}
```

Posting form data in JSON format

How to fetch processing result by converting form data to JSON format and subsequently posting it using Ajax, is explained here.

Difference between this method and "Posting form data" method, is explained.

- Controller

```
@RequestMapping("xxx")
@Controller
public class XxxController {

    @RequestMapping(value = "plusForJson", method = RequestMethod.POST)
    @ResponseBody
    public CalculationResult plusForJson(
            @Validated @RequestBody CalculationParameters params) { // (1)
        CalculationResult result = new CalculationResult();
        int sum = params.getNumber1() + params.getNumber2();
        result.setResultNumber(sum);
        return result;
    }

    // omitted

}
```

Sr. No.	Description
(1)	Assign `@org.springframework.web.bind.annotation.RequestBody` as the argument annotation of JavaBean for receiving form data. By assigning this annotation, data in JSON format stored in the request body is unmarshalled and converted to object. Specify `@Validated` when input validation is required. For error handling of input validation, refer to "*Input error handling*". For details on input validation, refer to *Input Validation*.

- JavaScript/HTML (JSP)

```
// (2)
function toJson($form) {
    var data = {};
    $($form.serializeArray()).each(function(i, v) {
        data[v.name] = v.value;
    });
```

```
    return JSON.stringify(data);
}

function plus() {

    $.ajax(contextPath + "/ajax/plusForJson", {
        type : "POST",
        contentType : "application/json;charset=utf-8", // (3)
        data : toJson($("#calculationForm")), // (2)
        dataType : "json",
        beforeSend : function(xhr) {
            xhr.setRequestHeader(csrfHeaderName, csrfToken);
        }

    }).done(function(json) {
        $("#calculationResult").text(json.resultNumber);

    }).fail(function(xhr) {
        $("#calculationResult").text("");

    });
    return false;
}
```

Sr. No.	Description
(2)	Function to change form input field to JSON string format.
(3)	Change the media type of Content-Type to `"application/json"` as the data stored in request body is in JSON format.

Following communication occurs when “=” button of the search form mentioned above, is clicked.

Main points are highlighted.

- Request data

```
POST /terasoluna-gfw-web-blank/ajax/plusForJson HTTP/1.1
Host: localhost:9999
Connection: keep-alive
Content-Length: 31
Accept: application/json, text/javascript, */*; q=0.01
Origin: http://localhost:9999
X-CSRF-TOKEN: 9d4f1e0c-c500-43f3-9125-a7a131ff88fa
X-Requested-With: XMLHttpRequest
User-Agent: Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like␣
↪Gecko) Chrome/30.0.1599.101 Safari/537.36
Content-Type: application/json;charset=UTF-8
Referer: http://localhost:9999/terasoluna-gfw-web-blank/ajax/xxe?
Accept-Encoding: gzip,deflate,sdch
Accept-Language: en-US,en;q=0.8,ja;q=0.6
Cookie: JSESSIONID=CECD7A6CB0431266B8D1173CCFA66B95

{"number1":"34","number2":"56"}
```

Input error handling

How to perform error handling when an incorrect input value is specified, is explained here.

Input error handling methods are widely classified into the following.

- Method that performs error handling by providing an exception handling method.
- Method that performs error handling by receiving `org.springframework.validation.BindingResult` as an argument of Controller handler method.

Handling BindException

`org.springframework.validation.BindException` is an exception class generated when an incorrect input value is specified while sending the data as request parameter for binding to JavaBean.

To receive request parameter and form data at the time of GET, in `"application/x-www-form-urlencoded"` format, exception handling of `BindException` class needs to be performed.

- Controller

```
@RequestMapping("xxx")
@Controller
public class XxxController {

    // omitted

    @ExceptionHandler(BindException.class) // (1)
    @ResponseStatus(value = HttpStatus.BAD_REQUEST) // (2)
    @ResponseBody // (3)
    public ErrorResults handleBindException(BindException e, Locale locale) {
↪// (4)
        // (5)
        ErrorResults errorResults = new ErrorResults();
        for (FieldError fieldError : e.getBindingResult().getFieldErrors()) {
            errorResults.add(fieldError.getCode(),
                    messageSource.getMessage(fieldError, locale),
                        fieldError.getField());
        }
        for (ObjectError objectError : e.getBindingResult().
↪getGlobalErrors()) {
            errorResults.add(objectError.getCode(),
                    messageSource.getMessage(objectError, locale),
                        objectError.getObjectName());
        }
        return errorResults;
    }

    // omitted

}
```

Sr. No.	Description
(1)	Define the error handling method in Controller. Assign `@org.springframework.web.bind.annotation.ExceptionHandler` annotation to the error handling method and specify the exception type to be handled in the value attribute. In the above example, `BindException.class` is specified as the exception for binding.
(2)	Specify the HTTP status information sent as response. In the above example, `400` (Bad Request) is specified.
(3)	Assign `@ResponseBody` annotation to write the returned object in response body.
(4)	Declare the exception class to be handled as an argument of the error handling method.
(5)	Implement error handling. In the above example, a JavaBean is created to return the error information.

Tip: Locale object can be received as an argument while creating a message for error handling by considering internationalization.

- JavaBean storing the error information

```
// (6)
public class ErrorResult implements Serializable {

    private static final long serialVersionUID = 1L;
```

```
    private String code;

    private String message;

    private String itemPath;

    public String getCode() {
        return code;
    }

    public void setCode(String code) {
        this.code = code;
    }

    public String getMessage() {
        return message;
    }

    public void setMessage(String message) {
        this.message = message;
    }

    public String getItemPath() {
        return itemPath;
    }

    public void setItemPath(String itemPath) {
        this.itemPath = itemPath;
    }

}
```

```
// (7)
public class ErrorResults implements Serializable {

    private static final long serialVersionUID = 1L;

    private List<ErrorResult> errorResults = new ArrayList<ErrorResult>();

    public List<ErrorResult> getErrorResults() {
        return errorResults;
    }

    public void setErrorResults(List<ErrorResult> errorResults) {
        this.errorResults = errorResults;
    }

    public ErrorResults add(String code, String message) {
        ErrorResult errorResult = new ErrorResult();
        errorResult.setCode(code);
```

```
        errorResult.setMessage(message);
        errorResults.add(errorResult);
        return this;
    }

    public ErrorResults add(String code, String message, String itemPath) {
        ErrorResult errorResult = new ErrorResult();
        errorResult.setCode(code);
        errorResult.setMessage(message);
        errorResult.setItemPath(itemPath);
        errorResults.add(errorResult);
        return this;
    }

}
```

Sr. No.	Description
(6)	JavaBean to store one record of error information.
(7)	JavaBean to store multiple JavaBeans, each of which stores one record of error information. JavaBeans mentioned in (6) are stored as a list.

Handling MethodArgumentNotValidException

`org.springframework.web.bind.MethodArgumentNotValidException` is the exception class generated when an incorrect input value is specified while binding the data stored in the request body to JavaBean using `@RequestBody` annotation.

To receive it in formats such as `"application/json"` or `"application/xml"` etc., exception handling of `MethodArgumentNotValidException` needs to be performed.

- Controller

```
@ExceptionHandler(MethodArgumentNotValidException.class) // (1)
@ResponseStatus(value = HttpStatus.BAD_REQUEST)
```

```
@ResponseBody
public ErrorResults handleMethodArgumentNotValidException(
        MethodArgumentNotValidException e, Locale locale) { // (1)
    ErrorResults errorResults = new ErrorResults();

    // implement error handling.
    // omitted

    return errorResults;
}
```

Sr. No.	Description
(1)	Specify `MethodArgumentNotValidException.class` as an exception for error handling. Other than this, it is same as `BindException`.

Handling HttpMessageNotReadableException

`org.springframework.http.converter.HttpMessageNotReadableException` is the exception class generated when a JavaBean could not be created from the data stored in Body, while binding the data stored in the request body to JavaBean, using `@RequestBody` annotation.

To receive it in formats such as `"application/json"` or `"application/xml"` etc., exception handling of `MethodArgumentNotValidException` needs to be performed.

Note: Causes of specific errors differ depending on the implementation of `HttpMessageConverter` or library to be used.

In `MappingJackson2HttpMessageConverter` implementation, wherein data in JSON format is to be converted to JavaBean using Jackson, if a string is specified in the Integer field instead of number, `HttpMessageNotReadableException` occurs.

- Controller

```
@ExceptionHandler(HttpMessageNotReadableException.class) // (1)
@ResponseStatus(value = HttpStatus.BAD_REQUEST)
@ResponseBody
public ErrorResults handleHttpMessageNotReadableException(
        HttpMessageNotReadableException e, Locale locale) {  // (1)
    ErrorResults errorResults = new ErrorResults();

    // implement error handling.
    // omitted

    return errorResults;
}
```

Sr. No.	Description
(1)	Specify `HttpMessageNotReadableException.class` as the exception of error handling object. Other than this, it is same as `BindException`.

Handling by using BindingResult

When same type of JavaBean is returned in case of normal termination and in case of input error, error handling can be performed by receiving `BindingResult` as the handler method argument.

This method can be used irrespective of the request data format.

When `BindingResult` is not to be specified as handler method argument, it is necessary to implement error handling by the exception handling method mentioned earlier.

- Controller

```
@RequestMapping(value = "plus", method = RequestMethod.POST)
@ResponseBody
public CalculationResult plus(
        @Validated @RequestBody CalculationParameters params,
        BindingResult bResult) { // (1)
    CalculationResult result = new CalculationResult();
    if (bResult.hasErrors()) { // (2)
```

```
        // (3)
        // implement error handling.
        // omitted

        return result; // (4)
    }
    int sum = params.getNumber1() + params.getNumber2();
    result.setResultNumber(sum);
    return result;
}
```

Sr. No.	Description
(1)	Declare `BindingResult` as a handler method argument. `BindingResult` needs to be declared immediately after the JavaBean for input validation.
(2)	Check whether there is any input value error.
(3)	If so, perform error handling for input error. In the above example, although error handling is omitted, it is assumed that settings for error message etc. are performed.
(4)	Return processing result.

Note: In the above example, HTTP status code `200` (OK) is returned as response for both normal process as well as error. When it is necessary to classify HTTP status codes as per processing results, it can be implemented by setting `org.springframework.http.ResponseEntity` as the return value. As another approach, error handling can be implemented by the exception handling method mentioned earlier, without specifying `BindingResult` as the handler method argument.

```
@RequestMapping(value = "plus", method = RequestMethod.POST)
@ResponseBody
public ResponseEntity<CalculationResult> plus(
        @Validated @RequestBody CalculationParameters params,
        BindingResult bResult) {
    CalculationResult result = new CalculationResult();
```

```
    if (bResult.hasErrors()) {

        // implement error handling.
        // omitted

        // (1)
        return ResponseEntity.badRequest().body(result);
    }
    // omitted

    // (2)
    return ResponseEntity.ok().body(result);
}
```

Sr. No.	Description
(1)	Return response data and HTTP status in case of input error.
(2)	Return response data and HTTP status in case of normal termination.

Business error handling

How to handle business errors is explained here.

Methods that handle business errors are widely classified as follows.

- Method that performs error handling by providing a business exception handling method.
- Method that catches business exception in the handler method of Controller and performs error handling.

Handling business exception by exception handling method

Business exceptions are handled by providing an exception handling method same as in case of input error.

This method is recommended when it is necessary to implement the same error handling in requests for multiple handler methods.

- Controller

```
@ExceptionHandler(BusinessException.class) // (1)
@ResponseStatus(value = HttpStatus.CONFLICT) // (2)
@ResponseBody
public ErrorResults handleHttpBusinessException(BusinessException e, // (1)
        Locale locale) {
    ErrorResults errorResults = new ErrorResults();

    // implement error handling.
    // omitted

    return errorResults;
}
```

Sr. No.	Description
(1)	Specify `BusinessException.class` as an exception for error handling. Other than this, it is similar to the input error handling for `BindException`.
(2)	Specify the HTTP status information sent as response. In the above example, `409` (Conflict) is specified.

Handling business exception in handler method

Business exception is caught by enclosing the process where the business error has occurred, in try clause.

This method is implemented when error handling is different for each request.

- Controller

```
@RequestMapping(value = "plus", method = RequestMethod.POST)
@ResponseBody
public ResponseEntity<CalculationResult> plusForJson(
        @Validated @RequestBody CalculationParameters params) {
    CalculationResult result = new CalculationResult();
```

```
    // omitted

    // (1)
    try {

        // call service method.
        // omitted

     // (2)
    } catch (BusinessException e) {

        // (3)
        // implement error handling.
        // omitted

        return ResponseEntity.status(HttpStatus.CONFLICT).body(result);
    }

    // omitted

    return ResponseEntity.ok().body(result);
}
```

Sr. No.	Description
(1)	Enclose the method call where business exception occurs, in try clause.
(2)	Catch business exception.
(3)	Perform the error handling intended for business exception error. In the above example, although error handling is omitted, it is assumed that settings for error message etc. are performed.

4.14 Health Check

4.14.1 Overview

This chapter explains about health check.

Load sharing and fallback of load balancer

A load balancer (hereafter referred to as LB) is used assuming that the Web system receives requests from a large number of users.

A LB is a device to distribute the load on the Web system by allocating requests to multiple servers and characterized in flexibly changing process efficiency of Web system by adding and deleting servers.

Health check is a function to monitor operational status of each server allocating the requests. LB uses the health check function wherein it allocates the requests to the servers which are running normally and does not allocate requests to the servers which are malfunctioning. Accordingly, it is possible to continue the operation without stopping the Web system even when a failure occurs in a specific server. (it is called as a fallback)

In the example below, LB manages 3 servers and allocates requests accordingly.

LB periodically sends requests to the server and monitors operational status of the server by checking status code and response returned from the server. When an abnormality occurs in Server A of Fig., LB detects the same and does not allocate the request to server A.

Client A which is originally connected to server A allocates the request to another server (here, server B) through LB.

Types of health checks

Health checks performed by LB are of various types. Example is shown below.

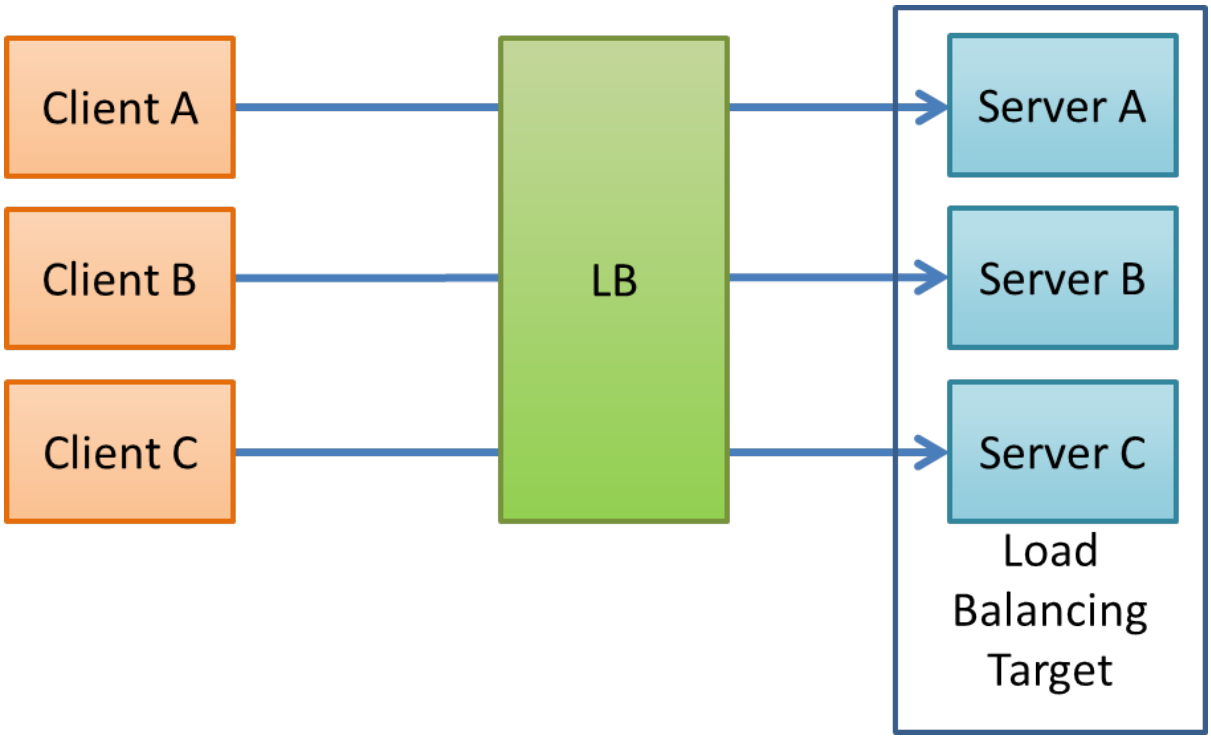


Figure.4.22 **Picture - About Load Balancing**

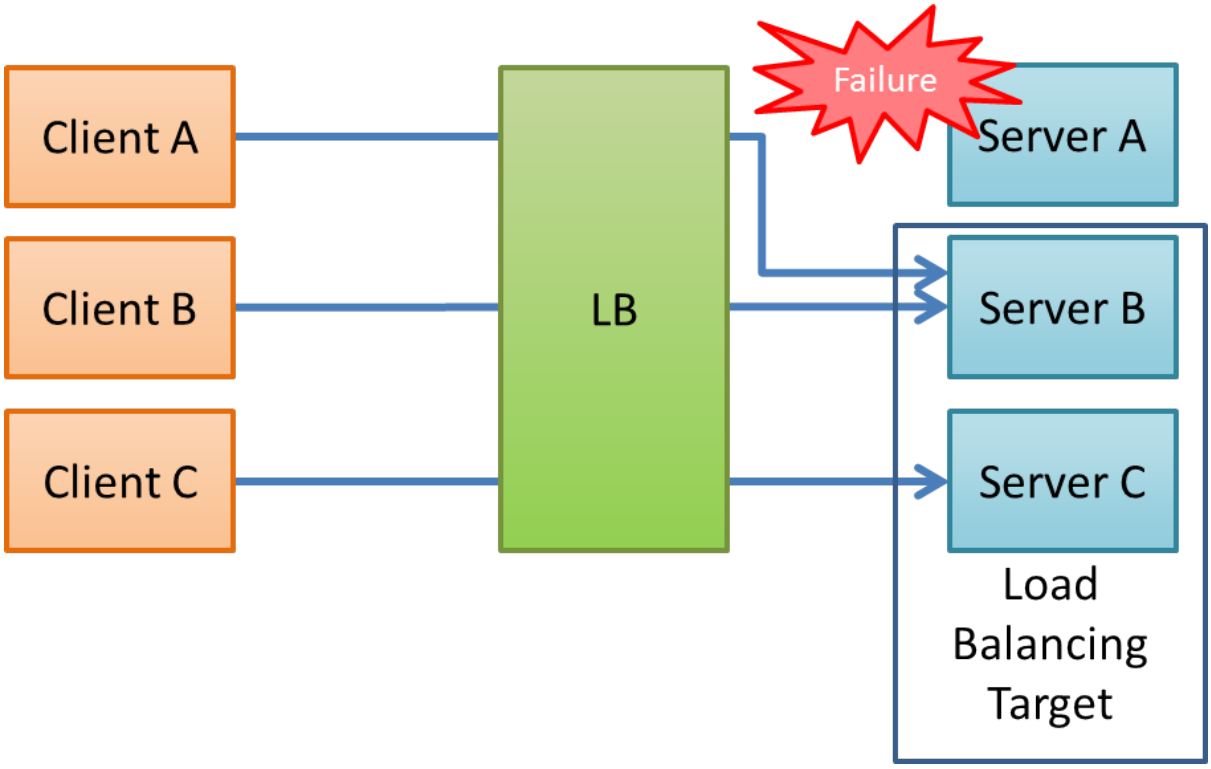


Figure.4.23 **Picture - About Fallback**

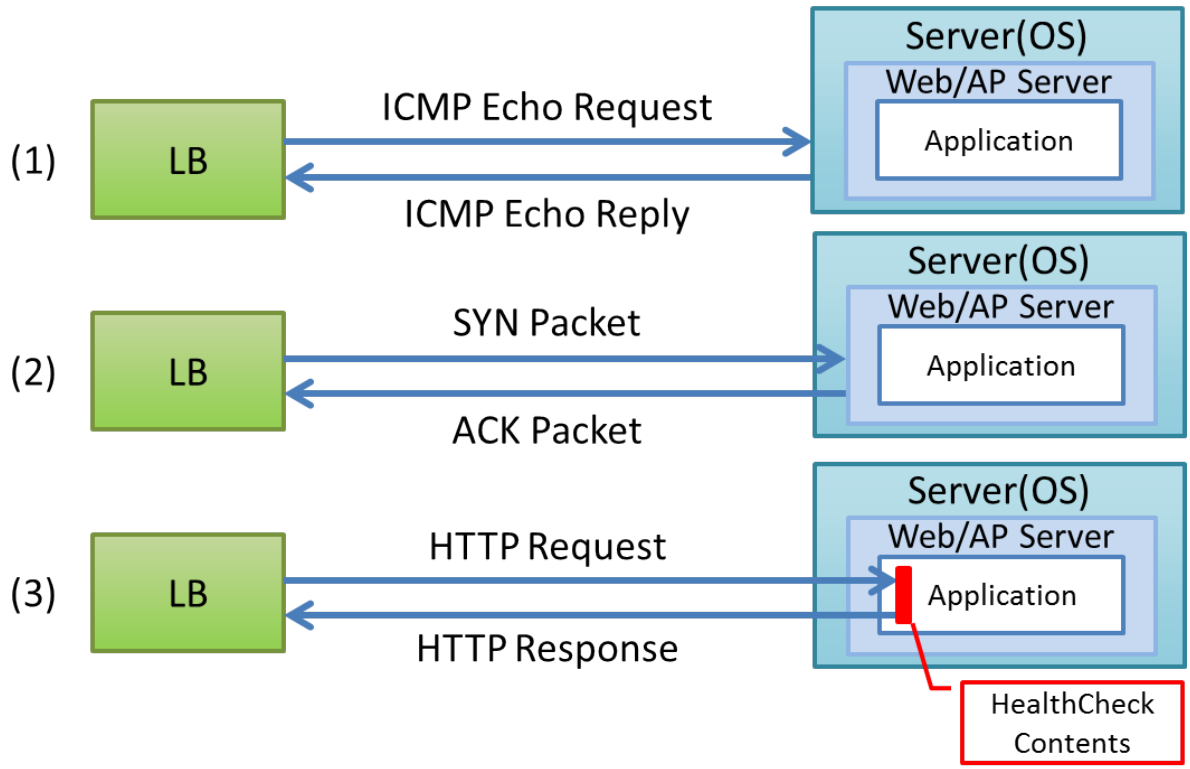


Figure.4.24 **Picture - HealthCheck Example**

Sr. No.	Types of health checks	Details
(1)	Health check in PING	Verify operational status at the network layer level of OSI reference model. PING is sent for a server (OS) and determined as “server running” if appropriate response is received from the server.
(2)	Health check in TCP/UDP	Verify operational status in transport layer level of OSI reference model. A request is sent to TCP port (or UDP port) of Web/AP server and determined as “server running” if the appropriate response is received from the server.
(3)	Health check in application	Verify operational status at the application layer level of OSI reference model. A HTTP request is sent to application running on Web/AP server and determined as “running” if the appropriate response is received from the server.

Verification is not possible for operational status of the application, in health check in PING or TCP/UDP. In case of Web application, the application must also be “running” along with running of server (OS) or Web/AP server.

Hence, this guideline recommends a health check to be performed for the application.

Structure of health check shown in the guideline

This guideline introduces an implementation example for the application wherein a health check is performed in the application.

Basically, a handler consisting of the structure shown in Fig. is implemented which receives a request from LB.

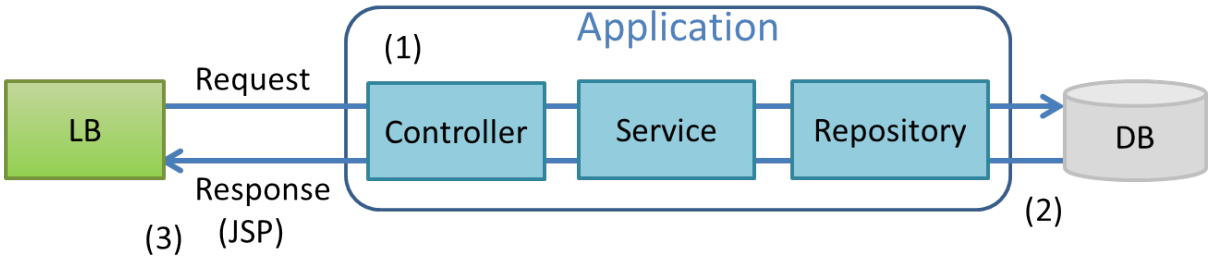


Figure.4.25 **Picture - HealthCheck Configuration**

Sr. No.	Description
(1)	Run Controller, Service and Repository by receiving a request from LB. A method to perform a simple health check also exists when only operational status is to be verified. However, this guideline also verifies whether the structure and framework used by the application is running correctly, using the health check and implements Controller, Service and Repository to achieve technological configuration of targeted application as much as possible.
(2)	Issue SQL from Repository and verify that the database is running. This is because, in an application with database access, an operation cannot be performed normally if an abnormality occurs in database, even if the application itself is running.
(3)	Use JSP as a View which returns a response. Although this guideline explains about JSP, communication method and response format should be changed appropriately in accordance with the characteristics of application, while using REST or SOAP. For details, refer *RESTful Web Service*or *SOAP Web Service (Server/Client)*.

Status codes and responses returned by the implementation example of the guideline are as below.

Health check process results	Status code	Response details
Success	200 (Normal)	3 characters of `OK`.
Error occurrence	Status code set by exception handling function	Response set by exception handling function

For changing configuration of exception handling. refer *Exception Handling*.

4.14.2 How to use

An implementation example shown in *Structure of health check shown in the guideline*is explained.

Repository interface

At first, `HealthCheckRepository` is created. `HealthCheckRepository` executes SQL for health check and verifies database operation.

Note that, an example to access database by using MyBatis3 is explained here. For adopting other methods, refer *Data Access*.

HealthCheckRepository.java

```
package com.example.domain.repository.healthcheck;

public interface HealthCheckRepository {
    void healthcheck();
}
```

A requisite minimum SQL is configured since only whether database can be accessed correctly is to be verified here.

In this guideline, SQL is configured so as to meet the following conditions.

- It should be a reference system
- It should not require parameters

An example of mapping file while using PostgreSQL is given below.

HealthCheckRepository.xml(when PostgreSQL is used)

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
    "http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<mapper namespace="com.example.domain.repository.healthcheck.HealthCheckRepository
↪">

    <select id="healthcheck" resultType="String">
```

```
      SELECT '1'
   </select>

</mapper>
```

Further, an example of mapping file while using Oracle is given below.

HealthCheckRepository.xml(When Oracle is used)

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
   "http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<mapper namespace="com.example.domain.repository.healthcheck.HealthCheckRepository
↪">

   <select id="healthcheck" resultType="String">
      SELECT '1' FROM DUAL
   </select>

</mapper>
```

Service class

Next, a `HealthCheckService` interface and a `HealthCheckServiceImpl` class which implements `HealthCheckService` interface are created.

`HealthCheckServiceImpl` calls `healthcheck` method of `healthcheckRepository`and performs health check for the database.

HealthCheckService.java

```
package com.example.domain.service.healthcheck;

public interface HealthCheckService {
   void healthcheck();
}
```

HealthCheckServiceImpl.java

```
package com.example.domain.service.healthcheck;

import healthcheck.domain.repository.healthcheck.HealthCheckRepository;
```

```
import javax.inject.Inject;

import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

@Service
@Transactional(readOnly = true)
public class HealthCheckServiceImpl implements HealthCheckService {

    @Inject
    HealthCheckRepository healthcheckRepository;

    @Override
    public void healthcheck() {
        healthcheckRepository.healthcheck();
    }
}
```

Controller class

Next, a `HealthCheckController` is created.

`healthcheck` method of `HealthcheckService` is called and transition is done to the specified path based on execution results. When it is verified that "database is running", a view to display `OK.` is returned.

HealthCheckController.java

```
package com.example.app.healthcheck;

import healthcheck.domain.service.healthcheck.HealthCheckService;

import javax.inject.Inject;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
public class HealthCheckController {

    @Inject
    HealthCheckService healthcheckService;

    @RequestMapping(value = "healthcheck") // (1)
    public String healthcheck(){
        healthcheckService.healthcheck();
        return "common/healthcheck/ok"; // (2)
```

```
    }
}
```

Sr. No.	Description
(1)	`value`attribute acts as a URL for health check in order to verify operational status.
(2)	The directory where JSP file is placed, is kept one layer deep so as not to receive configuration of Apache Tiles. For details, refer *Should not receive settings of Apache Tiles*.

JSP file

Finally, a JSP file that is moved once the health check is successful, is created.

While creating JSP file, ensure that a newline character is not inserted between `<%@page>` directive and `OK.` as shown below.

This is to minimise the data volume for the response.

ok.jsp

```
<%@page contentType="text/plain; charset=utf-8" language="java" pageEncoding="utf-8
↪" %>OK.
```

Other precautions must also be taken along with minimising the data volume for the response. For details, refer *Configuration for minimising the data volume for response*.

Configuring access rights

A URL for health check must always be kept active by using authentication and authorization functions, during a health check process.

For example, `<sec:intercept-url>` of spring-security.xml is configured for enabling the access for any role.

An example to perform exclusion settings under `/common/healthcheck` is shown below.

For details, refer *Authorization*.

spring-security.xml

```
<sec:http>
   <sec:intercept-url pattern="/healthcheck/**" access="permitAll"/>
   <!-- omitted -->
</sec:http>
```

Note: If authorization control is removed, URL for health check can be accessed by anyone. Hence, countermeasures like using LB to prevent external access are required.

4.14.3 Appendix

Configuration for minimising the data volume for response

As shown in *JSP file*, precautions must be primarily taken for following points in order to minimise data volume for response.

- *Apache Tiles settings should not be received*
- *Header file and footer file should not be read*
- *Additional line breaks are to be removed from the response*

Respective implementation examples for above are shown below.

Should not receive settings of Apache Tiles

As shown in *Controller class*, JSP file must be placed under a directory which does not conform to `<put-attribute>` tag of tiles-definitions.xml so as not to receive Apache Tiles configuration.
In the default configuration of blank project, since Apache Tiles configuration is applied to JSP corresponding to `/WEB-INF/views/{1}/{2}.jsp`, a directory which is a level deep in the hierarchy is created and JSP file is placed under `/WEB-INF/views/common/healthcheck/`.
For details, refer *Screen Layout using Tiles*.

Prevent reading of header and footer files

Precautions must be taken to check the impact of `<jsp-config>`tag of web.xml on JSP file.

When <include-prelude> and <include-coda> tags are set in <jsp-config> tag,
header and footer files are read in JSP file. Check whether the configuration includes reading of header and footer files.
Configuration example is shown below.

web.xml

```
<jsp-config>
    <jsp-property-group>
        <url-pattern>/WEB-INF/views/common/healthcheck/ok.jsp</url-pattern>
        <el-ignored>false</el-ignored>
        <page-encoding>UTF-8</page-encoding>
        <scripting-invalid>false</scripting-invalid>
        // (1)
    </jsp-property-group>
</jsp-config>
```

Sr. No.	Description
(1)	<include-prelude> tag and <include-coda> tag are not set in order to not to incorporate additional headers and footers.

Remove additional line breaks from response

For example, when <%@taglib> directive is set at the beginning of JSP, in order to use tag library, additional line breaks are output at the beginning of response.
Hence, trimDirectiveWhitespaces attribute is set in <%@page> directive and output of additional line breaks in ok.jsp is prevented.

ok.jsp (while setting trimDirectiveWhitespaces attribute)

```
<%@page contentType="text/plain; charset=utf-8" language="java" pageEncoding="utf-8
↪" trimDirectiveWhitespaces="true" %>OK.
```

Note: /WEB-INF/views/common/include.jsp is configured in <include-prelude>of web.xml,

in the default configuration of Blank project. Hence, it is necessary to configure as shown above or modify `<url-pattern>`so as not to include ok.jsp. If these actions are not taken, include.jsp is read by entire JSP file and line breaks are output in ok.jsp.

Warning: When WebLogic is used, line breaks are not removed when an additional character exists before `<%@page>`directive, even when `trimDirectiveWhitespaces`attribute is configured as shown above. Hence, a different method should be used. As an example, how to configure a `<trim-directive-whitespaces>` tag in `<jsp-property-group>` tag of web.xml is shown below.

web.xml(while setting <trim-directive-whitespaces> tag)

```
<jsp-config>
    <jsp-property-group>
        <url-pattern>/WEB-INF/views/common/healthcheck/ok.jsp</url-pattern>
↪ // (1)
        <el-ignored>false</el-ignored>
        <page-encoding>UTF-8</page-encoding>
        <scripting-invalid>false</scripting-invalid>
        <trim-directive-whitespaces>true</trim-directive-whitespaces> //␣
↪(2)
    </jsp-property-group>
</jsp-config>
```

Sr. No.	Description
(1)	Specify only ok.jsp in `<url-pattern>` tag in order to apply `<trim-directive-whitespaces>` tag only in ok.jsp.
(2)	Remove additional line breaks from targeted jsp file (ok.jsp) by setting `true` in `<trim-directive-whitespaces>` tag.

5

Web Services

The web service architecture adopted in this guideline is explained in detail.

5.1 RESTful Web Service

5.1.1 Overview

This section explains the basic concept of RESTful Web Service and its development by using Spring MVC.

Refer to the following for basic description of architecture, design and implementation of RESTful Web Service

- "*Architecture*"

  Basic architecture of RESTful Web Service is explained.

- "*How to design*"

  Points to be considered while designing a RESTful Web Service are explained.

- "*How to use*"

  Application structure of RESTful Web Service and API implementation methods are explained.

What is RESTful Web Service

REST is an abbreviation of "**RE**presentational **S**tate **T**ransfer", and is one of the **architecture styles** for building an application, wherein data is exchanged between client and server.

REST architecture style consists of various important fundamental rules and the services which are in accordance with these rules (system etc.) are expressed as **RESTful**.

In other words, “RESTful Web Service” is a Web service that is built in accordance with the fundamental rules of REST.

The concept of “resource” is of prime importance in RESTful Web Service.

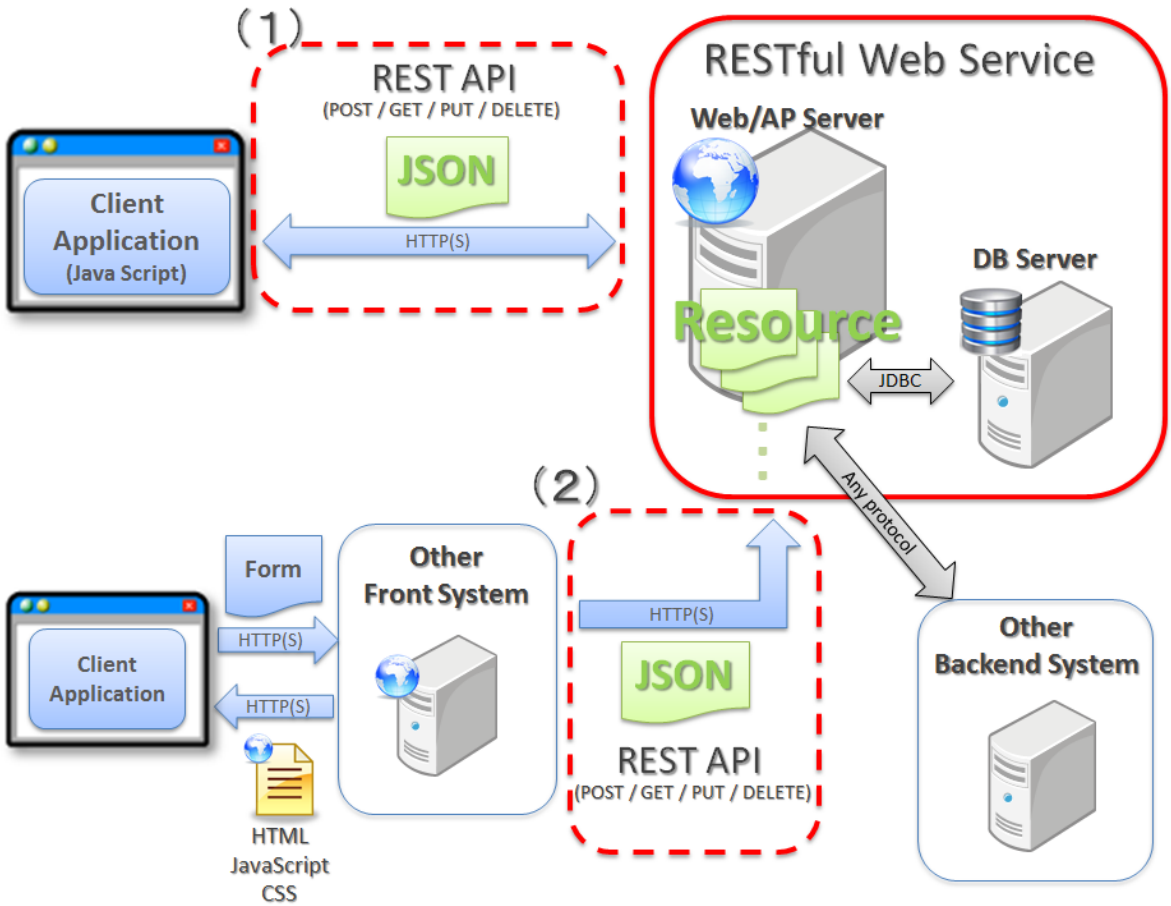
In RESTful Web Service, the information that should be provided to the client is extracted as “resource”, from the information stored in database etc. and the CRUD operation for this extracted “resource” is provided to the client using HTTP methods (POST/GET/PUT/DELETE).

The CRUD operation for “resource” is called “REST API” or “RESTful API” and is mentioned as “REST API” in this guideline.

Further, JSON or XML that have higher message visibility and data structure expressivity, are used as the message formats while exchanging resources between client and server.

System configuration of the application that uses RESTful Web Service mainly consists of following 2 patterns.

The basic architecture for exchanging resources between client and server is explained by using “*Architecture*”.



Sr. No.	Description
(1)	A resource is directly exchanged between client application with user interface and RESTful Web Service. This pattern is used to separate user interface dependent logic with higher number of requirement & specification changes and the logic for a data model which is more universal with less number of changes.
(2)	Rather than directly exchanging the resource with client applications having user interface, the resource is exchanged between systems. This pattern is used while building a system wherein, the business data stored by each system is managed centrally.

RESTful Web Service development

RESTful Web Service is developed in TERASOLUNA Server Framework for Java (5.x) using Spring MVC functionalities.

The common functionalities necessary for RESTful Web Service development are built in Spring MVC by default.

As a result, RESTful Web Service development can be initiated without adding any specific settings or implementations.

Main common functionalities built in Spring MVC by default, are given below.

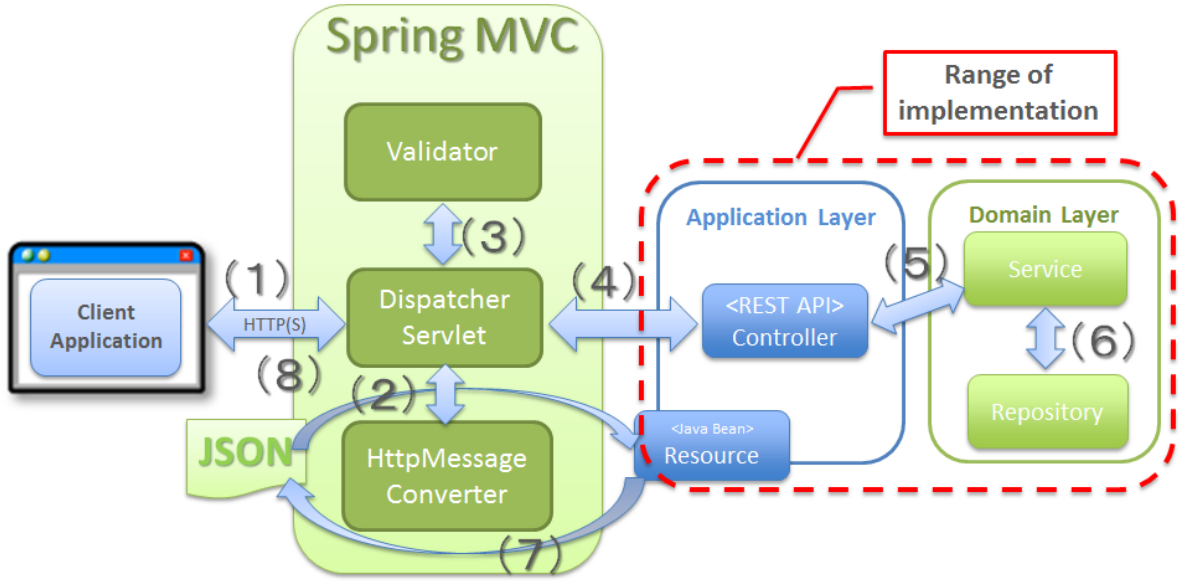
These functionalities can only be enabled by specifying annotations in the methods of the Controller that provides REST API.

Sr. No.	Function overview
(1)	It is a function which converts the JSON or XML format message set in request BODY, to Resource object (JavaBean) and delivers it to the Controller class method (REST API).
(2)	It is a function which implements input validation for the value stored in Resource object (JavaBean) that has been converted from message.
(3)	It is a function which converts Resource object (JavaBean) returned from the Controller class method (REST API) to JSON or XML format and sets it in response BODY.

Note: Exception handling

It is necessary to implement exception handling for each project since a generic functionality for the same is not provided by Spring MVC. For details on exception handling, refer to "*Implementing exception handling*".

When RESTful Web Service is developed using Spring MVC, the application is configured as given below. Among these, implementation is necessary for the portion marked with red frame.



Sr. No.	Process layer	Description
(1)	Spring MVC (Framework)	Spring MVC receives a request from client and determines the REST API (handler method of Controller) to be called.
(2)		Spring MVC converts the JSON format message specified in request BODY to Resource object by using `HttpMessageConverter`.
(3)		Spring MVC performs input validation for the value stored in Resource object using `Validator`.
(4)		Spring MVC calls REST API. Here, the Resource that has been converted from JSON and for which input validation is carried out, is delivered to REST API.
(5)	REST API	REST API calls Service method and performs the process for DomainObject such as Entity etc.

次のページに続く

Table. 5.1 – 前のページからの続き

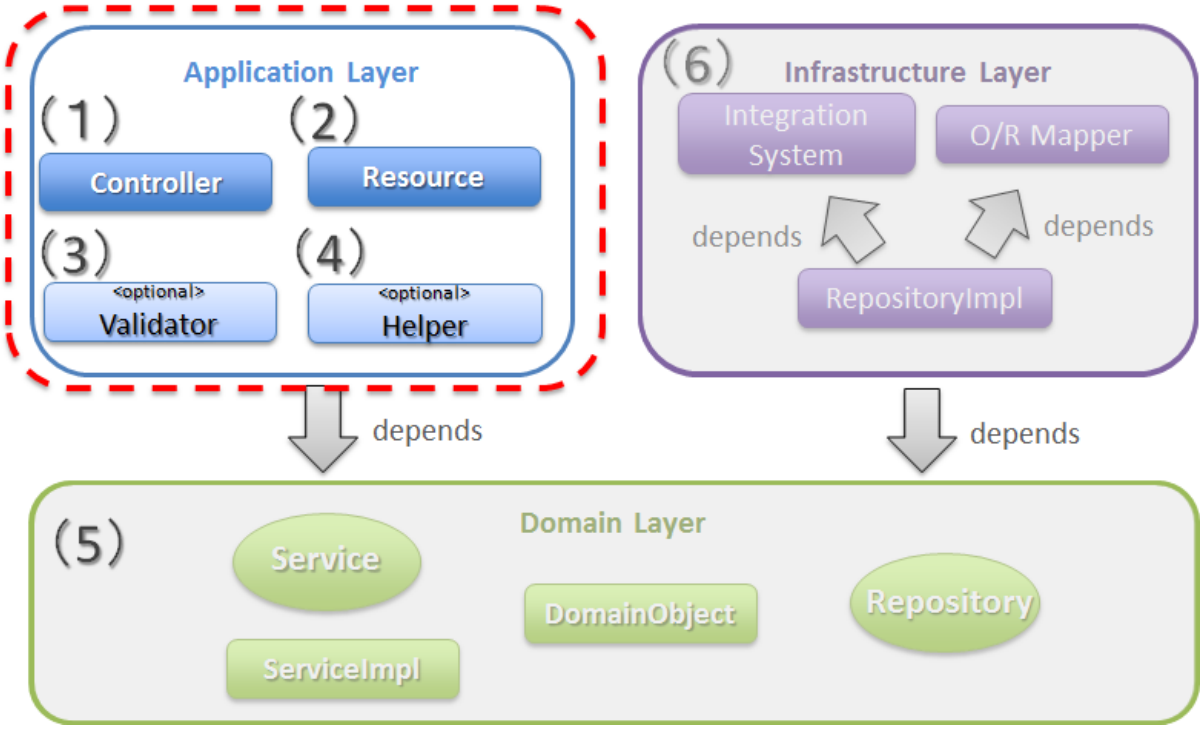
Sr. No.	Process layer	Description
(6)		Service method calls the Repository method and performs CRUD process for the DomainObject such as Entity etc.
(7)	Spring MVC (Framework)	Spring MVC converts the Resource object returned from REST API to JSON format message, by using `HttpMessageConverter`.
(8)		Spring MVC sets JSON format message in response BODY and responds to client.

Configuration for RESTful Web Service module

A lot of RESTful Web Service specific processing can be entrusted to Spring MVC by using the functionalities provided by the framework.

Therefore, configuration of the module to be developed is almost same as the development of conventional Web application that responds with HTML.

Configuration elements of the module are explained below.



- **Module for application layer**

Sr. No.	Module name	Description
(1)	Controller class	A class that provides REST API. Controller class is created by resource unit and specifies end points (URI) of REST API for each resource. CRUD process for the resource is implemented by delegating it to the Service of domain layer.
(2)	Resource class	Java Bean representing JSON (or XML) that acts as I/O for REST API. Annotation for Bean Validation and annotation for controlling JSON or XML format are specified in this class.
(3)	Validator Class (Optional)	Class that implements correlation validation for input value. If the correlation validation for input value is unnecessary, this class need not be created. Hence, it is considered as optional. For input value correlation validation, refer to "*Input Validation*".
(4)	Helper Class (Optional)	Class which implements the process that assists the process to be performed by the Controller. This class is created with the aim of simplifying the Controller processing. Basically, it implements a method that performs conversion of Resource object and DomainObject models. If the model can be converted simply by using copy of the value, "*Bean Mapping (Dozer)*" may be used without creating the Helper class. Hence, it is considered as optional.

- **Domain layer module**

Sr. No.	Description
(5)	The description is beyond the scope of this section since the module implemented in the domain layer is independent of application type. For role of each module, refer to "*Application Layering*" and for domain layer development, refer to "*Domain Layer Implementation*".

- **Infrastructure layer module**

Sr. No	Description
(6)	The description is beyond the scope of this section since the module implemented in the infrastructure layer is independent of application type. Refer to "*Application Layering*" for role of each module and "*Implementation of Infrastructure Layer*" for development of infrastructure layer.

REST API implementation sample

Before giving a detailed explanation, an implementation sample of Resource class and Controller class is given below to let one understand the kind of class created in the application layer.

The implementation sample given below is the REST API of Todo resource which is the topic of *Tutorial (Todo Application REST)*.

Note: It is strongly recommended to practice **:doc:'../../Tutorial/TutorialREST' first, before reading the detailed explanation.**

Aim of the tutorial is to emphasize the saying "Practice makes one perfect". Prior to detailed explanation, the user can gain the experience of actually practicing RESTful Web Service development using TERASOLUNA Server Framework for Java (5.x), with the help of this tutorial. When this firsthand experience of RESTful Web Service development is followed by reading the detailed explanation, the user gains a deeper understanding of the development.

Especially when the user does not have any experience of RESTful Web Service development, it is recommended to follow a process in the order namely, “Tutorial practice” –> “Detailed explanation of architecture, design and development (described in subsequent sections)” –> “Tutorial revision (Re-practice)”.

- Resources handled in implementation sample

  Resources handled in the implementation sample (Todo resources) are set in following JSON format.

```
{
    "todoId" : "9aef3ee3-30d4-4a7c-be4a-bc184ca1d558",
    "todoTitle" : "Hello World!",
    "finished" : false,
    "createdAt" : "2014-02-25T02:21:48.493+0000"
}
```

- Resource class implementation sample

  Resource class is created as the JavaBean representing the Todo resources shown above.

```
package todo.api.todo;

import java.io.Serializable;
import java.util.Date;

import javax.validation.constraints.NotNull;
import javax.validation.constraints.Size;

public class TodoResource implements Serializable {

    private static final long serialVersionUID = 1L;

    private String todoId;

    @NotNull
    @Size(min = 1, max = 30)
    private String todoTitle;

    private boolean finished;
```

```
    private Date createdAt;

    public String getTodoId() {
        return todoId;
    }

    public void setTodoId(String todoId) {
        this.todoId = todoId;
    }

    public String getTodoTitle() {
        return todoTitle;
    }

    public void setTodoTitle(String todoTitle) {
        this.todoTitle = todoTitle;
    }

    public boolean isFinished() {
        return finished;
    }

    public void setFinished(boolean finished) {
        this.finished = finished;
    }

    public Date getCreatedAt() {
        return createdAt;
    }

    public void setCreatedAt(Date createdAt) {
        this.createdAt = createdAt;
    }
}
```

- Implementation sample for Controller class (REST API)

  Following five REST APIs (Controller handler methods) are created for Todo resource.

Sr. No.	API Name	HTTP Method	Path	Status Code	Description
(1)	GET Todos	GET	`/api/v1/todos`	200 (OK)	All Todo resources are fetched.
(2)	POST Todos	POST	`/api/v1/todos`	201 (Created)	A new Todo resource is created.
(3)	GET Todo	GET	`/api/v1/todos/{todoId}`	200 (OK)	One Todo resource is fetched.
(4)	PUT Todo	PUT	`/api/v1/todos/{todoId}`	200 (OK)	Todo resource is updated to “completed”.
(5)	DELETE Todo	DELETE	`/api/v1/todos/{todoId}`	204 (No Content)	Todo resource is deleted.

```
package todo.api.todo;

import java.util.ArrayList;
import java.util.Collection;
import java.util.List;

import javax.inject.Inject;

import org.dozer.Mapper;
import org.springframework.http.HttpStatus;

import org.springframework.validation.annotation.Validated;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
```

```
import org.springframework.web.bind.annotation.ResponseStatus;
import org.springframework.web.bind.annotation.RestController;

import todo.domain.model.Todo;
import todo.domain.service.todo.TodoService;

@RestController
@RequestMapping("todos")
public class TodoRestController {
    @Inject
    TodoService todoService;
    @Inject
    Mapper beanMapper;

    // (1)
    @RequestMapping(method = RequestMethod.GET)

    @ResponseStatus(HttpStatus.OK)
    public List<TodoResource> getTodos() {
        Collection<Todo> todos = todoService.findAll();
        List<TodoResource> todoResources = new ArrayList<>();
        for (Todo todo : todos) {
            todoResources.add(beanMapper.map(todo, TodoResource.class));
        }
        return todoResources;
    }

    // (2)
    @RequestMapping(method = RequestMethod.POST)

    @ResponseStatus(HttpStatus.CREATED)
    public TodoResource postTodos(@RequestBody @Validated TodoResource␣
↪todoResource) {
        Todo createdTodo = todoService.create(beanMapper.map(todoResource,␣
↪Todo.class));
        TodoResource createdTodoResponse = beanMapper.map(createdTodo,␣
↪TodoResource.class);
        return createdTodoResponse;
    }

    // (3)
    @RequestMapping(value="{todoId}", method = RequestMethod.GET)

    @ResponseStatus(HttpStatus.OK)
    public TodoResource getTodo(@PathVariable("todoId") String todoId) {
        Todo todo = todoService.findOne(todoId);
        TodoResource todoResource = beanMapper.map(todo, TodoResource.class);
        return todoResource;
    }
```

```
    // (4)
    @RequestMapping(value="{todoId}", method = RequestMethod.PUT)

    @ResponseStatus(HttpStatus.OK)
    public TodoResource putTodo(@PathVariable("todoId") String todoId) {
        Todo finishedTodo = todoService.finish(todoId);
        TodoResource finishedTodoResource = beanMapper.map(finishedTodo,␣
↪TodoResource.class);
        return finishedTodoResource;
    }

    // (5)
    @RequestMapping(value="{todoId}", method = RequestMethod.DELETE)

    @ResponseStatus(HttpStatus.NO_CONTENT)
    public void deleteTodo(@PathVariable("todoId") String todoId) {
        todoService.delete(todoId);
    }

}
```

5.1.2 Architecture

This section explains the architecture for building a RESTful Web Service.

Resource Oriented Architecture (ROA) is used as the architecture for building RESTful Web Service.

ROA is an abbreviation of "**R**esource **O**riented **A**rchitecture" and defines **the basic architecture for building a Web Service in accordance with REST architecture style (rules)**.

It is important to thoroughly understand ROA architecture when creating RESTful Web Service.

This section explains following 7 elements of ROA architecture.

These form important architectural elements for building RESTful Web Service. However, it is not always necessary to apply all of these elements.

Necessary elements should be applied after considering the characteristics of the application to be developed.

Following five architectural elements must be applied regardless of the application characteristics.

Sr. No.	Architecture	Architecture overview
(1)	*Publishing as a resource on Web*	It is published as a Web resource through which information stored in the system is provided to the client.
(2)	*Identifying the resource using URI*	URI (Universal Resource Identifier) that can uniquely identify a Web resource is assigned to the resource published to the client.
(3)	*Resource operations using HTTP methods*	Resource related operations are implemented by using different HTTP methods (GET, POST, PUT and DELETE).
(4)	*Using an appropriate format*	JSON or XML that represents the data structure, is used as resource format.
(5)	*Using the appropriate HTTP status code*	Appropriate HTTP status code is set in the response returned to the client.

Following two architectural elements are applied depending on the characteristics of an application.

Sr. No.	Architecture	Architectural elements
(6)	*Stateless communication between client and server*	This element enables to perform the process only by the information requested from client, without retaining the application status on the server.
(7)	*Link to related resource*	It includes links to other resources (URI) inside a resource that are related to the specified resource.

Publishing as a resource on Web

It is published as a resource on Web as the means to provide information stored in the system to client.

It signifies that resources can be accessed using HTTP protocol and URI is used as a method to identify resources.

For example, following information is published on the Web as resource, for a Web system providing shopping site.

- Product information
- Stock information
- Order information
- Member information
- Authentication information for each member (Login ID and password etc.)
- Order history information for each member
- Authentication history information for each member
- and more ...

Identifying the resource using URI

URI (Universal Resource Identifier) that can uniquely identify a resource on the Web, is assigned to the resource to be published to the client.

URL (Uniform Resource Locator), which is a subset of the URI, is actually used.

In ROA, the ability to access a resource on the Web using URI, is called as “Addressability”.

It signifies that on using the same URI, the same resource can be accessed from anywhere.

URI assigned to RESTful Web Service is a combination of “**a noun that indicates the type of resource**” and “**a value (ID etc.) that uniquely identifies a resource**”.

For example, URI of product information handled by a Web system that provides a shopping site, is given below.

- *http://example.com/api/v1/items*

  "**items**" portion is the "noun that represents the type of resource". If there are multiple resources, a plural noun is used.

  In the above example, a plural noun is specified to indicate the product information. It forms the URI for batch operation of product information. If replaced to a file system, it corresponds to a directory.

- *http://example.com/api/v1/items/I312-535-01216*

  The part "**I312-535-01216**" in the above URI, represents "the value that identifies the resource" and varies for each resource.

  In the above example, product ID is specified as the value for uniquely identifying product information. It acts as the URI used to handle specific product information. If replaced by a file system, it corresponds to the files stored in a directory.

Warning: Verbs that indicate operations cannot be included in the URI assigned to RESTful Web Service are as shown below.

- *http://example.com/api/v1/items?get&itemId=I312-535-01216*
- *http://example.com/api/v1/items?delete&itemId=I312-535-01216*

URI mentioned in the above example is not suitable to be assigned to RESTful Web Service since it includes verbs like **get** or **delete**.

In RESTful Web Service, **Resource related operations are represented by using HTTP methods (GET, POST, PUT and DELETE).**

Resource operations using HTTP methods

Resource operations can be performed by using HTTP methods (GET, POST, PUT, DELETE).

In ROA, HTTP methods are called as "Unified interface".

It implies that HTTP methods can be executed for all the resources published on the Web and that the meaning of HTTP method does not change with each resource.

The association of resource operations assigned to HTTP methods and the post-conditions ensured by each operation, are explained below.

Sr. No.	HTTP method	Resource operations	Post-conditions that the operation should ensure
(1)	GET	Resource is fetched.	Safety, idempotency.
(2)	POST	Resource is created.	Server assigns the URI for created resource, this assigned URI is set to Location header of response and is returned to client.
(4)	PUT	Resource is created or updated.	Idempotency.
(5)	PATCH	Resource difference is updated.	Idempotency.
(6)	DELETE	Resource is deleted.	Idempotency.
(7)	HEAD	Meta information of resource is fetched. Same process as GET is performed and responds with header only.	Safety, Idempotency.
(8)	OPTIONS	Responds with a list of HTTP methods that can be used for resources.	Safety, idempotency.

Note: Ensuring safety and idempotency

When resource operation is performed using HTTP method, it is necessary to ensure “safety” and “idempotency” as post conditions.

[Safety]

It ensures that even if a particular value is multiplied several times by 1, the value does not change. (for example, if 10 is multiplied several times by 1, result remains 10). This guarantees that even if an operation is carried out for several times, resource status does not change.

[Idempotency]

It ensures that even if a value is multiplied a number of times by 0, the value remains 0 (for example, if 10 is multiplied a number of times or just once by 0, the result remains 0). This signifies that once an operation is performed, resource status does not change even if the same operation is performed later for a number of times. However, when another client is modifying the status of the same resource, idempotency need not be ensured and can be handled as a precondition error.

Tip: When client specifies the URI assigned to a resource for creating a resource

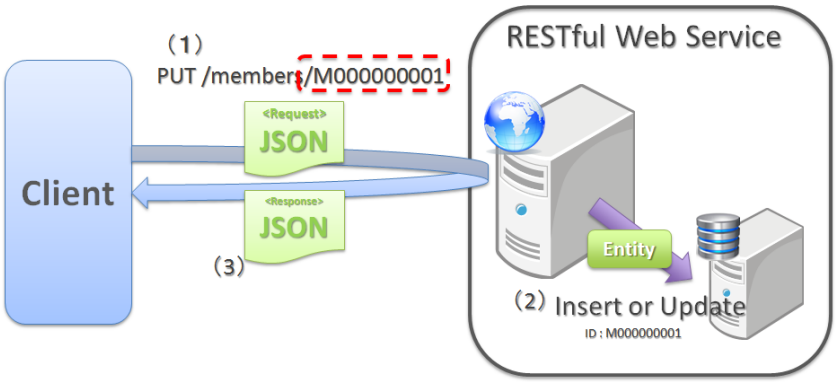
To create a resource, when the URI to be assigned to the resource is specified by client, **PUT method is called for the URI assigned to the resource to be created.**

When creating a resource using PUT method, the general operation is to,

- Create a resource when no resource exists in the specified URI
- Modify resource status when a resource already exists

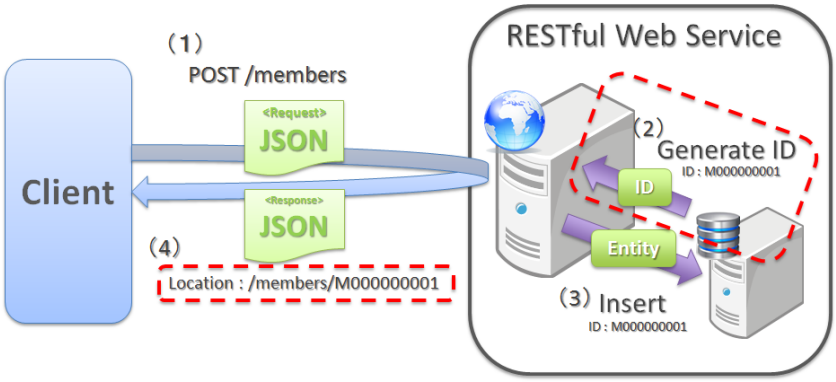
Following is the difference in process images while creating a resource using PUT and POST methods.

[Process image while creating a resource using PUT method]



Sr. No.	Description
(1)	PUT method is called by specifying URI (ID) of the resource to be created in URI.
(2)	Entity is created for the ID specified in URI. If the entity has already been created with same ID, the contents are updated.
(3)	Created or updated resource is sent as a response.

[Process image while creating a resource using POST method]



Sr. No.	Description
(1)	POST method is called.
(2)	ID that identifies the requested resource is generated.
(3)	Entity for the ID generated in (2) is created.
(4)	Created resource is sent as a response. URI for accessing the generated resource is set in the Location header of response.

Using an appropriate format

JSON or XML that indicate data structure, are used for resource format.

However, formats other than JSON or XML can also be used depending on the type of resource.
For example, a resource classified as statistical information can be published with line graph represented in image format (Binary data).

When multiple formats are supported as resource formats, any of the following methods is used to change the format.

- **Changing the format using an extension.**

  Response format can be changed by specifying the extension.
  This guideline recommends changing the format using extension.
  The reasons for recommending this format are, responding format can be easily specified and as the responding format is included in URI, it results in an intuitive URI.

  Note: Examples of URI where format is changed using extension

  - *http://example.com/api/v1/items.json*
  - *http://example.com/api/v1/items.xml*
  - *http://example.com/api/v1/items/I312-535-01216.json*
  - *http://example.com/api/v1/items/I312-535-01216.xml*

- **Changing format by using the MIME type in Accept header of request.**

  A typical MIME type used in RESTful Web Service is shown below.

Sr. No.	Format	MIME type
(1)	JSON	application/json
(2)	XML	application/xml

Using the appropriate HTTP status code

Appropriate HTTP status code is set in the response to be returned to the client.

Value indicating the method by which server has processed the request received from the client, is set in HTTP status code.

This is an HTTP specification and it is recommended to conform to the HTTP specifications wherever possible.

Tip: HTTP Specifications

Refer to RFC 7230 (Hypertext Transfer Protocol – HTTP/1.1) - 3.1.2 Status Line.

In a traditional Web system wherein HTML is returned in the browser, regardless of the process results, it was common that `"200 OK"` was returned as the response and process results were displayed in entity body (HTML),

In a traditional Web application that returns HTML, there were no issues since an operator (human) determined the process results.

However, if this structure is used to build a RESTful Web Service, following issues may exist potentially. Hence, it is recommended to set appropriate status codes.

Sr. No.	Potential issues
(1)	Even in cases where only the process result (success and failure) is to be determined, unnecessary process has to be performed, as analysis process is mandatory for entity body.
(2)	Since it is mandatory to be aware of the unique error codes defined in the system while handling errors, it may adversely affect the architecture (design and implementation) at the client side.
(3)	Intuitive error analysis may be obstructed when analyzing error causes at client side, since understanding the meaning of unique error codes defined in the system is required for the same.

Note: In RFC 7230 which specifies HTTP message syntax, output of description clauses of HTTP status code (`reason-phrase` ☒) is not mandatory and it is prescribed that the client should be ignored. For example, description clauses are not output in Tomcat 8.5 of implementation which is in conformance with RFC 7230.

Refer to RFC 7230(Hypertext Transfer Protocol – HTTP/1.1) - 3.1.2 Status Line.

Stateless communication between client and server

In this communication, only the information requested by the client is processed, without retaining the application status on the server.

In ROA, a state wherein application status is not retained on the server, is called “stateless”.

It signifies that application status is not retained in application server memory (HTTP session etc.) and resource related operations can be completed only by using the requested information.

In this guideline, **it is recommended to retain “stateless” state wherever possible.**

Note: Application status

Web page transition status, selection status for input value, pull down/checkbox/radio buttons and authentication status etc. are included in application status.

Note: Relation with CSRF measures

Please note that the "Stateless" state between client and server cannot be retained when the CSRF measures described in this guideline are implemented for RESTful Web Service as, the token values for CRSF measures are stored in HTTP sessions.

As a result, system availability must be considered while implementing CSRF measures.

Following measures need to be implemented for a system that requires high availability.

- Perform AP server clustering and session replication.
- Use a destination other than AP server memory for storing a session.

However, above measures may affect the performance. Hence, it is necessary to consider performance requirements as well.

For CSRF measures, refer to *CSRF Countermeasures*.

Todo

TBD

When high availability is required, it is advisable to review an architecture wherein, "token values for CSRF measures are stored in a destination other than the AP server memory (HTTP session)".

Basic architecture is currently under review and will be documented in subsequent versions.

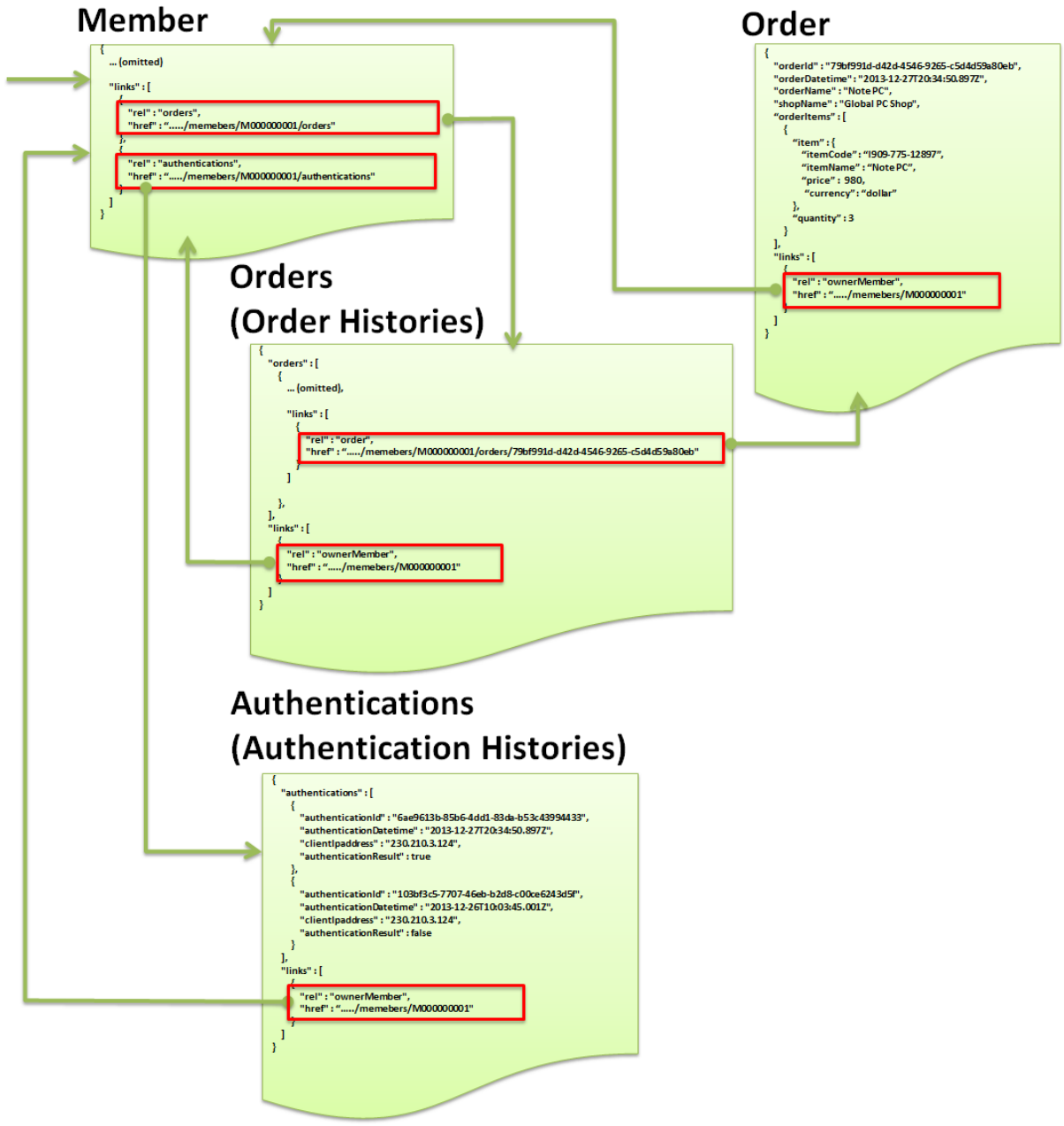
Link to related resource

Hypermedia link (URI) to another resource related to the specified resource, is included in the resource.

In ROA, the process of incorporating a hypermedia link for another resource in the resource status display, is called “Connectivity”.

It signifies that both the linked resources retain this mutual link and all the related resources can be accessed by following this link.

Connectivity of resources is described below, with the example of member information resource of a shopping site.



Sr. No.	Description

(1)	Following JSON is returned when the member information resource is fetched (`GET http://example.com/api/v1/members/M000000001`).
	`{`
	`"memberId" : "M000000001",`
	`"memberName" : "John Smith",`
	`"address" : {`
	`"address1" : "45 West 36th Street",`
	`"address2" : "7th Floor",`
	`"city" : "New York",`
	`"state" : "NY",`
	`"zipCode" : "10018"`
	`},`
	`"links" : [`
	`{`
	`"rel" : "orders",`
	`"href" : "http://example.com/api/v1/memebers/M000000001/orders`
	`},`
	`{`
	`"rel" : "authentications",`
	`"href" : "http://example.com/api/v1/memebers/M000000001/authen`
	`}`
	`]`
	`}`
	
	Highlighted portion is the hypermedia link (URI) to other related resource.
	In the above example, connectivity is retained between the order history resource and authentication history resource of each member.

次のページに続く

Table. 5.2 – 前のページからの続き

Sr. No.	Description

(2)	Following JSON is returned when order history resource is fetched (`GET http://example.com/api/v1/memebers/M000000001/orders`) using the hypermedia link (URI) set in the returned JSON.
	`{`
	`"orders" : [`
	`{`
	`"orderId" : "029b49d7-0efa-411b-bc5a-6570ce40ead8",`
	`"orderDatetime" : "2013-12-27T20:34:50.897Z",`
	`"orderName" : "Note PC",`
	`"shopName" : "Global PC Shop",`
	`"links" : [`
	`{`
	`"rel" : "order",`
	`"href" : "http://example.com/api/v1/memebers/M00000000`
	`}`
	`]`
	`},`
	`{`
	`"orderId" : "79bf991d-d42d-4546-9265-c5d4d59a80eb",`
	`"orderDatetime" : "2013-12-03T19:01:44.109Z",`
	`"orderName" : "Orange Juice 100%",`
	`"shopName" : "Global Food Shop",`
	`"links" : [`
	`{`
	`"rel" : "order",`
	`"href" : "http://example.com/api/v1/memebers/M00000000`
	`}`
	`]`
	`}`
	`],`
	`"links" : [`
	`{`
	`"rel" : "ownerMember",`
	`"href" : "http://example.com/api/v1/memebers/M000000001"`
	`}`
	`]`
	`}`
	
	Highlighted portion is the hypermedia link (URI) for another related resource.
	In the above example, connectivity is retained between the resource for owner member of order history and the order history resource.

次のページに続く

Table. 5.2 – 前のページからの続き

Sr. No.	Description

(3)	Following JSON is returned when the resource for owner member of order history is fetched again (`GET http://example.com/api/v1/memebers/M000000001`) and the resource for authentication history is fetched (`GET http://example.com/api/v1/memebers/M000000001/authentications/`) using hypermedia link (URI) set in the returned JSON.
	`{`
	`"authentications" : [`
	`{`
	`"authenticationId" : "6ae9613b-85b6-4dd1-83da-b53c43994433",`
	`"authenticationDatetime" : "2013-12-27T20:34:50.897Z",`
	`"clientIpaddress" : "230.210.3.124",`
	`"authenticationResult" : true`
	`},`
	`{`
	`"authenticationId" : "103bf3c5-7707-46eb-b2d8-c00ce6243d5f",`
	`"authenticationDatetime" : "2013-12-26T10:03:45.001Z",`
	`"clientIpaddress" : "230.210.3.124",`
	`"authenticationResult" : false`
	`}`
	`],`
	`"links" : [`
	`{`
	`"rel" : "ownerMember",`
	`"href" : "http://example.com/api/v1/memebers/M000000001"`
	`}`
	`]`
	`}`
	
	Highlighted portion is the hypermedia link (URI) to other linked resource.
	In the above example, connectivity is retained with respect to the resource for owner member of authentication history.

It is not mandatory for a resource to include hypermedia link (URI) to another resource.

When all the endpoints (URI) of REST API are already published, even if the link for related resource is set in the resource, it is highly unlikely that it will be used.

Particularly, it makes no sense to provide links for the REST API that exchanges resources between systems, since REST API end points that are already published can be accessed directly.

There is no need to provide a link where it is not required.

In contrast, when a resource is to be directly exchanged between a client application with user interface and RESTful Web service, the loose coupling between client and server can be enhanced by providing a link.

Following are the reasons for enhancing the coupling between client and server.

Sr. No.	Reasons to enhance the loose coupling
(1)	Client application needs to know only the logical name of the link in advance. Hence, it is not necessary to know the specific URI for calling REST API.
(2)	Since it is not necessary for the client application to know the specific URI, impact on server, owing to change in URI, can be minimized.

Whether to provide a hypermedia link (URI) to other resources should be determined on considering all the points described above.

Tip: Relation with HATEOAS

HATEOAS is an abbreviation for "**H**ypermedia **A**s **T**he **E**ngine **O**f **A**pplication **S**tate" and is one of the architectures for creating a RESTful Web application.

HATEOAS architecture includes the following processes.

- In the resources (JSON or XML) that are exchanged between client and server, the server includes a hypermedia link (URI) to an accessible resource.
- Client fetches required resources from the server through the hypermedia link in the resource display (JSON or XML), and changes application status (screen status etc.).

Therefore, providing a link for related resources is consistent with the HATEOAS architecture.

When loose coupling between server and client is to be enhanced, please review if using the HATEOAS architecture would be beneficial.

5.1.3 How to design

This section explains the design of RESTful Web Service.

Resource extraction

First, the resource published on the Web is extracted.

Precautions while extracting a resource are as given below.

Sr. No.	Precautions while extracting a resource
(1)	Resource published on the web is used as the information managed by database. However, **data model of the database must not be published as resource as it is, without careful consideration.** It should be closely investigated, as the fields stored in the database may include some fields that should not be disclosed to the client.
(2)	**When information type is different in spite of being managed by the same table of the database, publishing it as a separate resource may be considered.** There are cases wherein, even if essentially seen as different information, it is managed by the same table, due to same data structure. Hence, such cases need to be reviewed closely.
(3)	In RESTful Web Service, the information operated by an event is extracted as a resource. **The event itself should not be extracted as a resource.** For example, when creating RESTful Web Service to be called from the events (approve, deny, return etc.) generated by work flow functionality, information for managing the workflow status or the workflow itself, is extracted as a resource.

Assigning URI

URI is assigned to the extracted resource for identifying it.

It is recommended to use following formats for the URI.

- `http(s)://{Domain name (:Port number)}/{A value indicating REST API}/{API version}/{path for identifying a resource}`
- `http(s)://{Domain name indicating REST API(:Port number)}/{API version}/{path for identifying a resource}`

A typical example is given below.

- `http://example.com/api/v1/members/M000000001`
- `http://api.example.com/v1/members/M000000001`

Assigning a URI that indicates the API as REST API

It is recommended to include `api` within the URI domain or path, to clearly indicate that the URI is intended for RESTful Web Service (REST API).

Typically, the URI is as given below.

- `http://example.com/api/...`
- `http://api.example.com/...`

Assigning a URI for identifying the API version

It is recommended to include a value that identifies the API version, in the URI to be published to the client, since it may be necessary to run RESTful Web Service in multiple versions.

Typically, the URI format is as follows.

- `http://example.com/api/{API version}/{path for identifying a resource}`
- `http://api.example.com/{API version}/{path for identifying a resource}`

Todo

TBD

Whether API version should be included in URI, is currently being investigated.

Assigning a path for identifying resource

The 2 URLs given below are assigned for resources that are published on Web.
Following is an example of a URI when publishing member information on Web.

Sr. No.	URI format	Typical example of URI	Description
(1)	/{Noun that represents collection of resources}	/api/v1/members	It is the URI used for batch operations of resources.
(2)	/{Noun that represents collection of resources/resource identifier (ID etc)}	/api/v1/members/M0001	It is the URI used while operating a specific resource.

The URI for related resources published on Web are nested and then displayed.
Following example describes the URI for publishing order information for each member on the Web.

Sr. No.	URI format	Typical example of URI	Description
(3)	{Resource URI}/{Noun representing collection of related resources}	/api/v1/members/M0001/orders	It is the URI used at the time of batch operation of related resources.
(4)	{Resource URI}/{Noun representing collection of related resources}/{Identifier for related resource (ID etc.)}	/api/v1/members/M0001/orders/O0001	It is the URI used when operating a specific related resource.

When the related resource published on Web has a single element, the noun that indicates the related resource should be singular and not plural.

Following is the example of URI to publish credentials of each member on Web.

Sr. No.	URI format	Typical example of URI	Description
(5)	{URI for resource}/{Noun representing related resource}	/api/v1/members/M0001/credential	It is the URI used when operating a related resource with single element.

Assigning HTTP methods

CRUD operation for resources is published as REST API by assigning the following HTTP methods for the URI assigned to each resource.

Assigning HTTP methods for resource collection URI

Sr. No.	HTTP methods	Overview of the REST API to be implemented
(1)	GET	REST API that fetches collection of resources specified in URI, is implemented.
(2)	POST	REST API that creates and adds the specified resource to the collection is implemented.
(3)	PUT	REST API that performs batch update for resource specified in URI is implemented.
(4)	DELETE	REST API that performs batch deletion for resource specified in URI is implemented.
(5)	HEAD	REST API that fetches meta information of the resource collection specified in URI, is implemented. A process same as GET is performed and only header is sent as response.
(6)	OPTIONS	REST API that responds with the list of HTTP methods (API) supported by resource collection specified in URI, is implemented.

Assigning HTTP methods for URI of specific resources

Sr. No.	HTTP methods	Overview of REST API to be implemented
(1)	GET	REST API that fetches the resource specified in URI is implemented.
(2)	PUT	REST API that creates or updates the resource specified in URI is implemented.
(3)	DELETE	REST API that deletes the resource specified in URI is implemented.
(4)	HEAD	A REST API that fetches meta information of the resource specified in URI is implemented. A process same as GET is performed and only header is sent as a response.
(5)	OPTIONS	REST API that responds with list of HTTP methods (API) supported by the resource specified in URI is implemented.

Resource format

It is recommended to use JSON as the format for displaying a resource.

The explanation hereafter is based on the assumption that JSON is used as the format for displaying a resource.

JSON Field name

It is recommended to use "lower camel case" as the JSON field name.

It is recommended on considering its compatibility with JavaScript, which is assumed as one of the client applications.

The JSON sample with field name set in “lower camel case”, is as given below.

In “lower camel case”, first letter of the word is in lowercase and subsequent first letters of words are in uppercase.

```
{
    "memberId" : "M000000001"
}
```

NULL and blank characters

It is recommended to differentiate NULL and blank characters as JSON values.

Although, as application process, NULL and blank characters are often equated, it is advisable to differentiate NULL and blank characters as the value to be set in JSON.

JSON sample wherein NULL and blank characters are differentiated, is given below.

```
{
    "dateOfBirth" : null,
    "address1" : ""
}
```

Date format

It is recommended to use extended ISO-8601 format as the JSON date field format.

Format other than extended ISO-8601 format can be used. However, it is advisable to use the extended ISO-8601 format, if there is no particular reason otherwise.

There are two formats in ISO-8601 namely, basic format and extended format, however, readability is higher in extended format.

Basically, there are following three formats.

1. yyyy-MM-dd

```
{
    "dateOfBirth" : "1977-03-12"
}
```

2. yyyy-MM-dd'T'HH:mm:ss.SSSZ

```
{
    "lastModifiedAt" : "2014-03-12T22:22:36.637+09:00"
}
```

3. yyyy-MM-dd'T'HH:mm:ss.SSS'Z' (format for UTC)

```
{
    "lastModifiedAt" : "2014-03-12T13:11:27.356Z"
}
```

Hypermedia link format

It is recommended to use the following format to create a hypermedia link.

Sample of recommended format is as given below.

```
{
    "links" : [
        {
            "rel" : "ownerMember",
            "href" : "http://example.com/api/v1/memebers/M000000001"
        }
    ]
}
```

- Link object consisting of 2 fields - `"rel"` and `"href"` is retained in collection format.
- Link name for identifying the link is specified in `"rel"`.
- URI to access the resource is specified in `"href"`.
- `"links"` is the field which retains the Link object in collection format.

Format at the time of error response

When an error is detected, it is recommended to use a format that can retain the details of the error occurred.

Detailed error information should be included, especially when there is a possibility of the error being eliminated owing to re-operation by client.

In contrast, detailed error information should not be included when an event that exposes system vulnerability occurs. In such cases, the detailed error information should be output to a log.

Following is an example of the response format when error is detected.

```
{
  "code" : "e.ex.fw.7001",
  "message" : "Validation error occurred on item in the request body.",
  "details" : [ {
    "code" : "ExistInCodeList",
    "message" : "\"genderCode\" must exist in code list of CL_GENDER.",
    "target" : "genderCode"
  } ]
}
```

In the above example,

- Error code (code)
- Error message (message)
- Error details list (details)

are provided as error response formats.

It is assumed that the error details list is used when input validation error occurs. It is a format that can retain details like the field in which the error occurred and the information of the error.

HTTP Status Code

HTTP status code is sent as the response, in accordance with the following guidelines.

Sr. No.	Objectives
(1)	When the request is successful, an HTTP status code indicating success or transfer (2xx or 3xx system) is sent as response.
(2)	When the cause of request failure lies at client side, an HTTP status code indicating client error (4xx system) is sent as the response. When client is not responsible for request failure however, when the request may be successful through a re-operation by client, it is still considered as client error.
(3)	When the cause of request failure lies at server side, an HTTP status code indicating server error (5xx system) is sent as the response.

HTTP status codes when the request is successful

When the request is successful, following HTTP status codes are sent as responses, depending on status.

Sr. No.	HTTP Status codes	Description	Applicable conditions
(1)	200 OK	HTTP status code notifying that the request was successful.	It is sent as a response when the resource information corresponding to the request is output in the entity body of response, as a result of successful request,
(2)	201 Created	HTTP status code notifying the creation of a new resource.	It is used when a new resource is created using POST method. URI for created resource is set in the Location header of the response.
(3)	204 No Content	HTTP status code notifying a successful request.	It is sent as a response when the resource information corresponding to request is not output in the entity body of response, as a result of successful request.

Tip: The difference between "`200 OK` and "`204 No Content`" is whether the resource information is output/not output in the response body.

HTTP status code when the cause of request failure lies at client side

When the cause of request failure lies at client side, following HTTP status codes are sent as responses depending on the status.

Status codes that must be identified by individual REST APIs handling the resources, are as given below.

Sr. No.	HTTP Status code	Description	Applicable conditions
(1)	400 Bad Request	HTTP status code notifying that the request syntax or requested value is incorrect.	It is sent as a response when an incomplete JSON or XML format specified in entity body is detected or an incomplete input value is specified in JSON or XML format or in the request parameters.
(2)	404 Not Found	HTTP status code notifying that the specified resource does not exist.	It is sent as a response when resource corresponding to specified URI does not exist in the system.
(3)	409 Conflict	HTTP status code notifying that the process is terminated due to conflict in resource status when the request status is changed by requested contents.	It is sent as a response when an exclusive error or a business error is detected. Conflict details and error details required to resolve the conflict need to be output to the entity body.

Following status codes need not be identified by individual REST APIs which handle the resources.

These status codes need to be identified as the framework or common process.

Sr. No.	HTTP Status codes	Description	Applicable conditions
(4)	405 Method Not Allowed	HTTP status code notifying that the used HTTP method is not supported by the specified resource.	It is sent as a response when an unsupported HTTP method is used. The list of allowed methods is set in the Allow header of response.
(5)	406 Not Acceptable	HTTP status code notifying the inability to receive a request, as the resource status cannot be sent as a response in the specified format.	It is sent as a response when, the format specified in extension or Accept header is not supported as a response format.
(6)	415 Unsupported Media Type	HTTP status code notifying that the request cannot be received, as the format specified in entity body is not supported.	It is sent as a response when an unsupported format is specified in Content-Type header, as request format.

HTTP status code when the cause of request failure lies at server side

When the cause of request failure lies at server side, HTTP status codes given below are sent as responses, depending on the status.

Sr. No.	HTTP Status code	Description	Applicable conditions
(1)	500 Internal Server Error	HTTP status code notifying that an internal error has occurred in the server.	It is sent as a response when an unexpected error has occurred in the server or a status that should not occur during a normal operation is detected.

Authentication and Authorization

Please refer to : ref: *OAuth <OAuth>* for the mechanism of authentication and authorization using the OAuth 2 mechanism.

Conditional update control of resource

Todo

TBD

The process for conditional update (exclusive control) of a resource using HTTP header is explained here.

Conditional update using headers like Etag/Last-Modified-Since etc. will be described in subsequent versions.

Conditional acquisition control of resource

Todo

TBD

The process for conditional acquisition (304 not modified control) of resource using HTTP header is explained here.

Conditional acquisition using headers like Etag/Last-Modified etc. will be described in subsequent versions.

Cache control of resource

Todo

TBD

Cache control of resources which use HTTP header, is explained here.

Cache control of resources that use headers such as Cache-Control/Pragma/Expires etc. shall be described in subsequent versions.

Versioning

Todo

TBD

Version control of RESTful Web Service and details on performing parallel operations in multiple versions, will be described in subsequent versions.

5.1.4 How to use

This section explains the basic method to create RESTful Web Service.

Web application configuration

While building RESTful Web Service, Web application (war) is built by any one of the following configurations.

It is recommended to build a Web application that is exclusive to RESTful Web Service unless there is a specific reason otherwise.

Sr. No.	Configuration	Description
(1)	Build an exclusive Web application for RESTful Web Service.	It is recommended to build an exclusive Web application (war) for RESTful Web Service when an independence with client application (UI layer application) that uses RESTful Web Service, is to be ensured (is necessary). This method can be used to create RESTful Web Service when there are multiple client applications using RESTful Web Service.
(2)	Build by providing `DispatcherServlet` for RESTful Web Service.	When it is not necessary to ensure independence of client application (UI layer application) that uses RESTful Web Service, both the client application and RESTful Web Service can be built as a single Web application (war). However, it is strongly recommended to build it by dividing `DispatcherServlet` that receives the requests for RESTful Web Service and `DispatcherServlet` that receives client application requests.

Note: Client application (UI layer application)

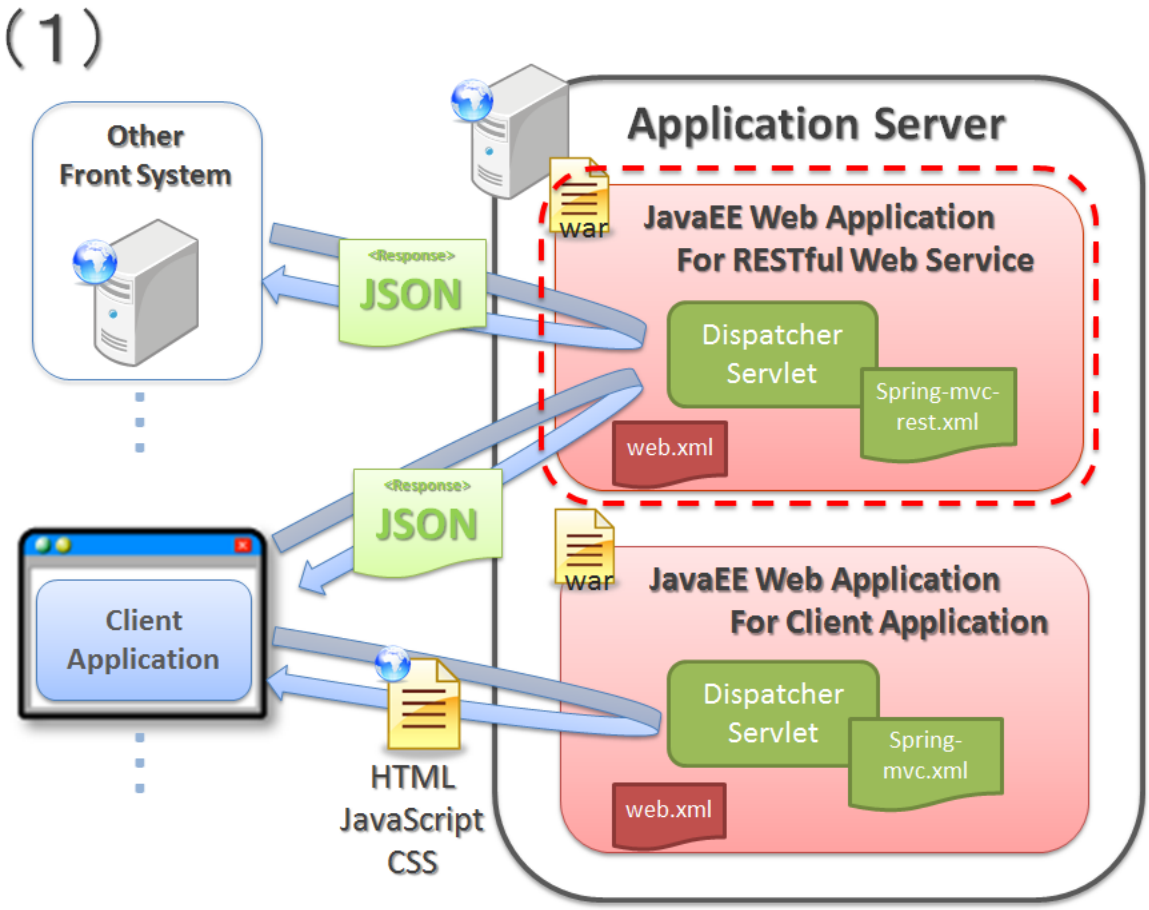
Client application (UI layer application) described here refers to the application that responds with client layer (UI layer) component called CSS (Cascading Style Sheets) and scripts like HTML, JavaScript etc. HTML generated by template engine such as JSP, is also considered.

Note: Why division of DispatcherServlet is recommended

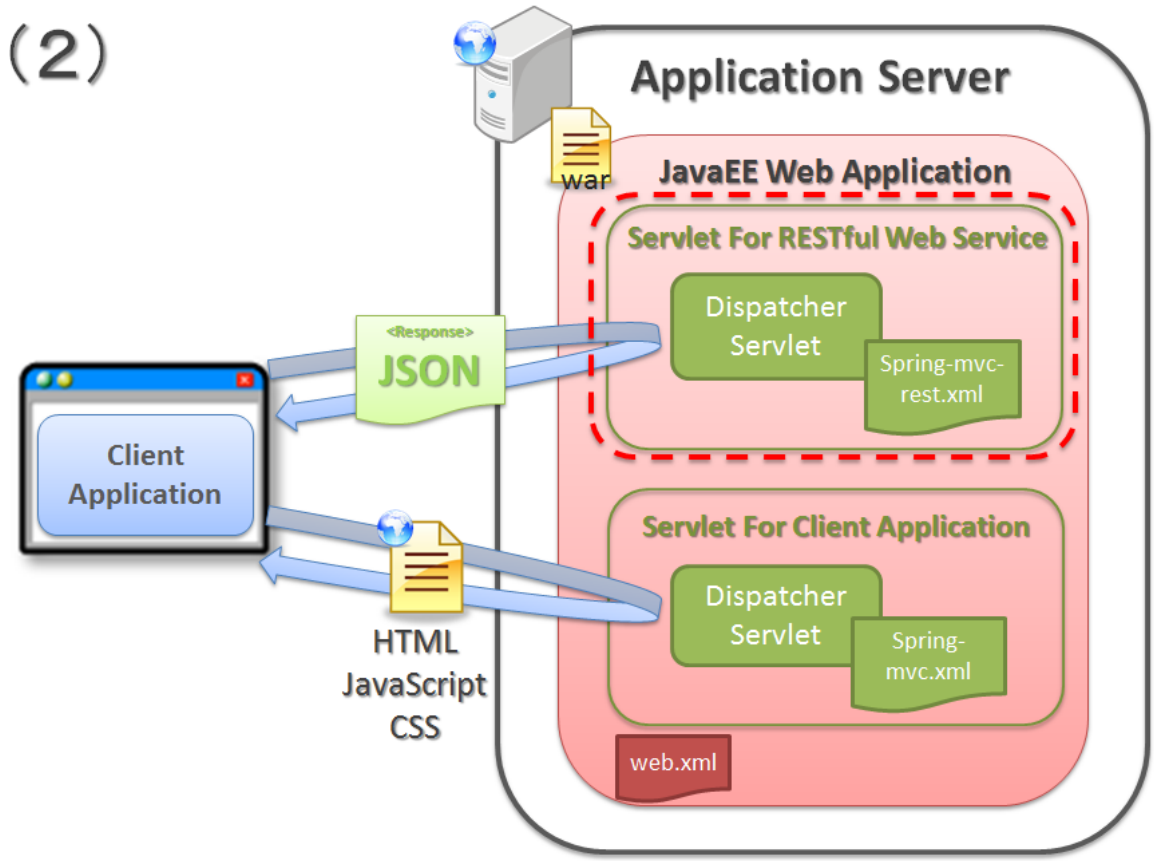
In Spring MVC, operation settings of the application are defined for each `DispatcherServlet`. Therefore, when the requests of RESTful Web Service and client application (UI layer application) are configured to be received from the same `DispatcherServlet`, specific operation settings for RESTful Web Service or client application cannot be defined, thus resulting in complex or cumbersome settings.

In this guideline, when RESTful Web Service and client application are to be configured as same Web application, it is recommended to divide `DispatcherServlet` to avoid occurrence of the issues described above.

Configuration image when building a Web application exclusive to RESTful Web Service, is as follows:



Configuration image when building RESTful Web Service and client application as a single application, is as follows:



pom.xml configuration

Configuration of dependency relation is not required if terasoluna-gfw-common-dependencies is used.

Warning: Configuration while using in Java SE 7 environment

terasoluna-gfw-common-dependencies configure a dependency relation considering Java SE 8 as a prerequisite. Java SE 8 dependency library is excluded as shown below while using in Java SE 7 environment. For

java SE 8 dependency library, refer "*OSS Versions* " of Architectural Overview.

```
<dependency>
    <groupId>org.terasoluna.gfw</groupId>
    <artifactId>terasoluna-gfw-common-dependencies</artifactId>
    <exclusions>
        <exclusion>
            <groupId>com.fasterxml.jackson.datatype</groupId>
            <artifactId>jackson-datatype-jsr310</artifactId>
        </exclusion>
    </exclusions>
</dependency>
```

Application settings

Application settings for RESTful Web Service are explained below.

Warning: DoS attack measures at the time of StAX(Streaming API for XML) use

If the StAX is used to parse the XML format data, protect DoS attack. For details, refer to CVE-2015-3192 - DoS Attack with XML Input.

Settings for activating the Spring MVC components necessary for RESTful Web Service

Create a bean definition file for RESTful Web Service.

Definitions that are required to operate the sample indicated in the explanation hereafter, are as follows:

- `spring-mvc-rest.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:mvc="http://www.springframework.org/schema/mvc"
    xmlns:util="http://www.springframework.org/schema/util"
    xmlns:aop="http://www.springframework.org/schema/aop"
    xsi:schemaLocation="
        http://www.springframework.org/schema/mvc
        http://www.springframework.org/schema/mvc/spring-mvc.xsd
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/util
        http://www.springframework.org/schema/util/spring-util.xsd
        http://www.springframework.org/schema/context
```

```
        http://www.springframework.org/schema/context/spring-context.xsd
        http://www.springframework.org/schema/aop
        http://www.springframework.org/schema/aop/spring-aop.xsd
">

    <!-- Load properties files for placeholder. -->
    <!-- (1) -->
    <context:property-placeholder
        location="classpath*:/META-INF/spring/*.properties" />

    <bean id="jsonMessageConverter"
        class="org.springframework.http.converter.json.
↪MappingJackson2HttpMessageConverter">
        <property name="objectMapper" ref="objectMapper" />
    </bean>

    <bean id="objectMapper" class="org.springframework.http.converter.json.
↪Jackson2ObjectMapperFactoryBean">
        <!-- (2) -->
        <property name="dateFormat">
            <bean class="com.fasterxml.jackson.databind.util.StdDateFormat" />
        </property>
    </bean>

    <!-- Register components of Spring MVC. -->
    <!-- (3) -->
     <mvc:annotation-driven>
        <mvc:message-converters register-defaults="false">
            <ref bean="jsonMessageConverter" />
        </mvc:message-converters>
        <!-- (4) -->
        <mvc:argument-resolvers>
            <bean class="org.springframework.data.web.
↪PageableHandlerMethodArgumentResolver" />
        </mvc:argument-resolvers>
    </mvc:annotation-driven>

    <!-- Register components of interceptor. -->
    <!-- (5) -->
    <mvc:interceptors>
        <mvc:interceptor>
            <mvc:mapping path="/**" />
            <bean class="org.terasoluna.gfw.web.logging.
↪TraceLoggingInterceptor" />
        </mvc:interceptor>
        <!-- omitted -->
    </mvc:interceptors>

    <!-- Scan & register components of RESTful Web Service. -->
    <!-- (6) -->
```

```
    <context:component-scan base-package="com.example.project.api" />

    <!-- Register components of AOP. -->
    <!-- (7) -->
    <bean id="handlerExceptionResolverLoggingInterceptor"
        class="org.terasoluna.gfw.web.exception.
↪HandlerExceptionResolverLoggingInterceptor">
        <property name="exceptionLogger" ref="exceptionLogger" />
    </bean>
    <aop:config>
        <aop:advisor advice-ref="handlerExceptionResolverLoggingInterceptor"
            pointcut="execution(* org.springframework.web.servlet.
↪HandlerExceptionResolver.resolveException(..))" />
    </aop:config>

</beans>
```

Sr. No.	Description
(1)	When the value defined in the property file needs to be referred by an application layer component, the property file should be read by using `<context:property-placeholder>` element. For the details of fetching a value from property file, refer to "*Properties Management*".
(2)	Add the settings for handling the JSON date field format as extended ISO-8601 format. Also, when Joda Time class is to be used as a property of JavaBean which represents a resource (Resource class), "*Configuration while using JSR-310 Date and Time API / Joda Time*" must be carried out.
(3)	Perform bean registration for the Spring MVC framework component necessary for providing RESTful Web Service. JSON can be used as a resource format by performing these settings. In the above example, resource format is restricted to JSON since the register-defaults attribute of `<mvc:message-converters>` element is set as `false`. To use XML as resource format, `MessageConverter` for XML, that performs the XXE Injection countermeasure, should be specified. For details on designated methods, refer to "*Enabling XXE Injection measures*" .

次のページに続く

Table. 5.3 – 前のページからの続き

Sr. No.	Description
(4)	Add the settings to enable page search functionality. For page search details, refer to "*Pagination*". This setting is not required if page search is unnecessary, however, it is alright if defined.
(5)	Perform bean registration for Spring MVC interceptor. In the above example, only the `TraceLoggingInterceptor` provided by common library is defined. However, when using JPA as data access, `OpenEntityManagerInViewInterceptor` setting needs to be added separately. Refer to *Database Access (JPA)* for `OpenEntityManagerInViewInterceptor`.
(6)	Scan the application layer components for RESTful Web Service (Controller or Helper class etc.) and perform bean registration. The `"com.example.project.api"` part is the **package name for each project.**
(7)	Specify AOP definition to output the exception handled by Spring MVC framework to a log. Refer to *Exception Handling* for `HandlerExceptionResolverLoggingInterceptor`.

Note: How to define a Bean for ObjectMapper

When a Bean is to be defined for `com.fasterxml.jackson.databind.ObjectMapper` of Jackson, `Jackson2ObjectMapperFactoryBean` provided by Spring should be used. If `Jackson2ObjectMapperFactoryBean` is used, extension module for JSR-310 Date and Time API or Joda Time can be auto-registered. Further, `ObjectMapper` configuration which is difficult to represent in Bean definition file of XML can be easily configured as well.

Also, when style is to be changed from directly defining a Bean for `ObjectMapper` to using `Jackson2ObjectMapperFactoryBean`, it should be noted that default value for the following configuration differs from the default value of Jackson (disabled).

- MapperFeature#DEFAULT_VIEW_INCLUSION
- DeserializationFeature#FAIL_ON_UNKNOWN_PROPERTIES

When `ObjectMapper` operation and default Jackson operation are to be matched, the configuration above is enabled using `featuresToEnable` property.

```
<bean id="objectMapper" class="org.springframework.http.converter.json.
↪Jackson2ObjectMapperFactoryBean">
    <!-- ... -->
    <property name="featuresToEnable">
        <array>
            <util:constant static-field="com.fasterxml.jackson.databind.
↪MapperFeature.DEFAULT_VIEW_INCLUSION"/>
            <util:constant static-field="com.fasterxml.jackson.databind.
↪DeserializationFeature.FAIL_ON_UNKNOWN_PROPERTIES"/>
        </array>
    </property>
</bean>
```

For `Jackson2ObjectMapperFactoryBean` details, refer JavaDoc of Jackson2ObjectMapperFactoryBean.

Note: Points to be noted when changing the jackson version from 1.x.x to 2.x.x

- Changed package

version	package
1.x.x	*org.codehaus.jackson*
2.x.x	*com.fasterxml.jackson*

- Please note that configuration of subordinate package also is also changed.

- Deprecated List
- http://fasterxml.github.io/jackson-core/javadoc/2.8/deprecated-list.html
- http://fasterxml.github.io/jackson-databind/javadoc/2.8/deprecated-list.html
- http://fasterxml.github.io/jackson-annotations/javadoc/2.8/deprecated-list.html

Servlet settings for RESTful Web Service

When RESTful Web Service and client application are built as separate Web applications, the settings are as follows:

When RESTful Web Service and client application are to be built as same Web application, it is necessary to perform "*Settings when RESTful Web Service and client application are operated as the same Web application*" .

- `web.xml`

```
<!-- omitted -->

<servlet>
    <!-- (1) -->
    <servlet-name>restAppServlet</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-
↪class>
    <init-param>
        <param-name>contextConfigLocation</param-name>
        <!-- (2) -->
        <param-value>classpath*:META-INF/spring/spring-mvc-rest.xml</param-
↪value>
    </init-param>
```

```
    <load-on-startup>1</load-on-startup>
</servlet>
<!-- (3) -->
<servlet-mapping>
    <servlet-name>restAppServlet</servlet-name>
    <url-pattern>/api/v1/*</url-pattern>
</servlet-mapping>

<!-- omitted -->
```

Sr. No.	Description
(1)	Specify a name which shows that it is a RESTful Web Service servlet, in `<servlet-name>` element. In the above example, `"restAppServlet"` is specified as the servlet name.
(2)	Specify a Spring MVC bean definition file used to build `DispatcherServlet` for RESTful Web Service. In the above example, `META-INF/spring/spring-mvc-rest.xml` in class path, is specified as the Spring MVC bean definition file.

(3)	Specify a servlet path pattern to be mapped with the `DispatcherServlet` of RESTful Web Service.
	In the above example, the servlet path under `"/api/v1/"` is mapped with the `DispatcherServlet` for RESTful Web Service.
	Typically, servlet paths like
	`"/api/v1/"`
	`"/api/v1/members"`
	`"/api/v1/members/xxxxx"`
	are mapped in the `DispatcherServlet("restAppServlet")` for RESTful Web Service.

Tip: Value specified in the value attribute of @RequestMapping annotation

For the value to be specified in value attribute of `@RequestMapping` annotation, specify the value assigned to the part of wild card (`*`) in `<url-pattern>` element.

For example, when `@RequestMapping(value = "members")` is specified, it is deployed as the

method to perform a process for path `"/api/v1/members"`. Therefore, it is not necessary to specify the path (`"api/v1"`) in value attribute of `@RequestMapping` annotation for mapping to divided servlets.

When `@RequestMapping(value = "api/v1/members")` is specified, it gets deployed as the method that performs a process for the `"/api/v1/api/v1/members"` path. Hence, please take note of same.

REST API implementation

How to implement REST API is explained here.

Hereafter, REST API implementation example is explained using member information (Member resource) of a shopping site.

Note: Domain layer implementation is not explained in this section, however, it is sent as attachment "*Source code of the domain layer class created at the time of REST API implementation*".

Please refer if required.

REST API specifications used in this explanation are as shown below.

Resource format

The resource format of member information should be the following JSON format.

In the following example, although all the fields are displayed, they are not used in the requests and responses of all API.

For example, `"password"` is used only in requests whereas `"createdAt"` or `"lastModifiedAt"` are used only in responses.

```
{
    "memberId" : "M000000001",
    "firstName" : "Firstname",
    "lastName" : "Lastname",
    "genderCode" : "1",
    "dateOfBirth" : "1977-03-13",
    "emailAddress" : "user1@test.com",
    "telephoneNumber" : "09012345678",
    "zipCode" : "1710051",
    "address" : "Tokyo",
    "credential" : {
        "signId" : "user1@test.com",
        "password" : "zaq12wsx",
        "passwordLastChangedAt" : "2014-03-13T04:39:14.831Z",
        "lastModifiedAt" : "2014-03-13T04:39:14.831Z"
    },
    "createdAt" : "2014-03-13T04:39:14.831Z",
    "lastModifiedAt" : "2014-03-13T04:39:14.831Z"
}
```

Note: This section illustrates an example wherein a hypermedia link for related resource is not provided. For details on implementation with hypermedia link, refer to "*Implementing hypermedia link*".

Specifications of resource fields

The specifications for each field of a resource (JSON) are as shown below.

Sr. No.	Item name	Type	I/O specifications	Number of digits (min-max)	Other specifications
(1)	memberId	String	I/O	10-10	It should be "Unspecified" (NULL) at the time of request for POST Members.
(2)	firstName	String	I/O	1-128	-

次のページに続く

Table. 5.4 – 前のページからの続き

Sr. No.	Item name	Type	I/O specifications	Number of digits (min-max)	Other specifications
(3)	lastName	String	I/O	1-128	-
(4)	genderCode	String (Code)	I/O	1-1	"0" : UNKNOWN "1" : MEN "2" : WOMEN
(5)	dateOfBirth	Date	I/O	-	yyyy-MM-dd format (extended ISO-8601 format)
(6)	emailAddress	String (E-mail)	I/O	1-256	-
(7)	telephoneNumber	String	I/O	0-20	-
(8)	zipCode	String	I/O	0-20	-
(9)	address	String	I/O	0-256	-
(10)	credential	Object (MemberCredential)	I/O	-	It is specified at the time of request for POST Members.
(11)	credential/signId	String (E-mail)	I/O	0-256	emailAddress value is applied when not specified.

次のページに続く

Table. 5.4 – 前のページからの続き

Sr. No.	Item name	Type	I/O specifications	Number of digits (min-max)	Other specifications
(12)	credential/ password	String	I	8-32	-
(13)	credential/ passwordLastChangedAt	Timestamp with time-zone	O	-	yyyy-MM-dd'T'HH:mm:ss.SSS'Z' format (extended ISO-8601 format)
(14)	credential/ lastModifiedAt	Timestamp with time-zone	O	-	yyyy-MM-dd'T'HH:mm:ss.SSS'Z'format (extended ISO-8601 format)
(15)	createdAt	Timestamp with time-zone	O	-	yyyy-MM-dd'T'HH:mm:ss.SSS'Z' format (extended ISO-8601 format)
(16)	lastModifiedAt	Timestamp with time-zone	O	-	yyyy-MM-dd'T'HH:mm:ss.SSS'Z' format (extended ISO-8601 format)

REST APIs List

APIs given below are used as the REST API to be implemented.

Sr. No.	API name	HTTP Method	Resource path	Status Code	API Overview
(1)	*GET Members*	GET	`/api/v1/members`	200 (OK)	Page is searched for Member resource that matches the condition.
(2)	*POST Members*	POST	`/api/v1/members`	201 (Created)	One Member resource is created.
(3)	*GET Member*	GET	`/api/v1/members/{memberId}`	200 (OK)	One Member resource is fetched.
(4)	*PUT Member*	PUT	`/api/v1/members/{memberId}`	200 (OK)	One Member resource is updated.
(5)	*DELETE Member*	DELETE	`/api/v1/members/{memberId}`	204 (No Content)	One Member resource is deleted.

Note: From Spring Framework 4.3, REST API for HEAD and OPTIONS methods is now implicitly provided, so developers do not need to explicitly implement these REST APIs.

Since the OPTIONS itself is not included in the Allow header that the implicitly prepared REST API for OPTIONS responds to, please note that it differs from the implementation example introduced in the development guidelines up to TERASOLUNA Server Framework for Java 5.2.x.

Creating REST API packages

Create a package to store REST API class.

It is recommended to assign `api` as the package name for the root package that stores REST API class and to create a package for each resource (lower case of resource name) under the same.
Resource name in the explanation is `Member`. Hence, the package name is `org.terasoluna.examples.rest.api.member`.

Note: Usually, following 4 types of classes are stored in the created package. It is recommended to use the following naming rules for name of the class to be created.

- `[Resource name]Resource`
- `[Resource name]RestController`
- `[Resource name]Validator` (created when required)
- `[Resource name]Helper` (created when required)

In the explanation, name of the resource is `Member`. As a result, the respective names will be as below.

- `MemberResource`
- `MemberRestController`
- `MemberValidator`
- `MemberHelper`

When handling a related resource, it is advisable to place the class for related resource also in the same package.

It is recommended to create a package named `common` that stores common parts for REST API just under the root package that stores the REST API class and to create sub packages at functionality level.

For example, a sub package that stores common parts which perform error handling is created with the name `error`.

The class for exception handling created in the subsequent explanation, is stored in a package called `org.terasoluna.examples.rest.api.common.error`.

Note: As long as it is clear that the package is storing common parts, it can have a name other than `common`.

Creating Resource class

In this guideline, it is recommended to provide a Resource class as the class representing the resource published on Web (represent JSON or XML).

Note: Reasons for creating a Resource class

The reason for creating a Resource class regardless of DomainObject class (for example, Entity class) being available is, user interface information (UI) which is used in the I/O with client and information handled by business process do not necessarily match.

If these are mixed and then used, the application layer may affect the domain layer, resulting in deteriorated maintainability. It is recommended to create the DomainObject and Resource class separately and convert data by using BeanMapper like Dozer etc.

Role of Resource class is as follows:

Sr. No.	Roles	Description
(1)	To define the data structure of a resource.	Define a data structure of the resource published on Web. Generally, it is very rare to publish the data structure managed by persistence layer of database etc. as it is, as a resource on Web.
(2)	To define format.	Specify a definition related to resource format using annotation. Annotation to be used differs according to the resource format (JSON/XML etc.). Jackson annotation is used for JSON format whereas JAXB annotation is used for XML format.
(3)	To define input validation rules.	Specify input validation rules for single item of each field by using Bean Validation annotation. For input validation details, refer to "*Input Validation*" .

Warning: Measures to circular reference

When you serialize a Resource class (JavaBean) in JSON or XML format and if property holds an object of cross reference relationship, the `StackOverflowError` and `OutOfMemoryError` occur due to circular reference, hence it is necessary to exercise caution.

In order to avoid a circular reference,

- `@com.fasterxml.jackson.annotation.JsonIgnore` annotation to exclude the property from serialization in case of serialized in JSON format using the Jackson
- `@javax.xml.bind.annotation.XmlTransient` annotation to exclude the property from serialization in case of serialized in XML format using the JAXB

can be added.

An example to exclude specific field from serialization while serializing in JSON format using Jackson is given below.

```
public class Order {
    private String orderId;
    private List<OrderLine> orderLines;
    // ...
}
```

```
public class OrderLine {
    @JsonIgnore
    private Order order;
    private String itemCode;
    private int quantity;
    // ...
}
```

Sr. No.	Description
(1)	Add `@JsonIgnore` annotation to exclude the property from serialization.

Example of Resource class creation is shown below.

- `MemberResource.java`

```
package org.terasoluna.examples.rest.api.member;

import java.io.Serializable;

import javax.validation.Valid;
import javax.validation.constraints.NotNull;
import javax.validation.constraints.Null;
import javax.validation.constraints.Past;
import javax.validation.constraints.Size;

import org.hibernate.validator.constraints.Email;
import org.hibernate.validator.constraints.NotEmpty;
import org.joda.time.DateTime;
import org.joda.time.LocalDate;
import org.springframework.format.annotation.DateTimeFormat;
import org.terasoluna.gfw.common.codelist.ExistInCodeList;

// (1)
public class MemberResource implements Serializable {

    private static final long serialVersionUID = 1L;

    // (2)
    interface PostMembers {
    }
```

```
interface PutMember {
}

@Null(groups = PostMembers.class)
@NotEmpty(groups = PutMember.class)
@Size(min = 10, max = 10, groups = PutMember.class)
private String memberId;

@NotEmpty
@Size(max = 128)
private String firstName;

@NotEmpty
@Size(max = 128)
private String lastName;

@NotEmpty
@ExistInCodeList(codeListId = "CL_GENDER")
private String genderCode;

@NotNull
@Past
private LocalDate dateOfBirth;

@NotEmpty
@Size(max = 256)
@Email
private String emailAddress;

@Size(max = 20)
private String telephoneNumber;

@Size(max = 20)
private String zipCode;

@Size(max = 256)
private String address;

@NotNull(groups = PostMembers.class)
@Null(groups = PutMember.class)
@Valid
// (3)
private MemberCredentialResource credential;

@Null
private DateTime createdAt;

@Null
private DateTime lastModifiedAt;
```

```
    // omitted setter and getter

}
```

Sr. No.	Description
(1)	JavaBean representing the Member resource.
(2)	The interface for specifying validation group of Bean Validation is defined. In this implementation, input validation is grouped, as different input validations are performed for POST and PUT methods. Refer to "*Input Validation*" for grouped validation.
(3)	JavaBean with nested related resource is defined in the field. In this implementation, the member credentials (sign ID and password) are handled as related resources. These are extracted as related resources by considering operations that carry out only the sign ID change and password change. However, REST API implementation for related resources is omitted here.

- `MemberCredentialResource.java`

```
package org.terasoluna.examples.rest.api.member;

import java.io.Serializable;

import javax.validation.constraints.NotNull;
import javax.validation.constraints.Null;
import javax.validation.constraints.Size;

import com.fasterxml.jackson.annotation.JsonInclude;
import org.hibernate.validator.constraints.Email;
import org.joda.time.DateTime;

// (4)
public class MemberCredentialResource implements Serializable {

    private static final long serialVersionUID = 1L;

    @Size(max = 256)
```

```
    @Email
    private String signId;

    // (5)
    @JsonInclude(JsonInclude.Include.NON_NULL)
    @NotNull
    @Size(min = 8, max = 32)
    private String password;

    @Null
    private DateTime passwordLastChangedAt;

    @Null
    private DateTime lastModifiedAt;

    // omitted setter and getter

}
```

Sr. No.	Description
(4)	JavaBean that represents Credential resource which is the related resource of Member resource.
(5)	An annotation is specified so that the field itself is not output in JSON when the value is `null`. It is specified so that the field 'password' is not output in responding JSON. In the above example, it is restricted to (`Inclusion.NON_NULL`) for NULL value, however it can also be specified as (`Inclusion.NON_EMPTY`) in case of an empty value.

- Adding the mapping definition of Bean

  In the subsequent implementations, Entity class and Resource class are copied by using "*Bean Mapping (Dozer)*" .

  Joda-Time classes namely, `org.joda.time.DateTime` and `org.joda.time.LocalDate` are included in the JavaBean shown above. However, Joda-Time objects are not correctly copied if "*Bean Mapping (Dozer)*" is used for copying.

  Therefore, it is necessary to apply "*How to copy Joda-Time classes using Dozer*" to copy the objects correctly.

Creating Controller class

Create controller class for each resource.

Refer to *Appendix* for the source code when implementation of all APIs is completed.

```
package org.terasoluna.examples.rest.api.member;

// omitted
import org.springframework.web.bind.annotation.RestController;
// omitted

@RequestMapping("members") // (1)
@RestController // (2)
public class MemberRestController {

    // omitted ...

}
```

Sr. No.	Description
(1)	Map resource collection URI (Servlet path) for Controller. Typically, specify a servlet path indicating a collection of resources in the value attribute of `@RequestMapping` annotation. In the above example, a servlet path called `/api/v1/members` is mapped.
(2)	Assign `@RestController` annotation for Controller. Assigning `@RestController` annotation has same meaning of: • Assigning `org.springframework.stereotype.Controller` annotation in a class • Assigning `@org.springframework.web.bind.annotation.ResponseBody` annotation in Controller method which is described later. By assigning `@ResponseBody` to Controller method, the returned Resource object is marshalled in JSON or XML and set in response body.

Tip: `@RestController` is an annotation added from Spring Framework 4.0.

Due to `@RestController` annotation, it is not necessary to assign `@ResponseBody` annotation to each method of Controller. Hence, it is possible to create Controller for REST API in a simple way. For details about `@RestController` annotation refer to: Here.

An example to create a Controller for REST API by combining `@Controller` annotation and `@ResponseBody` annotation in a conventional way is given below.

```
@RequestMapping("members")
@Controller
public class MemberRestController {

    @RequestMapping(method = RequestMethod.GET)
    @ResponseStatus(HttpStatus.OK)
    @ResponseBody
    public Page<MemberResource> getMembers() {
        // ...
    }

    // ...

}
```

Implementing REST API that fetches collection of resources

Example to implement the REST API wherein a page search is performed for member resource collection specified by URI.

- Creating the JavaBean for receiving search conditions

  When search conditions are necessary to fetch resource collection, create a JavaBean for receiving the search conditions.

```
// (1)
public class MembersSearchQuery implements Serializable {
    private static final long serialVersionUID = 1L;

    // (2)
    @NotEmpty
    private String name;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
```

```
}
```

Sr. No.	Description
(1)	Create a JavaBean for receiving search conditions. When search conditions are not necessary, JavaBean need not be created.
(2)	Match the property name with the parameter name of request parameter. In the above example, value `"John"` is set in the name property of JavaBean for request `/api/v1/members?name=John`.

- REST API implementation

  Implement a process wherein page search is performed for a collection of Member resources.

```
@RequestMapping("members")
@RestController
public class MemberRestController {

    // omitted

    @Inject
    MemberService memberService;

    @Inject
    Mapper beanMapper;

    // (3)
    @RequestMapping(method = RequestMethod.GET)
    // (4)
    @ResponseStatus(HttpStatus.OK)
    public Page<MemberResource> getMembers(
            // (5)
            @Validated MembersSearchQuery query,
            // (6)
            Pageable pageable) {

        // (7)
        Page<Member> page = memberService.searchMembers(query.getName(),␣
↪pageable);
```

```
        // (8)
        List<MemberResource> memberResources = new ArrayList<>();
        for (Member member : page.getContent()) {
            memberResources.add(beanMapper.map(member, MemberResource.class));
        }
        Page<MemberResource> responseResource = new PageImpl<>
↪(memberResources,
                pageable, page.getTotalElements());

        // (9)
        return responseResource;
    }

    // omitted

}
```

Sr. No.	Description
(3)	Specify `RequestMethod.GET` in method attribute of `@RequestMapping` annotation.

(4)	Assign `@org.springframework.web.bind.annotation.ResponseStatus` as method annotation and specify the status code returned as response.
	Set 200 (OK) in the value attribute of `@ResponseStatus` annotation.
	Tip: How to specify the status code
	A fixed status code sent as response is specified in this example using `@ResponseStatus` annotation. However, it can also be specified in Controller logic.
	`public ResponseEntity<Page<MemberResource>> getMembers(`
	`@Validated MembersSearchQuery query,`
	`Pageable pageable) {`
	`// omitted`
	`return ResponseEntity.ok().body(responseResource);`
	`}`
	When it is necessary to change the responding status codes based on process details or process results, `org.springframework.http.ResponseEntity` is used, as shown in the above implementation.

次のページに続く

Table. 5.6 – 前のページからの続き

Sr. No.	Description
(5)	Specify a JavaBean for receiving search conditions as an argument. When input validation is necessary, assign `@Validated` as argument annotation. For input validation details, refer to "*Input Validation*".
(6)	When page search is necessary, specify `org.springframework.data.domain.Pageable` as an argument. For page search details, refer to "*Pagination*".
(7)	Call Service method of domain layer and fetch resource information (Entity etc.) matching with the condition. For domain layer implementation, refer to "*Domain Layer Implementation*".
(8)	Generate resource object that retains information published on the Web based on the resource information matching with the conditions (Entity etc.). By using `org.springframework.data.domain.PageImpl` class while sending page search result as response, the fields that are necessary as response at the time of page search, can be sent to the client. In the above example, a Resource object is being generated from Entity by using Bean mapping library. For details on Bean mapping library, refer to "*Bean Mapping (Dozer)*" . **When the quantity of code for generating Resource objects is more, it is recommended to create a method for generating Resource object in Helper class.**
(9)	Return a Resource object generated in (8). The object returned here is marshalled in JSON or XML and set in response body.

Response at the time of using `PageImpl` class is as below.

Highlighted portion shows the fields specific for page search.


```
{
  "content" : [ {
    "memberId" : "M000000001",
    "firstName" : "John",
    "lastName" : "Smith",
    "genderCode" : "1",
    "dateOfBirth" : "1977-03-07",
    "emailAddress" : "john.smith@test.com",
    "telephoneNumber" : "09012345678",
    "zipCode" : "1710051",
    "address" : "Tokyo",
    "credential" : {
      "signId" : "john.smit@test.com",
      "passwordLastChangedAt" : "2014-03-13T10:18:08.003Z",
      "lastModifiedAt" : "2014-03-13T10:18:08.003Z"
    },
    "createdAt" : "2014-03-13T10:18:08.003Z",
    "lastModifiedAt" : "2014-03-13T10:18:08.003Z"
  }, {
    "memberId" : "M000000002",
    "firstName" : "Sophia",
    "lastName" : "Smith",
    "genderCode" : "2",
    "dateOfBirth" : "1977-03-07",
    "emailAddress" : "sophia.smith@test.com",
    "telephoneNumber" : "09012345678",
    "zipCode" : "1710051",
    "address" : "Tokyo",
    "credential" : {
      "signId" : "sophia.smith@test.com",
      "passwordLastChangedAt" : "2014-03-13T10:18:08.003Z",
      "lastModifiedAt" : "2014-03-13T10:18:08.003Z"
    },
    "createdAt" : "2014-03-13T10:18:08.003Z",
    "lastModifiedAt" : "2014-03-13T10:18:08.003Z"
  } ],
  "last" : false,
  "totalPages" : 13,
  "totalElements" : 25,
  "size" : 2,
  "number" : 1,
  "sort" : [ {
    "direction" : "DESC",
    "property" : "lastModifiedAt",
    "ignoreCase" : false,
    "nullHandling": "NATIVE",
```

```
    "ascending" : false
  } ],
  "numberOfElements" : 2,
  "first" : false
}
```

Note: Points to be noted due to changes in API specifications of Spring Data Commons

In case of "terasoluna-gfw-common 5.0.0.RELEASE or later version" dependent spring-data-commons (1.9.1 RELEASE or later), there is a change in API specifications of interface for page search functionality (`org.springframework.data.domain.Page`) and class (`org.springframework.data.domain.PageImpl` and `org.springframework.data.domain.Sort.Order`).

Specifically,

- In `Page` interface and `PageImpl` class, `isFirst()` and `isLast()` methods are added in spring-data-commons 1.8.0.RELEASE, and `isFirstPage()` and `isLastPage()` methods are deleted from spring-data-commons 1.9.0.RELEASE.
- In `Sort.Order` class, `nullHandling` property is added in spring-data-commons 1.8.0.RELEASE.

When using `Page` interface (`PageImpl` class) as resource object of REST API, that application may also need to be modified, as JSON and XML format get changed.

- Adding Bean mapping definition

  In the above implementation, `Member` object and `MemberResource` object are copied by using "*Bean Mapping (Dozer)*" .

  It is not necessary to add Bean mapping definition when simply a copy of field value can be used. However, in the above implementation, the setting needs to be such that `credential.password` is not copied while copying `Member` object details to `MemberResource` object.

  It is necessary to add Bean mapping definition so that specific fields are not copied.

```
<!-- (11) -->
<?xml version="1.0" encoding="UTF-8"?>
<mappings xmlns="http://dozer.sourceforge.net" xmlns:xsi="http://www.w3.org/
↪2001/XMLSchema-instance"
    xsi:schemaLocation="http://dozer.sourceforge.net
          http://dozer.sourceforge.net/schema/beanmapping.xsd">
```

```
    <mapping type="one-way">
        <class-a>org.terasoluna.examples.rest.domain.model.MemberCredential</
↪class-a>
        <class-b>org.terasoluna.examples.rest.api.member.
↪MemberCredentialResource</class-b>
        <!-- (12) -->
        <field-exclude>
            <a>password</a>
            <b>password</b>
        </field-exclude>
    </mapping>

</mappings>
```

Sr. No.	Description
(11)	Create a file that defines mapping rules for `Member` object and `MemberResource` object. It is recommended to create a mapping definition file of Dozer for each resource. In this implementation, it is stored in `/xxx-web/src/main/resources/META-INF/dozer/memberResource-mapping.xml`.
(12)	In the above example, `password` field is not copied while copying the details of `MemberCredential` which is a related entity of `Member`, to `MemberCredentialResource`, a related resource of `MemberResource`. For Bean mapping definition methods, refer to "*Bean Mapping (Dozer)*" .

- Request example

```
GET /rest-api-web/api/v1/members?name=Smith&page=0&size=2 HTTP/1.1
Accept: text/plain, application/json, application/*+json, */*
User-Agent: Java/1.7.0_51
Host: localhost:8080
Connection: keep-alive
```

- Response Example

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
X-Track: fb63a6d446f849feb8ccaa4c9a794333
Content-Type: application/json;charset=UTF-8
Transfer-Encoding: chunked
Date: Thu, 13 Mar 2014 11:10:43 GMT

{"content":[{"memberId":"M000000001","firstName":"John","lastName":"Smith",
↪"genderCode":"1","dateOfBirth":"2013-03-13","emailAddress":
↪"user1394709042120@test.com","telephoneNumber":"09012345678","zipCode":
↪"1710051","address":"Tokyo","credential":{"signId":"user1394709042120@test.
↪com","passwordLastChangedAt":"2014-03-13T11:10:43.066Z","lastModifiedAt":
↪"2014-03-13T11:10:43.066Z"},"createdAt":"2014-03-13T11:10:43.066Z",
↪"lastModifiedAt":"2014-03-13T11:10:43.066Z"},{"memberId":"M000000002",
↪"firstName":"Sophia","lastName":"Smith","genderCode":"2","dateOfBirth":
↪"2013-03-13","emailAddress":"user1394709043663@test.com","telephoneNumber":
↪"09012345678","zipCode":"1710051","address":"Tokyo","credential":{"signId":
↪"user1394709043663@test.com","passwordLastChangedAt":"2014-03-13T11:10:43.
↪678Z","lastModifiedAt":"2014-03-13T11:10:43.678Z"},"createdAt":"2014-03-
↪13T11:10:43.678Z","lastModifiedAt":"2014-03-13T11:10:43.678Z"}],"last":true,
↪"totalPages":1,"totalElements":2,"size":2,"number":0,"sort":null,
↪"numberOfElements":2,"first":true}
```

Tip: When page search is not necessary, Resource class list may be handled directly.

Following is the definition of the Controller method used when handling the list of Resource class directly.

```
@RequestMapping(method = RequestMethod.GET)
@ResponseStatus(HttpStatus.OK)
public List<MemberResource> getMembers(
        @Validated MembersSearchQuery query) {
    // omitted
}
```

JSON is as follows when list of Resource class is directly handled.

```
[ {
    "memberId" : "M000000001",
    "firstName" : "John",
    "lastName" : "Smith",
    "genderCode" : "1",
    "dateOfBirth" : "1977-03-07",
```

```
    "emailAddress" : "john.smith@test.com",
    "telephoneNumber" : "09012345678",
    "zipCode" : "1710051",
    "address" : "Tokyo",
    "credential" : {
      "signId" : "john.smit@test.com",
      "passwordLastChangedAt" : "2014-03-13T10:18:08.003Z",
      "lastModifiedAt" : "2014-03-13T10:18:08.003Z"
    },
    "createdAt" : "2014-03-13T10:18:08.003Z",
    "lastModifiedAt" : "2014-03-13T10:18:08.003Z"
}, {
    "memberId" : "M000000002",
    "firstName" : "Sophia",
    "lastName" : "Smith",
    "genderCode" : "2",
    "dateOfBirth" : "1977-03-07",
    "emailAddress" : "sophia.smith@test.com",
    "telephoneNumber" : "09012345678",
    "zipCode" : "1710051",
    "address" : "Tokyo",
    "credential" : {
      "signId" : "sophia.smith@test.com",
      "passwordLastChangedAt" : "2014-03-13T10:18:08.003Z",
      "lastModifiedAt" : "2014-03-13T10:18:08.003Z"
    },
    "createdAt" : "2014-03-13T10:18:08.003Z",
    "lastModifiedAt" : "2014-03-13T10:18:08.003Z"
} ]
```

Implementing REST API that adds a resource to collection

Example of implementation of REST API wherein a specified Member resource is created and added to the collection is given below.

- REST API implementation

  Implement a process that creates specified Member resource and adds it to the collection.

```
@RequestMapping("members")
@RestController
public class MemberRestController {

```

```
    // omitted

    // (1)
    @RequestMapping(method = RequestMethod.POST)
    // (2)
    @ResponseStatus(HttpStatus.CREATED)
    public MemberResource postMember(
            // (3)
            @RequestBody @Validated({ PostMembers.class, Default.class })
            MemberResource requestedResource) {

        // (4)
        Member inputMember = beanMapper.map(requestedResource, Member.class);
        Member createdMember = memberService.createMember(inputMember);

        MemberResource responseResource = beanMapper.map(createdMember,
                MemberResource.class);

        return responseResource;
    }

    // omitted

}
```

Sr. No.	Description
(1)	Specify `RequestMethod.POST` in the method attribute of `@RequestMapping` annotation.
(2)	Assign `@ResponseStatus` annotation as method annotation and specify responding status code. Set **201(Created)** in the value attribute of `@ResponseStatus` annotation.
(3)	Specify JavaBean (Resource class) that receives information of newly created resource as an argument. Assign `@org.springframework.web.bind.annotation.RequestBody` as argument annotation. By assigning `@RequestBody` annotation, JSON or XML data set in request Body is unmarshalled in Resource object. Assign `@Validated` annotation as argument annotation to enable input validation. For details on input validation, refer to "*Input Validation*" .
(4)	Call Service method of domain layer and create a new resource. For domain layer implementation, refer to "*Domain Layer Implementation*".

- Request example

```
POST /rest-api-web/api/v1/members HTTP/1.1
Accept: text/plain, application/json, application/*+json, */*
Content-Type: application/json;charset=UTF-8
User-Agent: Java/1.7.0_51
Host: localhost:8080
Connection: keep-alive
Content-Length: 248

{"firstName":"John","lastName":"Smith","genderCode":"1","dateOfBirth":"2013-
↪03-13","emailAddress":"user1394708306056@test.com","telephoneNumber":
↪"09012345678","zipCode":"1710051","address":"Tokyo","credential":{"signId":
↪null,"password":"zaq12wsx"}}
```

- Response example

```
HTTP/1.1 201 Created
Server: Apache-Coyote/1.1
X-Track: c7e9c8a9aa4f40ff87f3acdb77baccdf
Content-Type: application/json;charset=UTF-8
Transfer-Encoding: chunked
Date: Thu, 13 Mar 2014 10:58:26 GMT

{"memberId":"M000000023","firstName":"John","lastName":"Smith","genderCode":"1
↪","dateOfBirth":"2013-03-13","emailAddress":"user1394708306056@test.com",
↪"telephoneNumber":"09012345678","zipCode":"1710051","address":"Tokyo",
↪"credential":{"signId":"user1394708306056@test.com","passwordLastChangedAt":
↪"2014-03-13T10:58:26.324Z","lastModifiedAt":"2014-03-13T10:58:26.324Z"},
↪"createdAt":"2014-03-13T10:58:26.324Z","lastModifiedAt":"2014-03-13T10:58:
↪26.324Z"}
```

Implementing REST API that fetches specified resource

Implementation of REST API that fetches the Member resource specified by URI, is shown below.

- REST API implementation

  Implement a process that fetches the Member resource specified by URI.

```
@RequestMapping("members")
@RestController
public class MemberRestController {

    // omitted

    // (1)
    @RequestMapping(value = "{memberId}", method = RequestMethod.GET)
    // (2)
    @ResponseStatus(HttpStatus.OK)
    public MemberResource getMember(
            // (3)
            @PathVariable("memberId") String memberId) {
```

```
        // (4)
        Member member = memberService.getMember(memberId);

        MemberResource responseResource = beanMapper.map(member,
                MemberResource.class);

        return responseResource;
    }

    // omitted

}
```

Sr. No.	Description
(1)	Specify path variable (`{memberId}` in the example above) in value attribute whereas `RequestMethod.GET` in method attribute of `@RequestMapping` annotation. A value that uniquely identifies the resource is specified in `{memberId}`.
(2)	Assign `@ResponseStatus` annotation as method annotation and specify the responding status code. Set 200 (OK) in value attribute of `@ResponseStatus` annotation.
(3)	Fetch the value that uniquely identifies the resource from path variable. Value specified in path variable (`{memberId}`) can be received as method argument by specifying `@PathVariable("memberId")` as argument annotation. For details on path variable, refer to "*Retrieving values from URL path*". In the above example, when URI is `/api/v1/members/M12345`, `"M12345"` is stored in `memberId` of argument.
(4)	Call Service method of domain layer and acquire the resource information (Entity etc.) that matches with the ID fetched from path variable. For domain layer implementation, refer to "*Domain Layer Implementation*".

- Request example

```
GET /rest-api-web/api/v1/members/M000000003 HTTP/1.1
Accept: text/plain, application/json, application/*+json, */*
User-Agent: Java/1.7.0_51
Host: localhost:8080
Connection: keep-alive
```

- Response Example

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
X-Track: 275b4e7a61f946eea47672f272315ac2
Content-Type: application/json;charset=UTF-8
Transfer-Encoding: chunked
Date: Thu, 13 Mar 2014 11:25:13 GMT

{"memberId":"M000000003","firstName":"John","lastName":"Smith","genderCode":"1
↪","dateOfBirth":"2013-03-13","emailAddress":"user1394709913496@test.com",
↪"telephoneNumber":"09012345678","zipCode":"1710051","address":"Tokyo",
↪"credential":{"signId":"user1394709913496@test.com","passwordLastChangedAt":
↪"2014-03-13T11:25:13.762Z","lastModifiedAt":"2014-03-13T11:25:13.762Z"},
↪"createdAt":"2014-03-13T11:25:13.762Z","lastModifiedAt":"2014-03-13T11:25:
↪13.762Z"}
```

Implementing REST API that updates specified resource

Implementation of REST API that updates the Member resource specified in URI, is shown below.

- REST API implementation

  Implement a process that updates the Member resource specified in URI.

```
@RequestMapping("members")
@RestController
public class MemberRestController {

    // omitted

    // (1)
```

```
    @RequestMapping(value = "{memberId}", method = RequestMethod.PUT)
    // (2)
    @ResponseStatus(HttpStatus.OK)
    public MemberResource putMember(
            @PathVariable("memberId") String memberId,
            // (3)
            @RequestBody @Validated({ PutMember.class, Default.class })
            MemberResource requestedResource) {

        // (4)
        Member inputMember = beanMapper.map(
            requestedResource, Member.class);
        Member updatedMember = memberService.updateMember(
            memberId, inputMember);

        MemberResource responseResource = beanMapper.map(updatedMember,
                MemberResource.class);

        return responseResource;
    }

    // omitted

}
```

Sr. No.	Description
(1)	Specify path variable (`{memberId}` in the example above) in value attribute whereas `RequestMethod.PUT` in "method" attribute of `@RequestMapping` annotation. Value that uniquely identifies the resource is specified in `{memberId}`.
(2)	Assign `@ResponseStatus`annotation as method annotation and specify the responding status code. Set 200 (OK) in value attribute of `@ResponseStatus` annotation.
(3)	Specify JavaBean (Resource class) for receiving the details of resource update as an argument. By assigning `@RequestBody` annotation as argument annotation, JSON or XML data set in request Body is unmarshalled in Resource object. Assign `@Validated` annotation as argument annotation to enable input validation. For details on input validation, refer to "*Input Validation*" .
(4)	Call Service method of domain layer and update the resource information (Entity etc.) matching with the ID fetched from path variable. For domain layer implementation, refer to "*Domain Layer Implementation*".

- Request example

```
PUT /rest-api-web/api/v1/members/M000000004 HTTP/1.1
Accept: text/plain, application/json, application/*+json, */*
Content-Type: application/json;charset=UTF-8
User-Agent: Java/1.7.0_51
Host: localhost:8080
Connection: keep-alive
Content-Length: 221

{"memberId":"M000000004","firstName":"John","lastName":"Smith","genderCode":"1
↪","dateOfBirth":"2013-03-08","emailAddress":"user1394710559584@test.com",
↪"telephoneNumber":"09012345678","zipCode":"1710051","address":"Tokyo"}
```

- Response example

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
X-Track: 5e8fea3aae044e94bf20a293e155af57
Content-Type: application/json;charset=UTF-8
Transfer-Encoding: chunked
Date: Thu, 13 Mar 2014 11:35:59 GMT

{"memberId":"M000000004","firstName":"John","lastName":"Smith","genderCode":"1
↪","dateOfBirth":"2013-03-08","emailAddress":"user1394710559584@test.com",
↪"telephoneNumber":"09012345678","zipCode":"1710051","address":"Tokyo",
↪"credential":{"signId":"user1394710559584@test.com","passwordLastChangedAt":
↪"2014-03-13T11:35:59.847Z","lastModifiedAt":"2014-03-13T11:35:59.847Z"},
↪"createdAt":"2014-03-13T11:35:59.847Z","lastModifiedAt":"2014-03-13T11:36:
↪00.122Z"}
```

Implementing REST API that deletes specified resource

Implementation of REST API that deletes the Member resource specified by URI is as follows:

- REST API implementation

  Implement a process that deletes the Member resource specified by URI.

```
@RequestMapping("members")
@RestController
public class MemberRestController {

    // omitted

    // (1)
    @RequestMapping(value = "{memberId}", method = RequestMethod.DELETE)
    // (2)
    @ResponseStatus(HttpStatus.NO_CONTENT)
    public void deleteMember(
            @PathVariable("memberId") String memberId) {
```

```
        // (3)
        memberService.deleteMember(memberId);

    }

    // omitted

}
```

Sr. No.	Description
(1)	Specify path variable (`{memberId}` in the example above) in value attribute and `RequestMethod.DELETE` in method attribute of `@RequestMapping` annotation.
(2)	Assign `@ResponseStatus` annotation as method annotation and specify the responding status code. Set **204 (NO_CONTENT)** in value attribute of `@ResponseStatus` annotation.
(3)	Call Service method of domain layer and delete resource information (Entity etc.) matching with the ID fetched from path variable. For domain layer implementation, refer to "*Domain Layer Implementation*".

Note: To set deleted resource information in response BODY, set (200) OK in the status code.

- Request example

```
DELETE /rest-api-web/api/v1/members/M000000005 HTTP/1.1
Accept: text/plain, application/json, application/*+json, */*
User-Agent: Java/1.7.0_51
Host: localhost:8080
Connection: keep-alive
```

- Response example

```
HTTP/1.1 204 No Content
Server: Apache-Coyote/1.1
X-Track: e06c5bd40c864a299c48d9be3f12b2c0
Date: Thu, 13 Mar 2014 11:40:05 GMT
```

Implementing exception handling

How to handle the exceptions occurring in RESTful Web Service is explained below.

RESTful Web Service oriented generic exception handling feature is not provided in Spring MVC.

Alternately,
(`org.springframework.web.servlet.mvc.method.annotation.ResponseEntityExceptionHandler`)
is provided as the class that assists in implementing exception handling for RESTful Web Service.

This guideline recommends a common exception handling method, by creating an exception handling class that inherits the Spring MVC class and assigning `@ControllerAdvice` annotation to this exception handling class.

Method that handles exceptions in Spring MVC framework is implemented in advance in `ResponseEntityExceptionHandler` by using `@ExceptionHandler` annotation.

Therefore, individual handling of the exceptions in Spring MVC framework need not be implemented.

Further, HTTP status codes corresponding to the exceptions handled by `ResponseEntityExceptionHandler` are set by the same specifications as `DefaultHandlerExceptionResolver`.

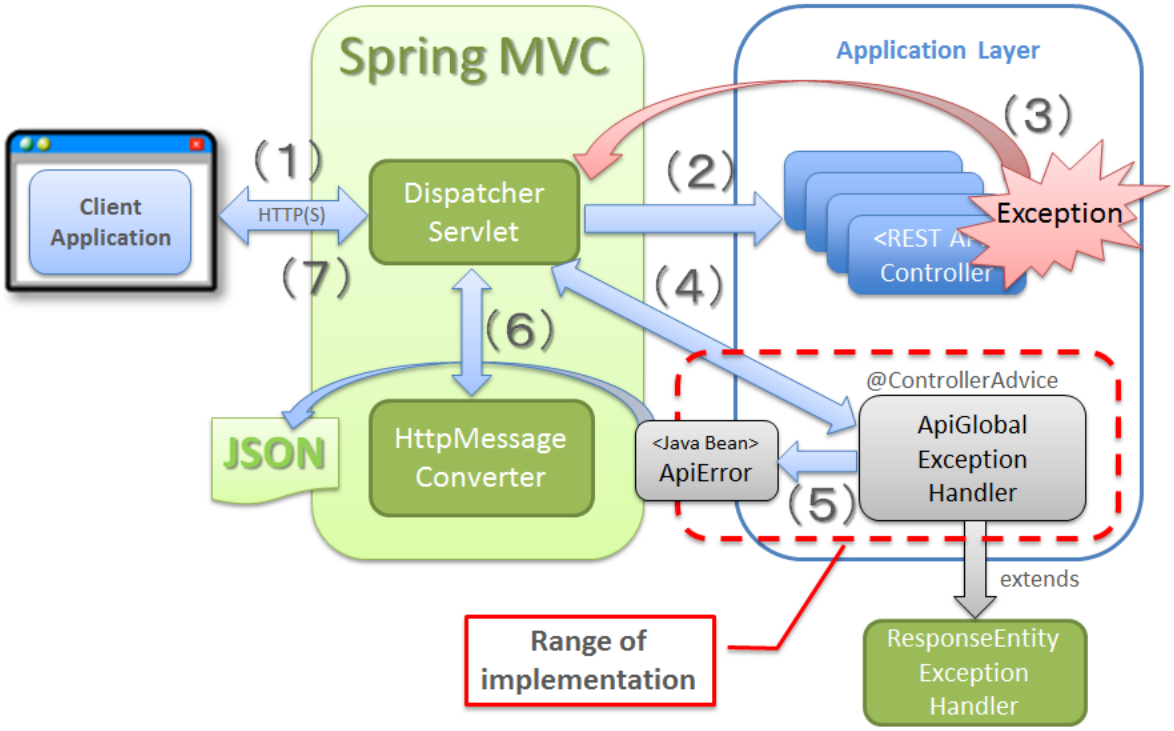
For handled exceptions and HTTP status codes thus set, refer to "*HTTP response code set by DefaultHandlerExceptionResolver*".

Although an empty response BODY is returned by default implementation of `ResponseEntityExceptionHandler`, it can be extended so as to output error information in the response Body.

This guideline recommends to output appropriate error information in the response Body.

Process flow wherein, exception handling class that inherits `ResponseEntityExceptionHandler` is

created and common exception handling is performed, is described before explaining the typical implementation. Please note that, separate implementation is required for the part marked in red frame.



Sr. No.	Processing layer	Description
(1) (2)	Spring MVC (Framework)	Spring MVC receives a request from client and calls REST API.
(3)		An exception occurs during REST API process. The exception occurred is caught by Spring MVC.
(4)		Spring MVC delegates the process to exception handling class.
(5)	Custom Exception Handler (Common Component)	An error object that retains error information is generated in the exception handling class and returned to Spring MVC.
(6)	Spring MVC (Framework)	Spring MVC converts the error object to JSON format message using `HttpMessageConverter`.
(7)		Spring MVC sets the JSON format error message in response BODY and sends response to the client.

Implementation to output error information in response Body

- Error information should be in the following JSON format.

```
{
  "code" : "e.ex.fw.7001",
  "message" : "Validation error occurred on item in the request body.",
  "details" : [ {
```

```
    "code" : "ExistInCodeList",
    "message" : "\"genderCode\" must exist in code list of CL_GENDER.",
    "target" : "genderCode"
  } ]
}
```

- Create JavaBean that retains error information.

```
package org.terasoluna.examples.rest.api.common.error;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;

import com.fasterxml.jackson.annotation.JsonInclude;

// (1)
public class ApiError implements Serializable {

    private static final long serialVersionUID = 1L;

    private final String code;

    private final String message;

    @JsonInclude(JsonInclude.Include.NON_EMPTY)
    private final String target; // (2)

    @JsonInclude(JsonInclude.Include.NON_EMPTY)
    private final List<ApiError> details = new ArrayList<>(); // (3)

    public ApiError(String code, String message) {
        this(code, message, null);
    }

    public ApiError(String code, String message, String target) {
        this.code = code;
        this.message = message;
        this.target = target;
    }

    public String getCode() {
        return code;
    }
```

```
    public String getMessage() {
        return message;
    }

    public String getTarget() {
        return target;
    }

    public List<ApiError> getDetails() {
        return details;
    }

    public void addDetail(ApiError detail) {
        details.add(detail);
    }

}
```

Sr. No.	Description
(1)	Create a class for retaining error information. In the above example, it is the class that retains lists of error codes, error messages, error targets and detailed error information.
(2)	Field that retains the value for identifying the target where error has occurred. When an error occurs in input validation, there are cases where the value that identifies the field where error has occurred needs to be returned to the client. In such cases, it is necessary to set a field that retains the field name where error has occurred.
(3)	Field that retains the list of detailed error information. When an error occurs in input validation, it is necessary to return all the error information to the client as there are cases with multiple error causes. In such cases, a field that lists and retains detailed error information is necessary.

Tip: When the value is `null` or empty, it is possible to avoid fields being output to JSON by specifying `@JsonInclude(JsonInclude.Include.NON_EMPTY)` in the field. When the condition to disable field output is to be restricted to `null`, it is advisable to specify `@JsonInclude(JsonInclude.Include.NON_NULL)`.

- Create a class for generating JavaBean that retains error information.

  Refer to *Appendix* for the source code when implementation of all exception handling is completed.

```
// (4)
@Component
public class ApiErrorCreator {

    @Inject
    MessageSource messageSource;

    public ApiError createApiError(WebRequest request, String errorCode,
            String defaultErrorMessage, Object... arguments) {
        // (5)
        String localizedMessage = messageSource.getMessage(errorCode,
                arguments, defaultErrorMessage, request.getLocale());
        return new ApiError(errorCode, localizedMessage);
    }

    // omitted

}
```

Sr. No.	Description
(4)	If needed, create a class that provides the method for generating error information. Creating this class is not mandatory. However, it is recommended to create it so as to clearly define the role division.
(5)	Fetch the error message from `MessageSource`. For message management methods, refer to "*Message Management*".

Tip: In the above example, `org.springframework.web.context.request.WebRequest` is received as an argument to support localization of messages. `WebRequest` is not necessary when message localization is not required.

The reason for using `WebRequest` as an argument instead of `java.util.Locale` is due to an additional requirement wherein, HTTP request details are to be embedded in the error message. When there is no such requirement to embed HTTP request details in error, `Locale` can also be used.

- `ResponseEntityExceptionHandler` method is extended and the implementation to output error information in response Body is carried out.

  Refer to *Appendix* for the source code when implementation for all exception handling is completed.

```
@ControllerAdvice // (6)
public class ApiGlobalExceptionHandler extends ResponseEntityExceptionHandler
 ↪{

    @Inject
    ApiErrorCreator apiErrorCreator;

    @Inject
    ExceptionCodeResolver exceptionCodeResolver;

    // (7)
    @Override
    protected ResponseEntity<Object> handleExceptionInternal(Exception ex,
            Object body, HttpHeaders headers, HttpStatus status,
            WebRequest request) {
        final Object apiError;
        // (8)
        if (body == null) {
            String errorCode = exceptionCodeResolver.resolveExceptionCode(ex);
            apiError = apiErrorCreator.createApiError(request, errorCode, ex
                    .getLocalizedMessage());
        } else {
            apiError = body;
        }
        // (9)
        return ResponseEntity.status(status).headers(headers).body(apiError);
    }

    // omitted

}
```

Sr. No.	Description
(6)	Create a class that inherits `ResponseEntityExceptionHandler` provided by Spring MVC and assign `@ControllerAdvice` annotation.
(7)	Override handleExceptionInternal method of `ResponseEntityExceptionHandler`.
(8)	When the JavaBean output to response Body is not specified, generate a JavaBean object that retains error information. In the above example, the exception class changes the error code by using `ExceptionCodeResolver` provided by common library. For setting example of `ExceptionCodeResolver`, refer to "*Resolving error codes and messages using ExceptionCodeResolver*" . When the JavaBean output to response Body is specified, use the specified JavaBean as it is. This process is implemented considering that error information is generated individually in the error handling process for each exception.
(9)	Set the error information generated in (8) in the 'Body' of HTTP Entity for response and then return the same. Error information thus returned is converted to JSON using framework and sent as a response. Appropriate values are set in the status code by `ResponseEntityExceptionHandler` provided by Spring MVC. Refer to "*HTTP response code set by DefaultHandlerExceptionResolver*" for status codes that are set.

Tip: Attribute of @ControllerAdvice annotation added in Spring Framework 4.0

By specifying an attribute of `@ControllerAdvice` annotation, it has been improved to allow flexibility in specifying Controller to apply a method implemented in the class wherein `@ControllerAdvice` is assigned. For details about attribute refer to: *Attribute of @ControllerAdvice*.

Note: Points to be noted while using an attribute of @ControllerAdvice annotation

By using an attribute of `@ControllerAdvice` annotation, it is possible to share exception handling in respective granularity, however, it is advisable not to specify attribute of `@ControllerAdvice` annotation for exception handling of common application (class corresponding to `ApiGlobalExceptionHandler` class in the above example).

When an attribute is specified in `@ControllerAdvice` annotation assigned in `ApiGlobalExceptionHandler` , a part of exception handling may not be possible that occurs in framework process provided by Spring MVC.

Specifically, in `ApiGlobalExceptionHandler` class, exception handling is not possible for the exceptions that occur when REST API (process method of Controller) corresponding to the request could not be found, hence, it is not possible to correctly respond to errors such as "405 Method Not Allowed", etc.

- Response example

```
HTTP/1.1 400 Bad Request
Server: Apache-Coyote/1.1
X-Track: e60b3b6468194e22852c8bfc7618e625
Content-Type: application/json;charset=UTF-8
Transfer-Encoding: chunked
Date: Thu, 13 Mar 2014 12:16:55 GMT
Connection: close

{"code":"e.ex.fw.7001","message":"Validation error occurred on item in the␣
↪request body.","details":[{"code":"ExistInCodeList","message":"\"genderCode\
↪" must exist in code list of CL_GENDER.","target":"genderCode"}]}
```

Implementing input error exception handling

Implementation for responding to input errors (syntax error, unit item check error, correlated field check error) is explained here.

Following three exceptions need to be handled in order to respond to input errors.

Sr. No.	Exception	Description
(1)	org.springframework.web.bind. MethodArgumentNotValidException	This exception occurs when there is an error during the input validation for JSON or XML specified in request BODY. Basically, it occurs when an invalid value is entered in the resource that is specified when POST or PUT method is performed for the resource.
(2)	org.springframework.validation. BindException	This exception occurs when there is an error during the input validation for request parameter (key=query string of value format). Basically, it occurs when an invalid value is entered in the search conditions specified at the time of GET method of resource collection.
(3)	org.springframework.http.converter. HttpMessageNotReadableException	This exception occurs when there is an error while generating Resource object from JSON or XML. Basically, it occurs in cases such as invalid JSON or XML syntax or violation of schema definition.

Note: `org.springframework.beans.TypeMismatchException` occurs when there is a type conversion error for value while fetching values from request parameter, request header and path variable, by using annotation provided by Spring Framework.

When following annotations are specified as arguments of Controller handler method (argument other than `String`), `TypeMismatchException` may occur.

- `@org.springframework.web.bind.annotation.RequestParam`
- `@org.springframework.web.bind.annotation.RequestHeader`
- `@org.springframework.web.bind.annotation.Pathvariable`
- `@org.springframework.web.bind.annotation.MatrixVariable`

`TypeMismatchException` is handled by `ResponseEntityExceptionHandler` resulting in 400 (Bad Request). As a result, individual handling is not required.

Refer to "*Resolving error codes and messages using ExceptionCodeResolver*" in order to resolve the error

codes and error messages to be set in error information.

- Method is created to generate error information for input validation errors.

```
@Component
public class ApiErrorCreator {

    @Inject
    MessageSource messageSource;

    // omitted

    // (1)
    public ApiError createBindingResultApiError(WebRequest request,
            String errorCode, BindingResult bindingResult,
            String defaultErrorMessage) {
        ApiError apiError = createApiError(request, errorCode,
                defaultErrorMessage);
        for (FieldError fieldError : bindingResult.getFieldErrors()) {
            apiError.addDetail(createApiError(request, fieldError, fieldError
                    .getField()));
        }
        for (ObjectError objectError : bindingResult.getGlobalErrors()) {
            apiError.addDetail(createApiError(request, objectError,␣
↪objectError
                    .getObjectName()));
        }
        return apiError;
    }

    // (2)
    private ApiError createApiError(WebRequest request,
            DefaultMessageSourceResolvable messageResolvable, String target) {
        String localizedMessage = messageSource.getMessage(messageResolvable,
                request.getLocale());
        return new ApiError(messageResolvable.getCode(), localizedMessage,␣
↪target);
    }

    // omitted

}
```

Sr. No.	Description
(1)	A method is created to generate error information for input validation. In the above example, single field check error (`FieldError`) and correlated field check error (`ObjectError`) are added to detailed error information. This method need not be provided when it is not necessary to output error information for each item.
(2)	A common method is created since same process is implemented for single field check error (`FieldError`) and correlation check error (`ObjectError`).

- `ResponseEntityExceptionHandler` method is extended and the implementation to output input validation error information in response Body, is performed.

```
@ControllerAdvice
public class ApiGlobalExceptionHandler extends ResponseEntityExceptionHandler
 ↪{

    @Inject
    ApiErrorCreator apiErrorCreator;

    @Inject
    ExceptionCodeResolver exceptionCodeResolver;

    // omitted

    // (3)
    @Override
    protected ResponseEntity<Object> handleMethodArgumentNotValid(
            MethodArgumentNotValidException ex, HttpHeaders headers,
            HttpStatus status, WebRequest request) {
        return handleBindingResult(ex, ex.getBindingResult(), headers, status,
                request);
    }

    // (4)
    @Override
    protected ResponseEntity<Object> handleBindException(BindException ex,
            HttpHeaders headers, HttpStatus status, WebRequest request) {
        return handleBindingResult(ex, ex.getBindingResult(), headers, status,
                request);
```

```
    }

    // (5)
    @Override
    protected ResponseEntity<Object> handleHttpMessageNotReadable(
            HttpMessageNotReadableException ex, HttpHeaders headers,
            HttpStatus status, WebRequest request) {
        if (ex.getCause() instanceof Exception) {
            return handleExceptionInternal((Exception) ex.getCause(), null,
                    headers, status, request);
        } else {
            return handleExceptionInternal(ex, null, headers, status,␣
↪request);
        }
    }

    // omitted

    // (6)
    protected ResponseEntity<Object> handleBindingResult(Exception ex,
            BindingResult bindingResult, HttpHeaders headers,
            HttpStatus status, WebRequest request) {
        String code = exceptionCodeResolver.resolveExceptionCode(ex);
        String errorCode = exceptionCodeResolver.resolveExceptionCode(ex);
        ApiError apiError = apiErrorCreator.createBindingResultApiError(
                request, errorCode, bindingResult, ex.getMessage());
        return handleExceptionInternal(ex, apiError, headers, status,␣
↪request);
    }

    // omitted

}
```

Sr. No.	Description
(3)	Override handleMethodArgumentNotValid method of `ResponseEntityExceptionHandler` and extend error handling for `MethodArgumentNotValidException`. In the above example, the process is delegated to a common method (6) that handles input validation errors. When it is not necessary to output error information for each item, overriding is not required. **400 (Bad Request)** is set in the status code and presence of some flaw in the field value of the specified resource is notified.
(4)	Override handleBindException method of `ResponseEntityExceptionHandler` and extend error handling for `BindException`. In the above example, the process is delegated to a common method (6) that handles input validation error. When it is not necessary to output error information for each item, overriding is not required. **400 (Bad Request)** is set in the status code and presence of flaw in the specified request parameter is notified.
(5)	Override handleHttpMessageNotReadable method of `ResponseEntityExceptionHandler` and extend error handling for `HttpMessageNotReadableException`. In the above example, detailed error handling is performed by using cause exception. If a detailed error handling is not necessary, overriding is not required. **400 (Bad Request)** is set in the status code and presence of flaw in the specified resource format etc. is notified
(6)	Generate a JavaBean object that retains error information for input validation error. In the above example, this method is created as a common method since same process is implemented in handleMethodArgumentNotValid and handleBindException.

Tip: Error handling when using JSON

When JSON is used as the resource format, following exception is stored as cause exception of `HttpMessageNotReadableException`.

Sr. No.	Exception class	Description
(1)	com.fasterxml.jackson.core. JsonParseException	It occurs when an invalid syntax is included for JSON.
(2)	com.fasterxml.jackson.databind.exc. UnrecognizedPropertyException	It occurs when a field which does not exist in Resource object is specified in JSON.
(3)	com.fasterxml.jackson.databind. JsonMappingException	It occurs when a value type conversion error occurs while converting from JSON to Resource object.

- Following error response is sent when an input validation error (single field check error, correlated field check error) occurs.

```
HTTP/1.1 400 Bad Request
Server: Apache-Coyote/1.1
X-Track: 13522b3badf2432ba4cad0dc7aeaee80
Content-Type: application/json;charset=UTF-8
Transfer-Encoding: chunked
Date: Wed, 19 Feb 2014 05:08:28 GMT
Connection: close

{"code":"e.ex.fw.7002","message":"Validation error occurred on item in the␣
↪request parameters.","details":[{"code":"NotEmpty","message":"\"{0}\" may␣
↪not be empty.","target":"name"}]}
```

- Following error response is sent when JSON errors (format error etc.) occur.

```
HTTP/1.1 400 Bad Request
Server: Apache-Coyote/1.1
X-Track: ca4c742a6bfd49e5bc01cd0b124738a1
Content-Type: application/json;charset=UTF-8
Transfer-Encoding: chunked
Date: Wed, 19 Feb 2014 13:32:24 GMT
Connection: close

{"code":"e.ex.fw.7003","message":"Request body format error occurred."}
```

Implementing exception handling for "Resource not found" error

When a resource does not exist, implementation for responding to the "resource not found" error, is explained below.

When a resource matching with the ID fetched from path variable is not found, generate an exception notifying "resource not found" .

`org.terasoluna.gfw.common.exception.ResourceNotFoundException` is provided by common library as an exception notifying "resource not found".

Implementation is as given below.

- When a resource matching with the ID fetched from path variable is not found, generate `ResourceNotFoundException`.

```
public Member getMember(String memberId) {
    Member member = memberRepository.findOne(memberId);
    if (member == null) {
        throw new ResourceNotFoundException(ResultMessages.error().add(
                "e.ex.mm.5001", memberId));
    }
    return member;
}
```

- Create a method to generate error information for `ResultMessages`.

```
@Component
public class ApiErrorCreator {

    // omitted

    // (1)
    public ApiError createResultMessagesApiError(WebRequest request,
            String rootErrorCode, ResultMessages resultMessages,
            String defaultErrorMessage) {
        ApiError apiError;
        if (resultMessages.getList().size() == 1) {
            ResultMessage resultMessage = resultMessages.iterator().next();
            String errorCode = resultMessage.getCode();
            String errorText = resultMessage.getText();
            if (errorCode == null && errorText == null) {
                errorCode = rootErrorCode;
            }
            apiError = createApiError(request, errorCode, errorText,
                    resultMessage.getArgs());
        } else {
            apiError = createApiError(request, rootErrorCode,
                    defaultErrorMessage);
            for (ResultMessage resultMessage : resultMessages.getList()) {
                apiError.addDetail(createApiError(request, resultMessage
                        .getCode(), resultMessage.getText(), resultMessage
                        .getArgs()));
            }
        }
        return apiError;
    }

    // omitted

}
```

Sr. No.	Description
(1)	Create a method for generating error information from process results. In the above example, the message information retained by `ResultMessages` is set in error information.

Note: In the above example, as `ResultMessages` can retain multiple messages, the process is divided as per when a single message is stored and when multiple messages are stored.

When it is not necessary to support multiple messages, the process wherein the message at the start is generated as error information, may be implemented.

- Create a method for handling the exception that notifies "resource not found" error, in the class that performs error handling.

```
@ControllerAdvice
public class ApiGlobalExceptionHandler extends ResponseEntityExceptionHandler
↪{

    @Inject
    ApiErrorCreator apiErrorCreator;

    @Inject
    ExceptionCodeResolver exceptionCodeResolver;

    // omitted

    // (2)
    @ExceptionHandler(ResourceNotFoundException.class)
    public ResponseEntity<Object> handleResourceNotFoundException(
            ResourceNotFoundException ex, WebRequest request) {
        return handleResultMessagesNotificationException(ex, new␣
↪HttpHeaders(),
                HttpStatus.NOT_FOUND, request);
    }

    // omitted

    // (3)
    private ResponseEntity<Object> handleResultMessagesNotificationException(
            ResultMessagesNotificationException ex, HttpHeaders headers,
            HttpStatus status, WebRequest request) {
        String errorCode = exceptionCodeResolver.resolveExceptionCode(ex);
        ApiError apiError = apiErrorCreator.createResultMessagesApiError(
                request, errorCode, ex.getResultMessages(), ex.getMessage());
        return handleExceptionInternal(ex, apiError, headers, status,␣
↪request);
    }
```

```
    // omitted

}
```

Sr. No.	Description
(2)	Add a method for handling `ResourceNotFoundException`. `ResourceNotFoundException` exception can be handled if `@ExceptionHandler(ResourceNotFoundException.class)` is specified as method annotation. In the above example, the process is delegated to method that handles exception of the parent class (`ResultMessagesNotificationException`) of `ResourceNotFoundException`. Set **404 (Not Found)** in the status code and notify a message stating, 'specified resource does not exist in the server'.
(3)	Generate a JavaBean object that retains error information for "resource not found" error and business error. In the above example, this method is created as a common method since the process is same as that for error handling of business error discussed hereafter.

- When resource is not found, following error response is generated.

```
HTTP/1.1 404 Not Found
Server: Apache-Coyote/1.1
X-Track: 5ee563877f3140fd904d0acf52eba398
Content-Type: application/json;charset=UTF-8
Transfer-Encoding: chunked
Date: Wed, 19 Feb 2014 08:46:18 GMT

{"code":"e.ex.mm.5001","message":"Specified member not found. member id :␣
↪M000000001"}
```

Implementing exception handling for business errors

An implementation wherein business error is sent as a response on detecting violation of business rule, is explained here.

Perform business rule check as Service process and generate business exception when a business rule violation is detected. For details on how to detect business error, refer to "*Notifying business error*".

- Create a method to handle business exception in the class that performs error handling.

```
@ControllerAdvice
public class ApiGlobalExceptionHandler extends ResponseEntityExceptionHandler
↪{

    // omitted

    // (1)
    @ExceptionHandler(BusinessException.class)
    public ResponseEntity<Object> handleBusinessException(BusinessException␣
↪ex,
            WebRequest request) {
        return handleResultMessagesNotificationException(ex, new␣
↪HttpHeaders(),
                HttpStatus.CONFLICT, request);
    }

    // omitted

}
```

Sr. No.	Description
(1)	Add a method for handling `BusinessException`. `BusinessException` can be handled if `@ExceptionHandler(BusinessException.class)` is specified as method annotation. In the above example, the process is delegated to the method that handles the exception of parent class (`ResultMessagesNotificationException`) of `BusinessException`. Set **409 (Conflict)**in the status code and send a message notifying although there are no errors in the resource itself specified by client, all the conditions necessary for operating the resource stored by the server are not in place.

- Following error response is generated when a business error occurs.

```
HTTP/1.1 409 Conflict
Server: Apache-Coyote/1.1
X-Track: 37c1a899d5f74e7a9c24662292837ef7
Content-Type: application/json;charset=UTF-8
Transfer-Encoding: chunked
Date: Wed, 19 Feb 2014 09:03:26 GMT

{"code":"e.ex.mm.8001","message":"Cannot use specified sign id. sign id :␣
↪user1@test.com"}
```

Implementing exception handling for exclusive errors

An implementation is explained here wherein, an exclusive error is generated and sent as a response.

Exclusive error handling is necessary when performing exclusive control.

For details of exclusive control, refer to "*Exclusive Control*" .

- Create a method for exclusive error handling in the class that performs error handling.

```
@ControllerAdvice
public class ApiGlobalExceptionHandler extends ResponseEntityExceptionHandler
↪{

    // omitted

    // (1)
    @ExceptionHandler({ OptimisticLockingFailureException.class,
            PessimisticLockingFailureException.class })
    public ResponseEntity<Object> handleLockingFailureException(Exception ex,
            WebRequest request) {
        return handleExceptionInternal(ex, null, new HttpHeaders(),
                HttpStatus.CONFLICT, request);
    }

    // omitted

}
```

Sr. No.	Description
(1)	Add a method for handling exclusive errors (`OptimisticLockingFailureException` and `PessimisticLockingFailureException`). If `@ExceptionHandler({ OptimisticLockingFailureException.class,PessimisticLockingFailureExceptic })` is specified as method annotation, exception handling of exclusive errors (`OptimisticLockingFailureException` and `PessimisticLockingFailureException`) can be performed. Set **409(Conflict)** in status code and send a message notifying that, 'although there are no flaws in the resource itself specified by client, the conditions for operating the resource could not be fulfilled due to conflict in the process'.

- When an exclusive error occurs, following error response is generated.

```
HTTP/1.1 409 Conflict
Server: Apache-Coyote/1.1
X-Track: 85200b5a51be42b29840e482ee35087f
Content-Type: application/json;charset=UTF-8
Transfer-Encoding: chunked
Date: Wed, 19 Feb 2014 16:32:45 GMT

{"code":"e.ex.fw.8002","message":"Conflict with other processing occurred."}
```

Implementing exception handling for system errors

An implementation wherein system error is sent as a response on detecting system abnormality, is explained here.

Generate system exception when any system abnormality is detected. Refer to "*Notifying system error*" for the details on how to detect system errors.

- Create a method to handle system exceptions in the class that performs error handling.

```
@ControllerAdvice
public class ApiGlobalExceptionHandler extends ResponseEntityExceptionHandler
↪{

    // omitted

    // (1)
    @ExceptionHandler(Exception.class)
    public ResponseEntity<Object> handleSystemError(Exception ex,
            WebRequest request) {
        return handleExceptionInternal(ex, null, new HttpHeaders(),
                HttpStatus.INTERNAL_SERVER_ERROR, request);
    }

    // omitted

}
```

Sr. No.	Description
(1)	Add a method for handling `Exception`. `Exception` can be handled if `@ExceptionHandler(Exception.class)` is specified as method annotation. In the above example, the system exceptions generated from dependent libraries being used, are also handled. Set **500 (Internal Server Error)** in status code.

- Following error response is generated when a system error occurs.

```
HTTP/1.1 500 Internal Server Error
Server: Apache-Coyote/1.1
X-Track: 3625d5a040a744e49b0a9b3763a24e9c
Content-Type: application/json;charset=UTF-8
Transfer-Encoding: chunked
Date: Wed, 19 Feb 2014 12:22:33 GMT
Connection: close

{"code":"e.ex.fw.9003","message":"System error occurred."}
```

> **Warning: Error message at the time of system error**
>
> When system error occurs, it is recommended that a simple error message that does not specify the error cause, is set as the message to be returned to the client. If an error message specifying the cause of error is set, it may expose system vulnerabilities to the client resulting in security issues.
>
> Cause of error is output to a log, for error analysis. This log is output by `ExceptionLogger` provided by common library as the default setting of Blank project. As a result, the settings and implementation for log output are not required.

Resolving error codes and messages using ExceptionCodeResolver

If `ExceptionCodeResolver` provided by common library is used, error codes can be resolved from the exception class.

Especially, this functionality proves to be convenient, when the cause of error lies at client side and when it is necessary to set the error message corresponding to error cause.

- `applicationContext.xml`

  Mapping exception class and error code (exception code).

```
<!-- omitted -->

<bean id="exceptionCodeResolver"
    class="org.terasoluna.gfw.common.exception.
↪SimpleMappingExceptionCodeResolver">
    <property name="exceptionMappings">
        <map>
            <!-- omitted -->
            <entry key="ResourceNotFoundException"             value="e.ex.
↪fw.5001" />
            <entry key="HttpRequestMethodNotSupportedException" value="e.ex.
↪fw.6001" />
            <entry key="MediaTypeNotAcceptableException"       value="e.ex.
↪fw.6002" />
            <entry key="HttpMediaTypeNotSupportedException"    value="e.ex.
↪fw.6003" />
            <entry key="MethodArgumentNotValidException"       value="e.ex.
↪fw.7001" />
            <entry key="BindException"                         value="e.ex.
↪fw.7002" />
```

```
            <entry key="JsonParseException"                    value="e.ex.
↪fw.7003" />
            <entry key="UnrecognizedPropertyException"         value="e.ex.
↪fw.7004" />
            <entry key="JsonMappingException"                  value="e.ex.
↪fw.7005" />
            <entry key="TypeMismatchException"                 value="e.ex.
↪fw.7006" />
            <entry key="BusinessException"                     value="e.ex.
↪fw.8001" />
            <entry key="OptimisticLockingFailureException"     value="e.ex.
↪fw.8002" />
            <entry key="PessimisticLockingFailureException"    value="e.ex.
↪fw.8002" />
            <entry key="DataAccessException"                   value="e.ex.
↪fw.9002" />
            <!-- omitted -->
        </map>
    </property>
    <property name="defaultExceptionCode" value="e.ex.fw.9001" />
</bean>

<!-- omitted -->
```

Configuration example of message corresponding to error code is shown below.

For message management, refer to "*Message Management*" .

- `xxx-web/src/main/resources/i18n/application-messages.properties`

  Message corresponding to error code (exception code) is set for the error that occurs in application layer.

```
# ---
# Application common messages
# ---
e.ex.fw.5001 = Resource not found.

e.ex.fw.6001 = Request method not supported.
e.ex.fw.6002 = Specified representation format not supported.
e.ex.fw.6003 = Specified media type in the request body not supported.

e.ex.fw.7001 = Validation error occurred on item in the request body.
e.ex.fw.7002 = Validation error occurred on item in the request parameters.
e.ex.fw.7003 = Request body format error occurred.
```

```
e.ex.fw.7004 = Unknown field exists in JSON.
e.ex.fw.7005 = Type mismatch error occurred in JSON field.
e.ex.fw.7006 = Type mismatch error occurred in request parameter or header or␣
↪path variable.

e.ex.fw.8001 = Business error occurred.
e.ex.fw.8002 = Conflict with other processing occurred.

e.ex.fw.9001 = System error occurred.
e.ex.fw.9002 = System error occurred.
e.ex.fw.9003 = System error occurred.

# omitted
```

- `xxx-web/src/main/resources/ValidationMessages.properties`

  Set a message corresponding to error code for the error that occurs in the input validation performed using Bean Validation.

```
# ---
# Bean Validation common messages
# ---

# for bean validation of standard
javax.validation.constraints.AssertFalse.message = "{0}" must be false.
javax.validation.constraints.AssertTrue.message  = "{0}" must be true.
javax.validation.constraints.DecimalMax.message  = "{0}" must be less than $
↪{inclusive == true ? 'or equal to ' : ''}{value}.
javax.validation.constraints.DecimalMin.message  = "{0}" must be greater than
↪${inclusive == true ? 'or equal to ' : ''}{value}.
javax.validation.constraints.Digits.message      = "{0}" numeric value out of␣
↪bounds (<{integer} digits>.<{fraction} digits> expected).
javax.validation.constraints.Future.message      = "{0}" must be in the␣
↪future.
javax.validation.constraints.Max.message         = "{0}" must be less than or␣
↪equal to {value}.
javax.validation.constraints.Min.message         = "{0}" must be greater than␣
↪or equal to {value}.
javax.validation.constraints.NotNull.message     = "{0}" may not be null.
javax.validation.constraints.Null.message        = "{0}" must be null.
javax.validation.constraints.Past.message        = "{0}" must be in the past.
javax.validation.constraints.Pattern.message     = "{0}" must match "{regexp}
↪".
javax.validation.constraints.Size.message        = "{0}" size must be between
↪{min} and {max}.

# for bean validation of hibernate
org.hibernate.validator.constraints.CreditCardNumber.message        = "{0}"␣
↪invalid credit card number.
org.hibernate.validator.constraints.EAN.message                     = "{0}"␣
↪invalid {type} barcode.
```

```
org.hibernate.validator.constraints.Email.message                   = "{0}"␣
↪not a well-formed email address.
org.hibernate.validator.constraints.Length.message                  = "{0}"␣
↪length must be between {min} and {max}.
org.hibernate.validator.constraints.LuhnCheck.message               = "{0}"␣
↪The check digit for ${validatedValue} is invalid, Luhn Modulo 10 checksum␣
↪failed.
org.hibernate.validator.constraints.Mod10Check.message              = "{0}"␣
↪The check digit for ${validatedValue} is invalid, Modulo 10 checksum failed.
org.hibernate.validator.constraints.Mod11Check.message              = "{0}"␣
↪The check digit for ${validatedValue} is invalid, Modulo 11 checksum failed.
org.hibernate.validator.constraints.ModCheck.message                = "{0}"
 The check digit for ${validatedValue} is invalid, ${modType} checksum failed.
org.hibernate.validator.constraints.NotBlank.message                = "{0}"␣
↪may not be empty.
org.hibernate.validator.constraints.NotEmpty.message                = "{0}"␣
↪may not be empty.
org.hibernate.validator.constraints.ParametersScriptAssert.message  = "{0}"␣
↪script expression "{script}" didn't evaluate to true.
org.hibernate.validator.constraints.Range.message                   = "{0}"␣
↪must be between {min} and {max}.
org.hibernate.validator.constraints.SafeHtml.message                = "{0}"␣
↪may have unsafe html content.
org.hibernate.validator.constraints.ScriptAssert.message            = "{0}"␣
↪script expression "{script}" didn't evaluate to true.
org.hibernate.validator.constraints.URL.message                     = "{0}"␣
↪must be a valid URL.

org.hibernate.validator.constraints.br.CNPJ.message                 = "{0}"␣
↪invalid Brazilian corporate taxpayer registry number (CNPJ).
org.hibernate.validator.constraints.br.CPF.message                  = "{0}"␣
↪invalid Brazilian individual taxpayer registry number (CPF).
org.hibernate.validator.constraints.br.TituloEleitoral.message      = "{0}"␣
↪invalid Brazilian Voter ID card number.

# for common library
org.terasoluna.gfw.common.codelist.ExistInCodeList.message   = "{0}" must␣
↪exist in code list of {codeListId}.
```

- `xxx-domain/src/main/resources/i18n/domain-messages.properties`

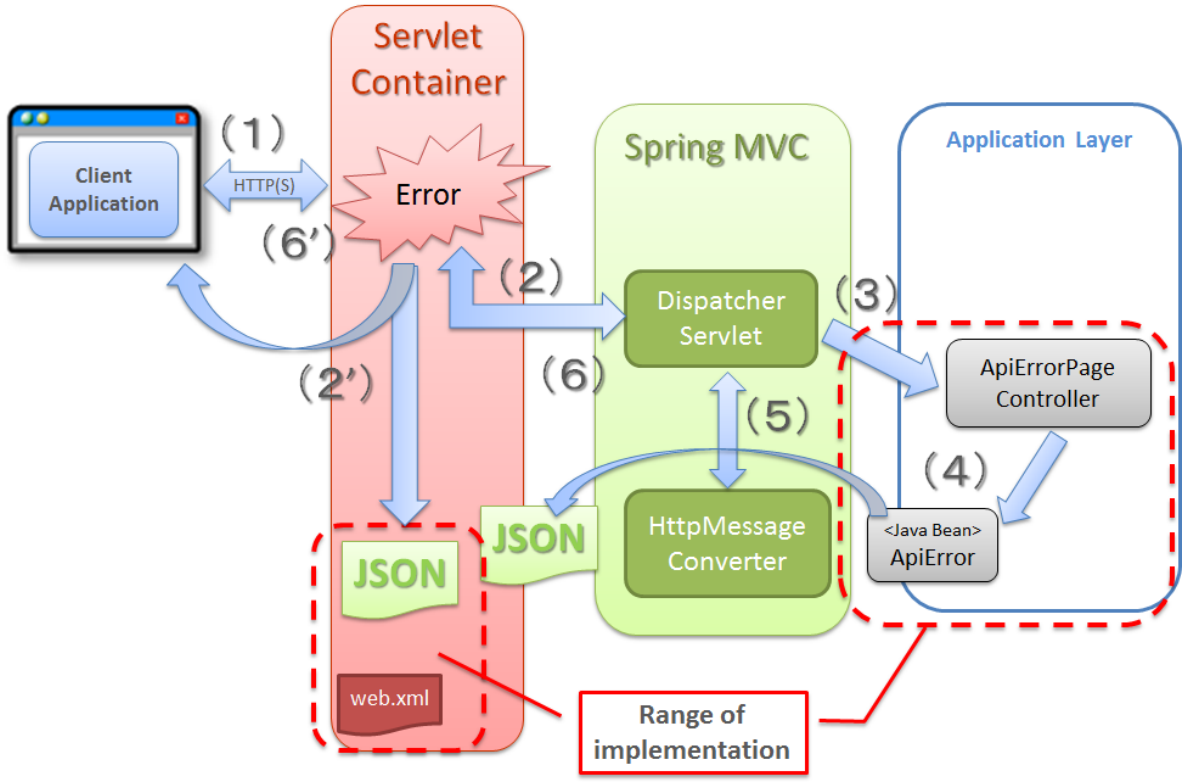
  Set a message corresponding to error code (exception code) for the error in domain layer.

```
# omitted

e.ex.mm.5001 = Specified member not found. member id : {0}
e.ex.mm.8001 = Cannot use specified sign id. sign id : {0}
# omitted
```

Implementing the error handling notified to Servlet Container

When an error occurs in Filter or when error responses are sent by using `HttpServletResponse#sendError`, the errors cannot be handled using exception handling feature of Spring MVC. Hence, these errors are notified to Servlet Container.

This section explains how to handle the errors notified to Servlet Container.



Sr. No.	Processing layer	Description
(1)	Servlet Container (AP Server)	Servlet Container receives a request from the client and performs process. Servlet Container detects an error during the process.
(2)		Servlet Container performs error handling according to the error page definition in `web.xml`. If the error is not fatal, Controller for error handling is called and the error is handled.
(2')		In case of a fatal error, a static JSON file provided in advance is fetched and a response is sent to the client.
(3)	Spring MVC (Framework)	Spring MVC calls the Controller that performs error handling.
(4)	Controller (Common Component)	An error object that retains error information is generated in the Controller and is returned to Spring MVC.
(5)	Spring MVC (Framework)	Spring MVC converts the error object to JSON format message by using `HttpMessageConverter`.
(6)		Spring MVC sets the JSON format error message in response BODY and sends a response to client.

Implementing the Controller that sends error response

Create a controller that sends the error response for the error notified to Servlet Container.

```
package org.terasoluna.examples.rest.api.common.error;

import java.util.HashMap;
import java.util.Map;

import javax.inject.Inject;
import javax.servlet.RequestDispatcher;

import org.springframework.http.HttpStatus;
import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.context.request.RequestAttributes;
import org.springframework.web.context.request.WebRequest;

// (1)
@RequestMapping("error")
@RestController
public class ApiErrorPageController {

    @Inject
    ApiErrorCreator apiErrorCreator; // (2)

    // (3)
    private final Map<HttpStatus, String> errorCodeMap = new HashMap
↪<HttpStatus, String>();

    // (4)
    public ApiErrorPageController() {
        errorCodeMap.put(HttpStatus.NOT_FOUND, "e.ex.fw.5001");
    }

    // (5)
    @RequestMapping
    public ResponseEntity<ApiError> handleErrorPage(WebRequest request) {
        // (6)
        HttpStatus httpStatus = HttpStatus.valueOf((Integer) request
                .getAttribute(RequestDispatcher.ERROR_STATUS_CODE,
                        RequestAttributes.SCOPE_REQUEST));
        // (7)
        String errorCode = errorCodeMap.get(httpStatus);
        // (8)
        ApiError apiError = apiErrorCreator.createApiError(request, errorCode,
                httpStatus.getReasonPhrase());
        // (9)
```

```
        return ResponseEntity.status(httpStatus).body(apiError);
    }

}
```

Sr. No.	Description
(1)	Create a Controller class for sending error response. In the above example, it is mapped to Servlet path "`/api/v1/error`".
(2)	Inject a class that creates error information.
(3)	Create `Map` for mapping HTTP status code and error code.
(4)	Register mapping of HTTP status code and error code.
(5)	Create a handler method that sends error response. Above example is implemented on considering only the case wherein, error page is handled by using a response code (`<error-code>`). Therefore, separate consideration is required for the case where an error page that is handled by using exception type (`<exception-type>`) is to be processed using this method.
(6)	Fetch status code stored in request scope.
(7)	Fetch error code corresponding to status code thus fetched.
(8)	Generate error information corresponding to error codes thus fetched.
(9)	Return the error information generated in (8).

Creating a static JSON file to be sent as response when a fatal error occurs

Create a static JSON file to be sent as a response when a fatal error occurs.

- `unhandledSystemError.json`

```
{"code":"e.ex.fw.9999","message":"Unhandled system error occurred."}
```

Settings for handling an error that is notified to Servlet Container

Settings for handling an error that is notified to Servlet Container are explained here.

- `web.xml`

```
<!-- omitted -->

<!-- (1) -->
<error-page>
    <error-code>404</error-code>
    <location>/api/v1/error</location>
</error-page>

<!-- (2) -->
<error-page>
    <exception-type>java.lang.Exception</exception-type>
    <location>/WEB-INF/views/common/error/unhandledSystemError.json</location>
</error-page>

<!-- (3) -->
<mime-mapping>
    <extension>json</extension>
    <mime-type>application/json;charset=UTF-8</mime-type>
</mime-mapping>

<!-- omitted -->
```

Sr. No.	Description
(1)	Add error page definition for response code if required. In the above example, when error "`404 Not Found`" occurs, Controller (‘‘ ApiErrorPageController‘‘) that is mapped in request “`/api/v1/error`” is called and error response is sent.
(2)	Add definition for handling a fatal error. When a fatal error occurs, it is recommended to respond with the static JSON provided in advance, as double failure may occur during the process that creates response information. In the above example, fixed JSON defined in “`/WEB-INF/views/common/error/unhandledSystemError.json`” is sent as response.
(3)	Specify MIME type of json. When multi byte characters are included in the JSON file created in (2), junk characters may be displayed at client side if `charset=UTF-8` is not specified. When there are no multi byte characters in the JSON file, this setting is not mandatory. However, it is always safe to incorporate this setting.

Note: In Servlet specification,, the behaviour wherein a path is specified which assigns query parameters in `<location>` of `<error-page>` is not defined. Hence, behaviour is likely to change according to AP server. Accordingly, it is not recommended to transfer information to transition destination at the time of error using a query parameter.

- When the request is sent to a non-existing path, following error response is sent.

```
HTTP/1.1 404 Not Found
Server: Apache-Coyote/1.1
X-Track: 2ad50fb5ba2441699c91a5b01edef83f
Content-Type: application/json;charset=UTF-8
Transfer-Encoding: chunked
Date: Wed, 19 Feb 2014 23:24:20 GMT
```

```
{"code":"e.ex.fw.5001","message":"Resource not found."}
```

- When a fatal error occurs, following error response is sent.

```
HTTP/1.1 500 Internal Server Error
Server: Apache-Coyote/1.1
X-Track: 69db3854a19f439781584321d9ce8336
Content-Type: application/json
Content-Length: 68
Date: Thu, 20 Feb 2014 00:13:43 GMT
Connection: close

{"code":"e.ex.fw.9999","message":"Unhandled system error occurred."}
```

Security measures

Implementing security measures for RESTful Web Service is explained here.

This guideline recommends using Spring Security for implementation of security measures.

Authentication and Authorization

Please refer to : ref: *OAuth <OAuth>* for how to achieve authentication and authorization using OAuth 2 (Spring Security OAuth 2).

CSRF measures

- Refer to *CSRF Measures* for the setting methods when CSRF measures are carried out for RESTful Web Service.

- Refer to *Disabling CSRF measures* for the setting methods when CSRF measures are not carried out for RESTful Web Service.

Conditional operations for resource

Todo

TBD

How to implement conditional process control using headers like Etag etc. will be explained in subsequent versions.

Cache control for resource

Todo

TBD

How to implement cache control using headers like Cache-Control/Expires/Pragma etc. will be explained in subsequent versions.

5.1.5 How to extend

Output control for response using @JsonView

Properties within the resource object can be grouped by using `@JsonView`.

Spring Framework implements this function by supporting Jackson function.

For details, refer JacksonJsonViews.

The properties those belong to the specified group alone can be output by specifying the group in Controller.
1 property may also belong to multiple groups.

Example of implementation when Member resource is handled in 2 formats of 'Overview' and 'Details' is as follows.
'Overview format' outputs the main item of Member resource and 'Details format' outputs all items of Member resource.

- `MemberResource.java`

```
package org.terasoluna.examples.rest.api.member;

import java.io.Serializable;

import org.joda.time.DateTime;
import org.joda.time.LocalDate;

import com.fasterxml.jackson.annotation.JsonView;

public class MemberResource implements Serializable {

    private static final long serialVersionUID = 1L;

    // (1)
    interface Summary {
    }

    // (2)
    interface Detail {
    }

    // (3)
    @JsonView({Summary.class, Detail.class})
    private String memberId;

    @JsonView({Summary.class, Detail.class})
    private String firstName;

    @JsonView({Summary.class, Detail.class})
    private String lastName;

    // (4)
    @JsonView(Detail.class)
    private String genderCode;

    @JsonView(Detail.class)
    private LocalDate dateOfBirth;
```

```
    @JsonView(Detail.class)
    private String emailAddress;

    @JsonView(Detail.class)
    private String telephoneNumber;

    @JsonView(Detail.class)
    private String zipCode;

    @JsonView(Detail.class)
    private String address;

    // (5)
    private DateTime createdAt;

    private DateTime lastModifiedAt;

    // omitted setter and getter

}
```

Sr. No.	Description
(1)	A marker interface to specify the group which controls the output is defined. A group to be specified at the time of overview output is defined in the above example.
(2)	A marker interface to specify the group which controls the output is defined. A group to be specified at the time of details output is defined in the above example.
(3)	The items to be output in multiple groups can be assigned to multiple groups by setting arguments to an array and passing multiple marker interfaces. Since the items in the above example are to be assigned to both the groups, overview and details, 2 marker interfaces are set as arguments.
(4)	The items to be output in single group can be assigned to the corresponding group by setting the marker interface to argument. Since there is 1 element in this case, it is not necessary to set it in the array. Since the items in the above example are to be assigned only to the details group, 1 marker interface is set to an argument.
(5)	`@JsonView` is not set in the items those do not belong to the group. It can be changed whether the items those do not belong to the group are to be output, according to the settings. Method of setting is described later.

- `MemberRestController.java`

```
package org.terasoluna.examples.rest.api.member;

import java.util.ArrayList;
import java.util.List;
```

```
import javax.inject.Inject;

import org.dozer.Mapper;
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.ResponseStatus;
import org.springframework.web.bind.annotation.RestController;
import org.terasoluna.examples.rest.domain.model.Member;
import org.terasoluna.examples.rest.domain.service.member.MemberService;

import com.fasterxml.jackson.annotation.JsonView;

@RequestMapping("members")
@RestController
public class MemberRestController {

    @Inject
    MemberService memberService;

    @Inject
    Mapper beanMapper;

    // (1)
    @JsonView(Summary.class)
    @RequestMapping(value = "{memberId}", params = "format=summary", method =␣
↪RequestMethod.GET)
    @ResponseStatus(HttpStatus.OK)
    public MemberResource getMemberSummary(@PathVariable("memberId") String␣
↪memberId) {

        Member member = memberService.getMember(memberId);

        MemberResource responseResource = beanMapper.map(member,
                MemberResource.class);

        return responseResource;
    }

    // (2)
    @JsonView(Detail.class)
    @RequestMapping(value = "{memberId}", params = "format=detail", method =␣
↪RequestMethod.GET)
    @ResponseStatus(HttpStatus.OK)
    public MemberResource getMemberDetail(@PathVariable("memberId") String␣
↪memberId) {

        Member member = memberService.getMember(memberId);
```

```
        MemberResource responseResource = beanMapper.map(member,
                MemberResource.class);

        return responseResource;
    }
}
```

Sr. No.	Description
(1)	Attach `@JsonView` and set the marker interface for the group to be output. Set `Summary` marker interface in the method that outputs the overview.
(2)	Set `Detail` marker interface in the method that outputs the details.

Properties those belong to the group specified in Controller alone are displayed in the body that is output.
The output example is as follows.

- Summary

```
{
  "memberId" : "M000000001",
  "firstName" : "John",
  "lastName" : "Smith",
  "createdAt" : "2014-03-14T11:02:41.477Z",
  "lastModifiedAt" : "2014-03-14T11:02:41.477Z"
}
```

- Detail

```
{
  "memberId" : "M000000001",
  "firstName" : "John",
  "lastName" : "Smith",
```

```
  "genderCode" : "1",
  "dateOfBirth" : "2013-03-14",
  "emailAddress" : "user1394794959984@test.com",
  "telephoneNumber" : "09012345678",
  "zipCode" : "1710051",
  "address" : "Tokyo",
  "createdAt" : "2014-03-14T11:02:41.477Z",
  "lastModifiedAt" : "2014-03-14T11:02:41.477Z"
}
```

Properties without `@JsonView` attached are output if `MapperFeature.DEFAULT_VIEW_INCLUSION` setting is enabled and are not output if disabled.

`MapperFeature.DEFAULT_VIEW_INCLUSION` is enabled in the above output example.

Set as follows when `MapperFeature.DEFAULT_VIEW_INCLUSION` is enabled.

```
<bean id="objectMapper" class="org.springframework.http.converter.json.
↪Jackson2ObjectMapperFactoryBean">
    <!-- ... -->

    <!-- (1) -->
    <property name="featuresToEnable">
        <array>
            <util:constant static-field="com.fasterxml.jackson.databind.
↪MapperFeature.DEFAULT_VIEW_INCLUSION"/>
        </array>
    </property>
</bean>
```

Sr. No.	Description
(1)	Setting is enabled by defining `MapperFeature.DEFAULT_VIEW_INCLUSION` in `featuresToEnable` element.

Set as follows when `MapperFeature.DEFAULT_VIEW_INCLUSION` is to be disabled.

```
<bean id="objectMapper" class="org.springframework.http.converter.json.
↪Jackson2ObjectMapperFactoryBean">
    <!-- ... -->

    <!-- (1) -->
    <property name="featuresToDisable">
        <array>
            <util:constant static-field="com.fasterxml.jackson.databind.
↪MapperFeature.DEFAULT_VIEW_INCLUSION"/>
        </array>
    </property>
</bean>
```

Sr. No.	Description
(1)	Setting is disabled by defining `MapperFeature.DEFAULT_VIEW_INCLUSION` in `featuresToDisable` element.

When `MapperFeature.DEFAULT_VIEW_INCLUSION` is disabled, contents of output in the earlier output example are changed as follows.

- Summary

```
{
  "memberId" : "M000000001",
  "firstName" : "John",
  "lastName" : "Smith"
}
```

- Detail

```
{
  "memberId" : "M000000001",
  "firstName" : "John",
  "lastName" : "Smith",
  "genderCode" : "1",
  "dateOfBirth" : "2013-03-14",
  "emailAddress" : "user1394794959984@test.com",
  "telephoneNumber" : "09012345678",
  "zipCode" : "1710051",
```

```
  "address" : "Tokyo"
}
```

Warning: You need to be careful since the default value when `MapperFeature.DEFAULT_VIEW_INCLUSION` is not specified, varies according to the setting method for `ObjectMapper`. It is also mentioned in *Settings for activating the Spring MVC components necessary for RESTful Web Service* that the default value becomes valid if the direct Bean definition is applied for the method to define Bean for `ObjectMapper`. If `Jackson2ObjectMapperFactoryBean` is used, the default value becomes invalid. For explicitly carrying out settings, it is recommended to include `MapperFeature.DEFAULT_VIEW_INCLUSION` specification for setting with either of the styles.

Note: `@JsonView` is created using the following 2 functions. These are the functions those can be used when the common processes before and after object mapping are to be implemented in the process methods to which `@RequestMapping` in the Controller is added.

- `org.springframework.web.servlet.mvc.method.annotation.RequestBodyAdvice`
- `org.springframework.web.servlet.mvc.method.annotation.ResponseBodyAdvice`

`@ControllerAdvice` can be applied by adding it to the implementation class of these interfaces. Refer *Implementing "@ControllerAdvice"* for the details of `@ControllerAdvice`.

`RequestBodyAdvice` can implement the following method.

- `org.springframework.web.servlet.mvc.method.annotation.RequestBodyAdvice`

Sr. No.	Method name	Overview
(1)	supports	Determine whether this Advice is applied to the request sent. It is applied if `true`.
(2)	handleEmptyBody	It is called before the contents of request body are reflected in the object used in Controller and when the body is empty.

次のページに続く

Table. 5.8 – 前のページからの続き

Sr. No.	Method name	Overview
(3)	beforeBodyRead	It is called before the contents of request body are reflected in the object used in Controller.
(4)	afterBodyRead	It is called after the contents of request body are reflected in the object used in Controller.

When it is not required to describe the processes with all above timings, they can be easily implemented by inheriting `org.springframework.web.servlet.mvc.method.annotation.RequestBodyAdviceAdapter` implemented with the above methods except supports not performing any action and overriding only the necessary portion.

`ResponseBodyAdvice` can implement the following method.

- `org.springframework.web.servlet.mvc.method.annotation.ResponseBodyAdvice`

Sr. No.	Method name	Overview
(1)	supports	Determine whether this Advice is applied to the request sent. It is applied if `true`.
(2)	beforeBodyWrite	It is called before reflecting the return value in the response after the process in Controller is completed.

5.1.6 Appendix

Configuration while using JSR-310 Date and Time API / Joda Time

When JSR-310 Date and Time API is to be used as a property of JavaBean which represents a resource (Resource class), it is not necessary to add a dependency relation since the dependency relation is defined in terasoluna-gfw-common-dependencies. On the contrary, when Joda Time class is to be used, an extension module provided by

Jackson in `pom.xml` is added to dependent library.

When Joda Time class is used

```
<dependency>
    <groupId>org.terasoluna.gfw</groupId>
    <artifactId>terasoluna-gfw-jodatime-dependencies</artifactId>
    <type>pom</type>
</dependency>
```

or

```
<dependency>
    <groupId>com.fasterxml.jackson.datatype</groupId>
    <artifactId>jackson-datatype-joda</artifactId>
</dependency>
```

Note:

In the above setting example, since it is assumed that the dependent library version is managed by the parent project terasoluna-gfw-parent , specifying the version in pom.xml is not necessary. The above jackson-datatype-joda used by terasoluna-gfw-parent is defined by Spring IO Platform.

Besides above, extension modules (`jackson-datatype-xxx`) for handling

- `java.nio.file.Path` added from Java SE 7
- `java.util.Optional` added from Java SE 8
- Objects that are set as Proxy by using Lazy Load function of Hibernate ORM

are provided by another Jackson.

Settings when RESTful Web Service and client application are operated as the same Web application

How to set `DispatcherServlet` for RESTful Web Service

When RESTful Web Service and client application are built as same Web application, it is recommended to divide as `DispatcherServlet` that receives the requests for RESTful Web Service and `DispatcherServlet` that receives client application requests.

How to divide `DispatcherServlet` is explained below.

- `web.xml`

```
<!-- omitted -->

<!-- (1) -->
<servlet>
    <servlet-name>appServlet</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-
↪class>
    <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>classpath*:META-INF/spring/spring-mvc.xml</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>appServlet</servlet-name>
    <url-pattern>/</url-pattern>
</servlet-mapping>

<!-- (2) -->
<servlet>
    <servlet-name>restAppServlet</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-
↪class>
    <init-param>
        <param-name>contextConfigLocation</param-name>
        <!-- (3) -->
        <param-value>classpath*:META-INF/spring/spring-mvc-rest.xml</param-
↪value>
    </init-param>
    <load-on-startup>2</load-on-startup>
</servlet>
<!-- (4) -->
<servlet-mapping>
    <servlet-name>restAppServlet</servlet-name>
    <url-pattern>/api/v1/*</url-pattern>
</servlet-mapping>

<!-- omitted -->
```

Sr. No.	Description
(1)	Request mapping with the `DispatcherServlet` that receives request for client application.
(2)	Add Servlet (`DispatcherServlet`) that receives the request for RESTful Web Service. Specify a name indicating it as RESTful Web Service Servlet in `<servlet-name>` element. In the above example, `"restAppServlet"` is specified as the Servlet name.
(3)	Specify Spring MVC bean definition file to be used while building the `DispatcherServlet` for RESTful Web Service. In the above example, `META-INF/spring/spring-mvc-rest.xml` in class path, is specified as the Spring MVC bean definition file.

(4)	Specify the pattern of servlet path mapped to the `DispatcherServlet`for RESTful Web Service.
	In the above example, the Servlet path under `"/api/v1/"` is mapped in the `DispatcherServlet` for RESTful Web Service.
	Typically, Servlet paths like
	`"/api/v1/"`
	`"/api/v1/members"`
	`"/api/v1/members/xxxxx"`
	are mapped in the `DispatcherServlet` (`"restAppServlet"`) for RESTful Web Service.

Tip: Value to be specified in the value attribute of @RequestMapping annotation

Value in the wild card (`*`) part specified in `<url-pattern>` element, is specified as the value in “value” attribute of `@RequestMapping` annotation.

For example, when `@RequestMapping(value = "members")` is specified, it is deployed as the method that performs process for path `"/api/v1/members"`. Therefore, it is not necessary to specify the path (`"api/v1"`) that performs mapping in divided Servlets, in the value attribute of `@RequestMapping` annotation.

When specified as `@RequestMapping(value = "api/v1/members")`, it ends up being deployed as the method performing the process for path `"/api/v1/api/v1/members"`. Hence, please take note

of same.

Implementing hypermedia link

Implementation for including a hypermedia link to a related resource in JSON, is explained here.

Implementing common parts

- Create a JavaBean that retains the link information.

```
package org.terasoluna.examples.rest.api.common.resource;

import java.io.Serializable;

// (1)
public class Link implements Serializable {

    private static final long serialVersionUID = 1L;

    private final String rel;

    private final String href;

    public Link(String rel, String href) {
        this.rel = rel;
        this.href = href;
    }

    public String getRel() {
        return rel;
    }

    public String getHref() {
        return href;
    }

}
```

Sr. No.	Description
(1)	Create a JavaBean for link information that retains the link name and URL.

- Create an abstract class of Resource that retains collection of link information.

```
package org.terasoluna.examples.rest.api.common.resource;

import java.net.URI;
import java.util.LinkedHashSet;
import java.util.Set;

import com.fasterxml.jackson.annotation.JsonInclude;

// (2)
public abstract class AbstractLinksSupportedResource {

    // (3)
    @JsonInclude(JsonInclude.Include.NON_EMPTY)
    private final Set<Link> links = new LinkedHashSet<>();

    public Set<Link> getLinks() {
        return links;
    }

    // (4)
    public AbstractLinksSupportedResource addLink(String rel, URI href) {
        links.add(new Link(rel, href.toString()));
        return this;
    }

    // (5)
    public AbstractLinksSupportedResource addSelf(URI href) {
        return addLink("self", href);
    }

    // (5)
    public AbstractLinksSupportedResource addParent(URI href) {
        return addLink("parent", href);
    }

}
```

Sr. No.	Description
(2)	Create an abstract class (JavaBean) of the Resource that retains collection of link information. It is assumed that this is the class inherited by the Resource class supporting hypermedia link.
(3)	Define a field that retains information of multiple links. In the above example, `@JsonInclude(JsonInclude.Include.NON_EMPTY)` is specified to prevent output to JSON when link is not specified.
(4)	Create a method to add link information.
(5)	If required, create a method to add common link information. In the above example, methods to add link information for accessing the resource itself (`"self"`) and to add link information for accessing parent resource (`"parent"`) are created.

Implementation for each resource

- Inherit an abstract class of Resource that retains collection of link information in Resource class.

```
package org.terasoluna.examples.rest.api.member;

// (1)
public class MemberResource extends
    AbstractLinksSupportedResource implements Serializable {

    // omitted

}
```

Sr. No.	Description
(1)	Inherit an abstract class of Resource that retains collection of link information. By this, the field (`links`) that retains collection of link information is imported, thus making it a Resource class that supports hypermedia link.

- Adding hypermedia link by REST API process.

```
@RequestMapping("members")
@RestController
public class MemberRestController {

    // omitted

    @RequestMapping(value = "{memberId}", method = RequestMethod.GET)
    @ResponseStatus(HttpStatus.OK)
    public MemberResource getMember(
            @PathVariable("memberId") String memberId
            // (2)
            UriComponentsBuilder uriBuilder) {

        Member member = memberService.getMember(memberId);

        MemberResource responseResource = beanMapper.map(member,
                MemberResource.class);

        // (3)
        responseResource.addSelf(uriBuilder.path("/members").
↪pathSegment(memberId)
                .build().toUri());

        return responseResource;
    }

    // omitted

}
```

Sr. No.	Description
(2)	Specify `org.springframework.web.util.UriComponentsBuilder` class, provided by Spring MVC, in method argument in order to build URI set in link information. When `UriComponentsBuilder` class is specified in method argument of Controller, instance of `org.springframework.web.servlet.support.ServletUriComponentsBuilder` class, that has inherited `UriComponentsBuilder` class in Spring MVC, is passed at the time of method execution.
(3)	Add link information to resource. In the above example, `UriComponentsBuilder` class method is called to build URI set in link information and URI to access to one's resource is added to resource. `ServletUriComponentsBuilder` instances passed as method argument of Controller are initiated based on the information of `<servlet-mapping>` element described in web.xml. It is not resource dependent. By using URI Template Patterns ,etc. provided in Spring Framework, an URI can be built based on the request information. Thus, a generic build process which is independent of a resource, can be implemented. In the above example, when GET is done for `http://example.com/api/v1/members/M000000001`, built URI is same as the requested URI `(http://example.com/api/v1/members/M000000001)`. Method that builds an URI to be set in link information should be implemented as required.

Tip: When building an URL in `ServletUriComponentsBuilder`, by referring to "`X-Forwarded-Host` " header, a configuration with a load balancer or Web server between client and the application server is considered. However, it should be noted that the expected URI will not be built if it does not match with the path configuration.

- Response example

  Following response is obtained in actual operation.

```
GET /rest-api-web/api/v1/members/M000000001 HTTP/1.1
Accept: text/plain, application/json, application/*+json, */*
User-Agent: Java/1.7.0_51
Host: localhost:8080
Connection: keep-alive
```

```
{
  "links" : [ {
    "rel" : "self",
    "href" : "http://localhost:8080/rest-api-web/api/v1/members/M000000001"
  } ],
  "memberId" : "M000000001",
  "firstName" : "John",
  "lastName" : "Smith",
  "genderCode" : "1",
  "dateOfBirth" : "2013-03-14",
  "emailAddress" : "user1394794959984@test.com",
  "telephoneNumber" : "09012345678",
  "zipCode" : "1710051",
  "address" : "Tokyo",
  "credential" : {
    "signId" : "user1394794959984@test.com",
    "passwordLastChangedAt" : "2014-03-14T11:02:41.477Z",
    "lastModifiedAt" : "2014-03-14T11:02:41.477Z"
  },
  "createdAt" : "2014-03-14T11:02:41.477Z",
  "lastModifiedAt" : "2014-03-14T11:02:41.477Z"
}
```

Creating RESTful Web Service conforming to HTTP specifications

A part of REST API implementation explained in this version does not conform to HTTP specifications.

This section explains the implementation that creates the RESTful Web Service conforming to HTTP specifications.

Settings for location header at the time of POST

When conforming to HTTP specifications, it is necessary to set the URI of created resource in the response header of POST method (“Location header”).

Implementation to set the URI of created resource in the response header in case of POST (“Location header”) is

explained here.

Implementation for each resource

- The URI of created resource is set in Location header by REST API process.

```
@RequestMapping("members")
@RestController
public class MemberRestController {

    // omitted

    @RequestMapping(method = RequestMethod.POST)
    public ResponseEntity<MemberResource> postMembers(
            @RequestBody @Validated({ PostMembers.class, Default.class })
            MemberResource requestedResource,
            // (1)
            UriComponentsBuilder uriBuilder) {

        Member creatingMember = beanMapper.map(requestedResource, Member.
↪class);

        Member createdMember = memberService.createMember(creatingMember);

        MemberResource responseResource = beanMapper.map(createdMember,
                MemberResource.class);

        // (2)
        URI createdUri = uriBuilder.path("/members/{memberId}")
                .buildAndExpand(responseResource.getMemberId()).toUri();

        // (3)
        return ResponseEntity.created(createdUri).body(responseResource);
    }

    // omitted

}
```

Sr. No.	Description
(1)	Specify `org.springframework.web.util.UriComponentsBuilder` class provided by Spring MVC, in method argument in order to build an URI of created resource. When `UriComponentsBuilder` class is specified in method argument of Controller, an instance of `org.springframework.web.servlet.support.ServletUriComponentsBuilder` class, that has inherited `UriComponentsBuilder` class by Spring MVC, is passed at the time of method execution.
(2)	Build an URI of created resource. In above example, by `path` method, a path using URI Template Patterns is added to `ServletUriComponentsBuilder` instance passed as argument, `buildAndExpand` method is called and an URI of created resource is built by binding the ID of created resource. `ServletUriComponentsBuilder` instances passed as method argument of Controller are initiated based on the information of `<servlet-mapping>` element described in web.xml. It is not resource dependent. By using URI Template Patterns ,etc. provided by Spring Framework, an URI can be built based on the request information. Thus, a generic build process which is independent of a resource, can be implemented. For example, if POST method is used for `http://example.com/api/v1/members`, the built URI will be “requested URI + `"/"` + ID of created resource”. Typically, if `"M000000001"` is specified in ID, it becomes `http://example.com/api/v1/members/M000000001`. Method that builds an URI to be set in link information should be implemented as required.

次のページに続く

Table. 5.10 – 前のページからの続き

Sr. No.	Description
(3)	Create the `org.springframework.http.ResponseEntity` instance using following parameters, and return it. • Status code : 201(Created) • Location header : Created resource's URI • Response body : Created resource object

Tip: When building an URL in `ServletUriComponentsBuilder`, by referring to "`X-Forwarded-Host` " header, a configuration with a load balancer or Web server between client and the application server is considered. However, it should be noted that the expected URI will not be built if it does not match with the path configuration.

- Response example

  Following response header is obtained in the actual operation.

```
HTTP/1.1 201 Created
Server: Apache-Coyote/1.1
X-Track: 693e132312d64998a7d8d6cabf3d13ef
Location: http://localhost:8080/rest-api-web/api/v1/members/M000000001
Content-Type: application/json;charset=UTF-8
Transfer-Encoding: chunked
Date: Fri, 14 Mar 2014 12:34:31 GMT
```

Disabling CSRF measures

Settings that prevent CSRF measures implementation for RESTful Web Service requests, are explained below.

Tip: Need to use session is eliminated when CSRF measures are not to be implemented.

In the following configuration example, session will not be used in Spring Security process.

CSRF measures are enabled as per the default settings of Blank project. By adding following settings, the process for CSRF measures is prevented for RESTful Web Service requests.

- `spring-security.xml`

```
<!-- omitted -->

<!-- (1) -->
<sec:http
    pattern="/api/v1/**"
```

```
    create-session="stateless">
    <sec:http-basic />
    <sec:csrf disabled="true"/>
</sec:http>

<sec:http>
    <sec:access-denied-handler ref="accessDeniedHandler"/>
    <sec:custom-filter ref="userIdMDCPutFilter" after="ANONYMOUS_FILTER"/>
    <sec:form-login/>
    <sec:logout/>
    <sec:session-management />
</sec:http>

<!-- omitted -->
```

Sr. No.	Description
(1)	Add Spring Security definition for REST API. URL pattern of REST API request path is specified in `pattern` attribute of `<sec:http>` element. In the above example, request path starting with `/api/v1/` is handled as the REST API request path. Further, session is no longer used in Spring Security process by setting `create-session` attribute to `stateless`. Specify `disabled="true"` in `<sec:csrf>` element for invalidating CSRF countermeasures.

Enabling XXE Injection measures

When handling XML format data in RESTful Web Service, it is necessary to implement XXE (XML External Entity) Injection measures.

terasoluna-gfw-web 1.0.1.RELEASE or higher versions are Spring MVC (3.2.10.RELEASE and above) dependent. As these Spring MVC versions implement XXE Injection measures, it is not necessary to implement them individually.

> **Warning: XXE (XML External Entity) Injection measures**
>
> When using terasoluna-gfw-web 1.0.0.RELEASE, a class provided by Spring-oxm should be used, as this version is dependent on the Spring MVC version (3.2.4.RELEASE), that does not implement XXE Injection measures.

Adding Spring-oxm as dependent artifact.

- `pom.xml`

```
<!-- omitted -->

<!-- (1) -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-oxm</artifactId>
    <version>${org.springframework-version}</version> <!-- (2) -->
</dependency>

<!-- omitted -->
```

Sr. No.	Description
(1)	Add Spring-oxm as a dependent artifact.
(2)	Spring version should be fetched from the placeholder (${org.springframework-version}) that manages version number of Spring defined in `pom.xml` of terasoluna-gfw-parent.

Perform Bean definition for mutual conversion between XML and object by using the class provided by Spring-oxm.

- spring-mvc-rest.xml

```
<!-- omitted -->

<!-- (1) -->
<bean id="xmlMarshaller" class="org.springframework.oxm.jaxb.Jaxb2Marshaller">
    <property name="packagesToScan" value="com.examples.app" /> <!-- (2) -->
</bean>

<!-- omitted -->

<mvc:annotation-driven>

    <mvc:message-converters>
        <!-- (3) -->
        <bean class="org.springframework.http.converter.xml.
↪MarshallingHttpMessageConverter">
            <property name="marshaller" ref="xmlMarshaller" /> <!-- (4) -->
            <property name="unmarshaller" ref="xmlMarshaller" /> <!-- (5) -->
        </bean>
    </mvc:message-converters>

    <!-- omitted -->

</mvc:annotation-driven>

<!-- omitted -->
```

Sr. No.	Description
(1)	Perform Bean definition for `Jaxb2Marshaller` provided by Spring-oxm. For `Jaxb2Marshaller`, XXE injection measures are carried out by default.
(2)	Specify package name with JAXB JavaBean (JavaBean assigned with `javax.xml.bind.annotation.XmlRootElement` annotation) stored in `packagesToScan` property. JAXB JavaBean stored under the specified package is scanned and is registered as JavaBean for marshalling and unmarshalling. JavaBean is scanned by the mechanism same as base-package attribute of `<context:component-scan>`.
(3)	Add bean definition of `MarshallingHttpMessageConverter` to `<mvc:message-converters>` element which is the child element of `<mvc:annotation-driven>` element.
(4)	Specify `Jaxb2Marshaller` bean defined in (1) in `marshaller` property.
(5)	Specify `Jaxb2Marshaller` bean defined in (1) in `unmarshaller` property.

How to copy Joda-Time classes using Dozer

How to copy Joda-Time classes (`org.joda.time.DateTime`, `org.joda.time.LocalDate` etc.) using Dozer is explained here.

Create a custom converter for converting the Joda-Time class.

For details of custom converter, refer to "*Bean Mapping (Dozer)*".

- JodaDateTimeConverter.java

```
package org.terasoluna.examples.rest.infra.dozer.converter;

import org.dozer.DozerConverter;
import org.joda.time.DateTime;

public class JodaDateTimeConverter extends DozerConverter<DateTime, DateTime>
 ↪{

    public JodaDateTimeConverter() {
        super(DateTime.class, DateTime.class);
    }

    @Override
    public DateTime convertTo(DateTime source, DateTime destination) {
        // This method not called, because type of from/to is same.
        return convertFrom(source, destination);
    }

    @Override
    public DateTime convertFrom(DateTime source, DateTime destination) {
        return source;
    }

}
```

- JodaLocalDateConverter.java

```
package org.terasoluna.examples.rest.infra.dozer.converter;

import org.dozer.DozerConverter;
import org.joda.time.LocalDate;

public class JodaLocalDateConverter extends
                                    DozerConverter<LocalDate, LocalDate> {

    public JodaLocalDateConverter() {
        super(LocalDate.class, LocalDate.class);
    }

    @Override
    public LocalDate convertTo(LocalDate source, LocalDate destination) {
        // This method not called, because type of from/to is same.
        return convertFrom(source, destination);
    }

    @Override
    public LocalDate convertFrom(LocalDate source, LocalDate destination) {
        return source;
```

```
    }

}
```

Apply the created Custom converter to Dozer.

For details of custom converter, refer to *Bean Mapping (Dozer)*.

```
<!-- (1) -->
<?xml version="1.0" encoding="UTF-8"?>
<mappings xmlns="http://dozer.sourceforge.net" xmlns:xsi="http://www.w3.org/
↪2001/XMLSchema-instance"
    xsi:schemaLocation="
        http://dozer.sourceforge.net http://dozer.sourceforge.net/schema/
↪beanmapping.xsd
    ">

    <configuration>
        <custom-converters>
            <!-- (2) -->
            <converter type="org.terasoluna.examples.rest.infra.dozer.
↪converter.JodaDateTimeConverter">
                <class-a>org.joda.time.DateTime</class-a>
                <class-b>org.joda.time.DateTime</class-b>
            </converter>
            <converter type="org.terasoluna.examples.rest.infra.dozer.
↪converter.JodaLocalDateConverter">
                <class-a>org.joda.time.LocalDate</class-a>
                <class-b>org.joda.time.LocalDate</class-b>
            </converter>
        </custom-converters>
    </configuration>

</mappings>
```

Sr. No.	Description
(1)	Create a file that defines operation settings for Dozer. In this implementation, it is stored in `/xxx-domain/src/main/resources/META-INF/dozer/dozer-configration-mapping.`
(2)	In the above example, custom converter definitions for Joda-Time classes (`org.joda.time.DateTime` and `org.joda.time.LocalDate`) are added.

Note: When Dozer is used in domain layer as well, it is recommended to store the file defining Dozer operation settings, in domain layer project (`xxx-domain`).

When Dozer is used only in application layer, it may be stored in application layer project (`xxx-web`).

Source code for application layer

Out of the application layer source codes used in explanation *How to use*, full version of the source code that was pasted in fragments, is attached herewith.

Sr. No.	Section	File name
(1)	*REST API implementation*	*MemberRestController.java*
(2)	*Implementing exception handling*	*ApiErrorCreator.java*
(3)		*ApiGlobalExceptionHandler.java*

Following files are excluded.

- JavaBean
- Configuration file

MemberRestController.java

java/org/terasoluna/examples/rest/api/member/MemberRestController.java

```
package org.terasoluna.examples.rest.api.member;

import java.util.ArrayList;
import java.util.List;

import javax.inject.Inject;
import javax.validation.groups.Default;

import org.dozer.Mapper;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.PageImpl;
import org.springframework.data.domain.Pageable;
import org.springframework.http.HttpStatus;

import org.springframework.validation.annotation.Validated;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

import org.springframework.web.bind.annotation.ResponseStatus;
import org.springframework.web.bind.annotation.RestController;
import org.terasoluna.examples.rest.api.member.MemberResource.PostMembers;
import org.terasoluna.examples.rest.api.member.MemberResource.PutMember;
import org.terasoluna.examples.rest.domain.model.Member;
import org.terasoluna.examples.rest.domain.service.member.MemberService;

@RequestMapping("members")
@RestController
public class MemberRestController {

    @Inject
    MemberService memberService;

    @Inject
```

```
    Mapper beanMapper;

    @RequestMapping(method = RequestMethod.GET)

    @ResponseStatus(HttpStatus.OK)
    public Page<MemberResource> getMembers(@Validated MembersSearchQuery query,
            Pageable pageable) {

        Page<Member> page = memberService.searchMembers(query.getName(), pageable);

        List<MemberResource> memberResources = new ArrayList<>();
        for (Member member : page.getContent()) {
            memberResources.add(beanMapper.map(member, MemberResource.class));
        }
        Page<MemberResource> responseResource =
            new PageImpl<>(memberResources, pageable, page.getTotalElements());

        return responseResource;
    }

    @RequestMapping(method = RequestMethod.GET)

    @ResponseStatus(HttpStatus.OK)
    public List<MemberResource> getMembers() {
        List<Member> members = memberService.findAll();

        List<MemberResource> memberResources = new ArrayList<>();
        for (Member member : members) {
            memberResources.add(beanMapper.map(member, MemberResource.class));
        }
        return memberResources;
    }

    @RequestMapping(method = RequestMethod.POST)

    @ResponseStatus(HttpStatus.CREATED)
    public MemberResource postMembers(@RequestBody @Validated({
            PostMembers.class, Default.class }) MemberResource requestedResource) {

        Member creatingMember = beanMapper.map(requestedResource, Member.class);

        Member createdMember = memberService.createMember(creatingMember);

        MemberResource responseResource = beanMapper.map(createdMember,
                MemberResource.class);

        return responseResource;
    }

    @RequestMapping(value = "{memberId}", method = RequestMethod.GET)
```

```
    @ResponseStatus(HttpStatus.OK)
    public MemberResource getMember(@PathVariable("memberId") String memberId) {

        Member member = memberService.getMember(memberId);

        MemberResource responseResource = beanMapper.map(member,
                MemberResource.class);

        return responseResource;
    }

    @RequestMapping(value = "{memberId}", method = RequestMethod.PUT)

    @ResponseStatus(HttpStatus.OK)
    public MemberResource putMember(
            @PathVariable("memberId") String memberId,
            @RequestBody @Validated({
            PutMember.class, Default.class }) MemberResource requestedResource) {

        Member updatingMember = beanMapper.map(requestedResource, Member.class);

        Member updatedMember = memberService.updateMember(memberId,
                updatingMember);

        Member updatedMember = memberService.updateMember(memberId,
                MemberResource.class);

        return responseResource;
    }

    @RequestMapping(value = "{memberId}", method = RequestMethod.DELETE)

    @ResponseStatus(HttpStatus.NO_CONTENT)
    public void deleteMember(@PathVariable("memberId") String memberId) {

        memberService.deleteMember(memberId);

    }

}
```

ApiErrorCreator.java

java/org/terasoluna/examples/rest/api/common/error/ApiErrorCreator.java

```
package org.terasoluna.examples.rest.api.common.error;

import javax.inject.Inject;

import org.springframework.context.MessageSource;
import org.springframework.context.support.DefaultMessageSourceResolvable;
import org.springframework.stereotype.Component;
import org.springframework.validation.BindingResult;
import org.springframework.validation.FieldError;
import org.springframework.validation.ObjectError;
import org.springframework.web.context.request.WebRequest;
import org.terasoluna.gfw.common.message.ResultMessage;
import org.terasoluna.gfw.common.message.ResultMessages;

@Component
public class ApiErrorCreator {

    @Inject
    MessageSource messageSource;

    public ApiError createApiError(WebRequest request, String errorCode,
            String defaultErrorMessage, Object... arguments) {
        String localizedMessage = messageSource.getMessage(errorCode,
                arguments, defaultErrorMessage, request.getLocale());
        return new ApiError(errorCode, localizedMessage);
    }

    public ApiError createBindingResultApiError(WebRequest request,
            String errorCode, BindingResult bindingResult,
            String defaultErrorMessage) {
        ApiError apiError = createApiError(request, errorCode,
                defaultErrorMessage);
        for (FieldError fieldError : bindingResult.getFieldErrors()) {
            apiError.addDetail(createApiError(request, fieldError, fieldError
                    .getField()));
        }
        for (ObjectError objectError : bindingResult.getGlobalErrors()) {
            apiError.addDetail(createApiError(request, objectError, objectError
                    .getObjectName()));
        }
        return apiError;
    }

    private ApiError createApiError(WebRequest request,
            DefaultMessageSourceResolvable messageResolvable, String target) {
        String localizedMessage = messageSource.getMessage(messageResolvable,
```

```
                request.getLocale());
        return new ApiError(messageResolvable.getCode(), localizedMessage, target);
    }

    public ApiError createResultMessagesApiError(WebRequest request,
            String rootErrorCode, ResultMessages resultMessages,
            String defaultErrorMessage) {
        ApiError apiError;
        if (resultMessages.getList().size() == 1) {
            ResultMessage resultMessage = resultMessages.iterator().next();
            String errorCode = resultMessage.getCode();
            String errorText = resultMessage.getText();
            if (errorCode == null && errorText == null) {
                errorCode = rootErrorCode;
            }
            apiError = createApiError(request, errorCode, errorText,
                    resultMessage.getArgs());
        } else {
            apiError = createApiError(request, rootErrorCode,
                    defaultErrorMessage);
            for (ResultMessage resultMessage : resultMessages.getList()) {
                apiError.addDetail(createApiError(request, resultMessage
                        .getCode(), resultMessage.getText(), resultMessage
                        .getArgs()));
            }
        }
        return apiError;
    }

}
```

ApiGlobalExceptionHandler.java

java/org/terasoluna/examples/rest/api/common/error/ApiGlobalExceptionHandler.java

```
package org.terasoluna.examples.rest.api.common.error;

import javax.inject.Inject;

import org.springframework.dao.OptimisticLockingFailureException;
import org.springframework.dao.PessimisticLockingFailureException;
import org.springframework.http.HttpHeaders;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
```

```
import org.springframework.http.converter.HttpMessageNotReadableException;
import org.springframework.validation.BindException;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.MethodArgumentNotValidException;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.context.request.WebRequest;
import org.springframework.web.servlet.mvc.method.annotation.
↪ResponseEntityExceptionHandler;
import org.terasoluna.gfw.common.exception.BusinessException;
import org.terasoluna.gfw.common.exception.ExceptionCodeResolver;
import org.terasoluna.gfw.common.exception.ResourceNotFoundException;
import org.terasoluna.gfw.common.exception.ResultMessagesNotificationException;

@ControllerAdvice
public class ApiGlobalExceptionHandler extends ResponseEntityExceptionHandler {

    @Inject
    ApiErrorCreator apiErrorCreator;

    @Inject
    ExceptionCodeResolver exceptionCodeResolver;

    @Override
    protected ResponseEntity<Object> handleExceptionInternal(Exception ex,
            Object body, HttpHeaders headers, HttpStatus status,
            WebRequest request) {
        final Object apiError;
        if (body == null) {
            String errorCode = exceptionCodeResolver.resolveExceptionCode(ex);
            apiError = apiErrorCreator.createApiError(request, errorCode, ex
                    .getLocalizedMessage());
        } else {
            apiError = body;
        }
        return ResponseEntity.status(status).headers(headers).body(apiError);
    }

    @Override
    protected ResponseEntity<Object> handleMethodArgumentNotValid(
            MethodArgumentNotValidException ex, HttpHeaders headers,
            HttpStatus status, WebRequest request) {
        return handleBindingResult(ex, ex.getBindingResult(), headers, status,
                request);
    }

    @Override
    protected ResponseEntity<Object> handleBindException(BindException ex,
            HttpHeaders headers, HttpStatus status, WebRequest request) {
        return handleBindingResult(ex, ex.getBindingResult(), headers, status,
```

```
            request);
    }

    private ResponseEntity<Object> handleBindingResult(Exception ex,
            BindingResult bindingResult, HttpHeaders headers,
            HttpStatus status, WebRequest request) {
        String errorCode = exceptionCodeResolver.resolveExceptionCode(ex);
        ApiError apiError = apiErrorCreator.createBindingResultApiError(
                request, errorCode, bindingResult, ex.getMessage());
        return handleExceptionInternal(ex, apiError, headers, status, request);
    }

    @Override
    protected ResponseEntity<Object> handleHttpMessageNotReadable(
            HttpMessageNotReadableException ex, HttpHeaders headers,
            HttpStatus status, WebRequest request) {
        if (ex.getCause() instanceof Exception) {
            return handleExceptionInternal((Exception) ex.getCause(), null,
                    headers, status, request);
        } else {
            return handleExceptionInternal(ex, null, headers, status, request);
        }
    }

    @ExceptionHandler(ResourceNotFoundException.class)
    public ResponseEntity<Object> handleResourceNotFoundException(
            ResourceNotFoundException ex, WebRequest request) {
        return handleResultMessagesNotificationException(ex, new HttpHeaders(),
                HttpStatus.NOT_FOUND, request);
    }

    @ExceptionHandler(BusinessException.class)
    public ResponseEntity<Object> handleBusinessException(BusinessException ex,
            WebRequest request) {
        return handleResultMessagesNotificationException(ex, new HttpHeaders(),
                HttpStatus.CONFLICT, request);
    }

    private ResponseEntity<Object> handleResultMessagesNotificationException(
            ResultMessagesNotificationException ex, HttpHeaders headers,
            HttpStatus status, WebRequest request) {
        String errorCode = exceptionCodeResolver.resolveExceptionCode(ex);
        ApiError apiError = apiErrorCreator.createResultMessagesApiError(
                request, errorCode, ex.getResultMessages(), ex.getMessage());
        return handleExceptionInternal(ex, apiError, headers, status, request);
    }

    @ExceptionHandler({ OptimisticLockingFailureException.class,
            PessimisticLockingFailureException.class })
    public ResponseEntity<Object> handleLockingFailureException(Exception ex,
```

```
            WebRequest request) {
        return handleExceptionInternal(ex, null, new HttpHeaders(),
                HttpStatus.CONFLICT, request);
    }

    @ExceptionHandler(Exception.class)
    public ResponseEntity<Object> handleSystemError(Exception ex,
            WebRequest request) {
        return handleExceptionInternal(ex, null, new HttpHeaders(),
                HttpStatus.INTERNAL_SERVER_ERROR, request);
    }

}
```

Source code of the domain layer class created at the time of REST API implementation

Source code of domain layer class called from REST API explained in *How to use* is attached herewith.

Also, infrastructure layer is implemented by using MyBatis3.

Sr. No.	Classification	File name
(1)	model	*Member.java*
(2)		*MemberCredentia.java*
(3)		*Gender.java*
(4)	repository	*MemberRepository.java*
(5)	service	*MemberService.java*
(6)		*MemberServiceImpl.java*
(7)	other	*DomainMessageCodes.java*
(8)		*GenderTypeHandler.java*
(9)		*member-mapping.xml*
(10)		*mybatis-config.xml*
(11)		*MemberRepository.xml*

Following files are excluded.

- JavaBean other than Entity
- Configuration files other than Dozer

Member.java

java/org/terasoluna/examples/rest/domain/model/Member.java

```
package org.terasoluna.examples.rest.domain.model;

import java.io.Serializable;
import org.joda.time.DateTime;
import org.joda.time.LocalDate;

public class Member implements Serializable {

    private static final long serialVersionUID = 1L;

    private String memberId;

    private String firstName;

    private String lastName;

    private Gender gender;

    private LocalDate dateOfBirth;

    private String emailAddress;

    private String telephoneNumber;

    private String zipCode;

    private String address;

    private DateTime createdAt;

    private DateTime lastModifiedAt;

    private long version;

    private MemberCredential credential;
```

```
    public String getMemberId() {
        return memberId;
    }

    public void setMemberId(String memberId) {
        this.memberId = memberId;
    }

    public String getFirstName() {
        return firstName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

    public Gender getGender() {
        return gender;
    }

    public void setGender(Gender gender) {
        this.gender = gender;
    }

    public String getGenderCode() {
        if (gender == null) {
            return null;
        } else {
            return gender.getCode();
        }
    }

    public void setGenderCode(String genderCode) {
        this.gender = Gender.getByCode(genderCode);
    }

    public LocalDate getDateOfBirth() {
        return dateOfBirth;
    }

    public void setDateOfBirth(LocalDate dateOfBirth) {
```

```
        this.dateOfBirth = dateOfBirth;
    }

    public String getEmailAddress() {
        return emailAddress;
    }

    public void setEmailAddress(String emailAddress) {
        this.emailAddress = emailAddress;
    }

    public String getTelephoneNumber() {
        return telephoneNumber;
    }

    public void setTelephoneNumber(String telephoneNumber) {
        this.telephoneNumber = telephoneNumber;
    }

    public String getZipCode() {
        return zipCode;
    }

    public void setZipCode(String zipCode) {
        this.zipCode = zipCode;
    }

    public String getAddress() {
        return address;
    }

    public void setAddress(String address) {
        this.address = address;
    }

    public DateTime getCreatedAt() {
        return createdAt;
    }

    public void setCreatedAt(DateTime createdAt) {
        this.createdAt = createdAt;
    }

    public DateTime getLastModifiedAt() {
        return lastModifiedAt;
    }

    public void setLastModifiedAt(DateTime lastModifiedAt) {
        this.lastModifiedAt = lastModifiedAt;
    }
```

```
    public long getVersion() {
        return version;
    }

    public void setVersion(long version) {
        this.version = version;
    }

    public MemberCredential getCredential() {
        return credential;
    }

    public void setCredential(MemberCredential credential) {
        this.credential = credential;
    }

}
```

MemberCredentia.java

java/org/terasoluna/examples/rest/domain/model/MemberCredential.java

```
package org.terasoluna.examples.rest.domain.model;

import java.io.Serializable;
import org.joda.time.DateTime;

public class MemberCredential implements Serializable {

    private static final long serialVersionUID = 1L;

    private String memberId;

    private String signId;

    private String password;

    private String previousPassword;

    private DateTime passwordLastChangedAt;

    private DateTime lastModifiedAt;
```

```
    private long version;

    public String getMemberId() {
        return memberId;
    }

    public void setMemberId(String memberId) {
        this.memberId = memberId;
    }

    public String getSignId() {
        return signId;
    }

    public void setSignId(String signId) {
        this.signId = signId;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public String getPreviousPassword() {
        return previousPassword;
    }

    public void setPreviousPassword(String previousPassword) {
        this.previousPassword = previousPassword;
    }

    public DateTime getPasswordLastChangedAt() {
        return passwordLastChangedAt;
    }

    public void setPasswordLastChangedAt(DateTime passwordLastChangedAt) {
        this.passwordLastChangedAt = passwordLastChangedAt;
    }

    public DateTime getLastModifiedAt() {
        return lastModifiedAt;
    }

    public void setLastModifiedAt(DateTime lastModifiedAt) {
        this.lastModifiedAt = lastModifiedAt;
    }
```

```
    public long getVersion() {
        return version;
    }

    public void setVersion(long version) {
        this.version = version;
    }

}
```

Gender.java

java/org/terasoluna/examples/rest/domain/model/Gender.java

```
package org.terasoluna.examples.rest.domain.model;

import java.util.Collections;
import java.util.HashMap;
import java.util.Map;

import org.springframework.util.Assert;

public enum Gender {

    UNKNOWN("0"), MEN("1"), WOMEN("2");

    private static final Map<String, Gender> genderMap;

    static {
        Map<String, Gender> map = new HashMap<>();
        for (Gender gender : values()) {
            map.put(gender.code, gender);
        }
        genderMap = Collections.unmodifiableMap(map);
    }

    private final String code;

    private Gender(String code) {
        this.code = code;
    }

    public static Gender getByCode(String code) {
        Gender gender = genderMap.get(code);
```

```
        Assert.notNull(gender, "gender code is invalid. code : " + code);
        return gender;
    }

    public String getCode() {
        return code;
    }

}
```

MemberRepository.java

java/org/terasoluna/examples/rest/domain/repository/member/MemberRepository.java

```
package org.terasoluna.examples.rest.domain.repository.member;

import java.util.List;
import org.apache.ibatis.session.RowBounds;

import org.terasoluna.examples.rest.domain.model.Member;

public interface MemberRepository {

    Member findOne(String memberId);

    List<Member> findAll();

    long countByContainsName(String name);
    List<Member> findPageByContainsName(String name, RowBounds rowBounds);

    void createMember(Member creatingMember);
    void createCredential(Member creatingMember);

    boolean updateMember(Member updatingMember);

    void deleteMember(String memberId);
    void deleteCredential(String memberId);

}
```

MemberService.java

java/org/terasoluna/examples/rest/domain/service/member/MemberService.java

```
package org.terasoluna.examples.rest.domain.service.member;

import java.util.List;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.terasoluna.examples.rest.domain.model.Member;

public interface MemberService {

    List<Member> findAll();

    Page<Member> searchMembers(String name, Pageable pageable);

    Member getMember(String memberId);

    Member createMember(Member creatingMember);

    Member updateMember(String memberId, Member updatingMember);

    void deleteMember(String memberId);

}
```

MemberServiceImpl.java

java/org/terasoluna/examples/rest/domain/service/member/MemberServiceImpl.java

```
package org.terasoluna.examples.rest.domain.service.member;

import java.util.ArrayList;
import java.util.List;
import javax.inject.Inject;
import org.apache.ibatis.session.RowBounds;
import org.dozer.Mapper;
import org.joda.time.DateTime;
import org.springframework.dao.DuplicateKeyException;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.PageImpl;
import org.springframework.data.domain.Pageable;
import org.springframework.orm.ObjectOptimisticLockingFailureException;
```

```
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import org.springframework.util.StringUtils;
import org.terasoluna.examples.rest.domain.message.DomainMessageCodes;
import org.terasoluna.examples.rest.domain.model.Member;
import org.terasoluna.examples.rest.domain.model.MemberCredential;
import org.terasoluna.examples.rest.domain.repository.member.MemberRepository;
import org.terasoluna.gfw.common.date.jodatime.JodaTimeDateFactory;
import org.terasoluna.gfw.common.exception.BusinessException;
import org.terasoluna.gfw.common.exception.ResourceNotFoundException;
import org.terasoluna.gfw.common.message.ResultMessages;

@Transactional
@Service
public class MemberServiceImpl implements MemberService {

    @Inject
    MemberRepository memberRepository;

    @Inject
    JodaTimeDateFactory dateFactory;

    @Inject
    PasswordEncoder passwordEncoder;

    @Inject
    Mapper beanMapper;

    @Override
    @Transactional(readOnly = true)
    public List<RestMember> findAll() {
        return restMemberRepository.findAll();
    }

    @Override
    @Transactional(readOnly = true)
    public Page<Member> searchMembers(String name, Pageable pageable) {
        List<Member> members = null;
        // Count Members by search criteria
        long total = memberRepository.countByContainsName(name);
        if (0 < total) {
             RowBounds rowBounds = new RowBounds(pageable.getOffset(), pageable.
↪getPageSize());
             members = memberRepository.findPageByContainsName(name, rowBounds);
        } else {
            members = new ArrayList<Member>();
        }
        return new PageImpl<Member>(members, pageable, total);
    }
```

```
    @Override
    @Transactional(readOnly = true)
    public Member getMember(String memberId) {
        // find member
        Member member = memberRepository.findOne(memberId);
        if (member == null) {
            // If member is not exists
            throw new ResourceNotFoundException(ResultMessages.error().add(
                            DomainMessageCodes.E_EX_MM_5001, memberId));
        }
        return member;
    }

    @Override
    public Member createMember(Member creatingMember) {
        MemberCredential creatingCredential = creatingMember
                            .getCredential();

        // get processing current date time
        DateTime currentDateTime = dateFactory.newDateTime();

        creatingMember.setCreatedAt(currentDateTime);
        creatingMember.setLastModifiedAt(currentDateTime);

        // decide sign id(email-address)
        String signId = creatingCredential.getSignId();
        if (!StringUtils.hasLength(signId)) {
            signId = creatingMember.getEmailAddress();
            creatingCredential.setSignId(signId.toLowerCase());
        }

        // encrypt password
        String rawPassword = creatingCredential.getPassword();
        creatingCredential.setPassword(passwordEncoder.encode(rawPassword));
        creatingCredential.setPasswordLastChangedAt(currentDateTime);
        creatingCredential.setLastModifiedAt(currentDateTime);

        // save member & member credential
        try {

            // Registering member details
            memberRepository.createMember(creatingMember);
            // //Registering credential details
            memberRepository.createCredential(creatingMember);
            return creatingMember;
        } catch (DuplicateKeyException e) {
            // If sign id is already used
            throw new BusinessException(ResultMessages.error().add(
                            DomainMessageCodes.E_EX_MM_8001,
```

```
                            creatingCredential.getSignId()), e);
        }
    }

    @Override
    public Member updateMember(String memberId, Member updatingMember) {
        // get member
        Member member = getMember(memberId);

        // override updating member attributes
        beanMapper.map(updatingMember, member, "member.update");

        // get processing current date time
        DateTime currentDateTime = dateFactory.newDateTime();
        member.setLastModifiedAt(currentDateTime);

        // save updating member
        boolean updated = memberRepository.updateMember(member);
        if (!updated) {
                throw new ObjectOptimisticLockingFailureException(Member.class,
                                member.getMemberId());
        }
        return member;
    }

    @Override
    public void deleteMember(String memberId) {

        // First Delete from credential (Child)
        memberRepository.deleteCredential(memberId);
        // Delete member
        memberRepository.deleteMember(memberId);
    }

}
```

DomainMessageCodes.java

java/org/terasoluna/examples/rest/domain/message/DomainMessageCodes.java

```
package org.terasoluna.examples.rest.domain.message;

/**
 * Message codes of domain layer message.
```

```
 * @author DomainMessageCodesGenerator
 */
public class DomainMessageCodes {

    private DomainMessageCodes() {
        // NOP
    }

    /** e.ex.mm.5001=Specified member not found. member id : {0} */
    public static final String E_EX_MM_5001 = "e.ex.mm.5001";

    /** e.ex.mm.8001=Cannot use specified sign id. sign id : {0} */
    public static final String E_EX_MM_8001 = "e.ex.mm.8001";
}
```

GenderTypeHandler.java

This is a TypeHandler for mapping the Enum type code value.

java/org/terasoluna/examples/infra/mybatis/typehandler/GenderTypeHandler.java

```
package org.terasoluna.examples.infra.mybatis.typehandler;

import java.sql.CallableStatement;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import org.terasoluna.examples.domain.model.Gender;
import org.apache.ibatis.type.JdbcType;
import org.apache.ibatis.type.BaseTypeHandler;

public class GenderTypeHandler extends BaseTypeHandler<Gender> {

    @Override
    public Gender getNullableResult(ResultSet rs, String columnName) throws␣
↪SQLException {
            return getByCode(rs.getString(columnName));
    }

    @Override
    public Gender getNullableResult(ResultSet rs, int columnIndex) throws␣
↪SQLException {
```

```
            return getByCode(rs.getString(columnIndex));
    }

    @Override
    public Gender getNullableResult(CallableStatement cs, int columnIndex)
                    throws SQLException {
            return getByCode(cs.getString(columnIndex));
    }

    @Override
    public void setNonNullParameter(PreparedStatement ps, int i,
                    Gender parameter, JdbcType jdbcType) throws SQLException {
            ps.setString(i, parameter.getCode());
    }

    private Gender getByCode(String byCode) {
            if (byCode == null) {
                return null;
            } else {
                return Gender.getByCode(byCode);
            }
    }
}
```

member-mapping.xml

In the implemented Service class, "*Bean Mapping (Dozer)*" is used while copying the value specified by client in `Member` object.

When it is alright to simply copy the field values, Bean mapping definition need not be added. However, in this implementation it needs to be ensured that, items which are not to be updated (`memberId`, `credential`, `createdAt` and `version`) are not copied.

Bean mapping definition needs to be added in order to ensure that specific fields are not copied.

`resources/META-INF/dozer/member-mapping.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<mappings xmlns="http://dozer.sourceforge.net" xmlns:xsi="http://www.w3.org/2001/
↪XMLSchema-instance"
    xsi:schemaLocation="http://dozer.sourceforge.net
          http://dozer.sourceforge.net/schema/beanmapping.xsd">
```

```
    <mapping map-id="member.update">
        <class-a>org.terasoluna.examples.rest.domain.model.Member</class-a>
        <class-b>org.terasoluna.examples.rest.domain.model.Member</class-b>
        <field-exclude>
            <a>memberId</a>
            <b>memberId</b>
        </field-exclude>
        <field-exclude>
            <a>credential</a>
            <b>credential</b>
        </field-exclude>
        <field-exclude>
            <a>createdAt</a>
            <b>createdAt</b>
        </field-exclude>
        <field-exclude>
            <a>lastModifiedAt</a>
            <b>lastModifiedAt</b>
        </field-exclude>
        <field-exclude>
            <a>version</a>
            <b>version</b>
        </field-exclude>
    </mapping>

</mappings>
```

mybatis-config.xml

MyBatis3 operations can be customized by adding a configuration values in MyBatis configuration file.
MyBatis3 does not support Joda-time classes (org.joda.time.DateTime, org.joda.time.LocalDateTime, org.joda.time.LocalDate etc.).

Hence, when Joda-Time class is used in the field of Entity class, it is necessary to provide a `TypeHandler` for Joda-Time.

`TypeHandler` implementation for mapping the org.joda.time.DateTime with java.sql.Timestamp is done by referring [*Implementing TypeHandler for Joda-Time*].

`resources/META-INF/mybatis/mybatis-config.xml`

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE configuration PUBLIC "-//mybatis.org/DTD Config 3.0//EN"
    "http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>

    <settings>
        <setting name="jdbcTypeForNull" value="NULL" />
        <setting name="mapUnderscoreToCamelCase" value="true" />
    </settings>

    <typeAliases>
        <package name="org.terasoluna.examples.infra.mybatis.typehandler" />
    </typeAliases>

    <typeHandlers>
       <package name="org.terasoluna.examples.infra.mybatis.typehandler" />
    </typeHandlers>

</configuration>
```

MemberRepository.xml

resources/org/terasoluna/examples/rest/domain/repository/member/MemberRepository.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
    "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper
    namespace="org.terasoluna.examples.rest.domain.repository.member.
↪MemberRepository">

    <resultMap id="MemberResultMap" type="Member">
        <id property="memberId" column="member_id" />
        <result property="firstName" column="first_name" />
        <result property="lastName" column="last_name" />
        <result property="gender" column="gender" />
        <result property="dateOfBirth" column="date_of_birth" />
        <result property="emailAddress" column="email_address" />
        <result property="telephoneNumber" column="telephone_number" />
        <result property="zipCode" column="zip_code" />
        <result property="address" column="address" />
        <result property="createdAt" column="created_at" />
        <result property="lastModifiedAt" column="last_modified_at" />
        <result property="version" column="version" />
```

```
        <result property="credential.memberId" column="member_id" />
        <result property="credential.signId" column="sign_id" />
        <result property="credential.password" column="password" />
        <result property="credential.previousPassword" column="previous_password" /
↪>
        <result property="credential.passwordLastChangedAt" column="password_last_
↪changed_at" />
        <result property="credential.lastModifiedAt" column="credential_last_
↪modified_at" />
        <result property="credential.version" column="credential_version" />
    </resultMap>

    <sql id="selectMember">
        SELECT
         member.member_id as member_id
         ,member.first_name as first_name
         ,member.last_name as last_name
         ,member.gender as gender
         ,member.date_of_birth as date_of_birth
         ,member.email_address as email_address
         ,member.telephone_number as telephone_number
         ,member.zip_code as zip_code
         ,member.address as address
         ,member.created_at as created_at
         ,member.last_modified_at as last_modified_at
         ,member.version as version
         ,credential.sign_id as sign_id
         ,credential.password as password
         ,credential.previous_password as previous_password
         ,credential.password_last_changed_at as password_last_changed_at
         ,credential.last_modified_at as credential_last_modified_at
         ,credential.version as credential_version
        FROM
         t_member member
         INNER JOIN t_member_credential credential ON credential.member_id =␣
↪member.member_id
    </sql>

    <sql id="whereMember">
        WHERE
            member.first_name LIKE #{nameContainingCondition} ESCAPE '~'
            OR member.last_name LIKE #{nameContainingCondition} ESCAPE '~'
    </sql>

    <select id="findAll" resultMap="RestMemberResultMap">
        <include refid="selectRestMember" />
        ORDER BY member_id ASC
    </select>

        <select id="findOne" parameterType="string" resultMap="MemberResultMap">
```

```
        <include refid="selectMember" />
        WHERE
        member.member_id = #{memberId}
    </select>

    <select id="countByContainsName" parameterType="string" resultType="_long">
        <bind name="nameContainingCondition"
        value="@org.terasoluna.gfw.common.query.
↪QueryEscapeUtils@toStartingWithCondition(_parameter)" />
        SELECT
        COUNT(*)
        FROM
        t_member member
        <include refid="whereMember" />
    </select>

    <select id="findPageByContainsName" parameterType="string"
        resultMap="MemberResultMap">
        <bind name="nameContainingCondition"
        value="@org.terasoluna.gfw.common.query.
↪QueryEscapeUtils@toStartingWithCondition(_parameter)" />
        <include refid="selectMember" />
        <include refid="whereMember" />
        ORDER BY member_id ASC
    </select>

    <insert id="createMember" parameterType="Member">
        <selectKey keyProperty="memberId" resultType="string" order="BEFORE">
            SELECT 'M'||TO_CHAR(NEXTVAL('s_member'),'FM000000000')
        </selectKey>
        INSERT INTO
        t_member
        (
        member_id
        ,first_name
        ,last_name
        ,gender
        ,date_of_birth
        ,email_address
        ,telephone_number
        ,zip_code
        ,address
        ,created_at
        ,last_modified_at
        ,version
        )
        VALUES
        (
        #{memberId}
        ,#{firstName}
```

```
        ,#{lastName}
        ,#{gender}
        ,#{dateOfBirth}
        ,#{emailAddress}
        ,#{telephoneNumber}
        ,#{zipCode}
        ,#{address}
        ,#{createdAt}
        ,#{lastModifiedAt}
        ,1
        )
    </insert>

    <insert id="createCredential" parameterType="Member">
        INSERT INTO
        t_member_credential
        (
        member_id
        ,sign_id
        ,password
        ,previous_password
        ,password_last_changed_at
        ,last_modified_at
        ,version
        )
        VALUES
        (
        #{memberId}
        ,#{credential.signId}
        ,#{credential.password}
        ,#{credential.previousPassword}
        ,#{credential.passwordLastChangedAt}
        ,#{credential.lastModifiedAt}
        ,1
        )
    </insert>

    <update id="updateMember" parameterType="Member">
        UPDATE
            t_member
        SET
            first_name = #{firstName}
            ,last_name = #{lastName}
            ,gender = #{gender}
            ,date_of_birth = #{dateOfBirth}
            ,email_address = #{emailAddress}
            ,telephone_number = #{telephoneNumber}
            ,zip_code = #{zipCode}
            ,address = #{address}
            ,created_at = #{createdAt}
```

```
            ,last_modified_at = #{lastModifiedAt}
            ,version = version + 1
        WHERE
            member_id = #{memberId}
            AND version = #{version}
    </update>

    <delete id="deleteCredential" parameterType="string">
        DELETE FROM t_member_credential
        WHERE
        member_id = #{memberId}
    </delete>

    <delete id="deleteMember" parameterType="string">
        DELETE FROM t_member
        WHERE
        member_id = #{memberId}
    </delete>

</mapper>
```

5.2 REST Client (HTTP Client)

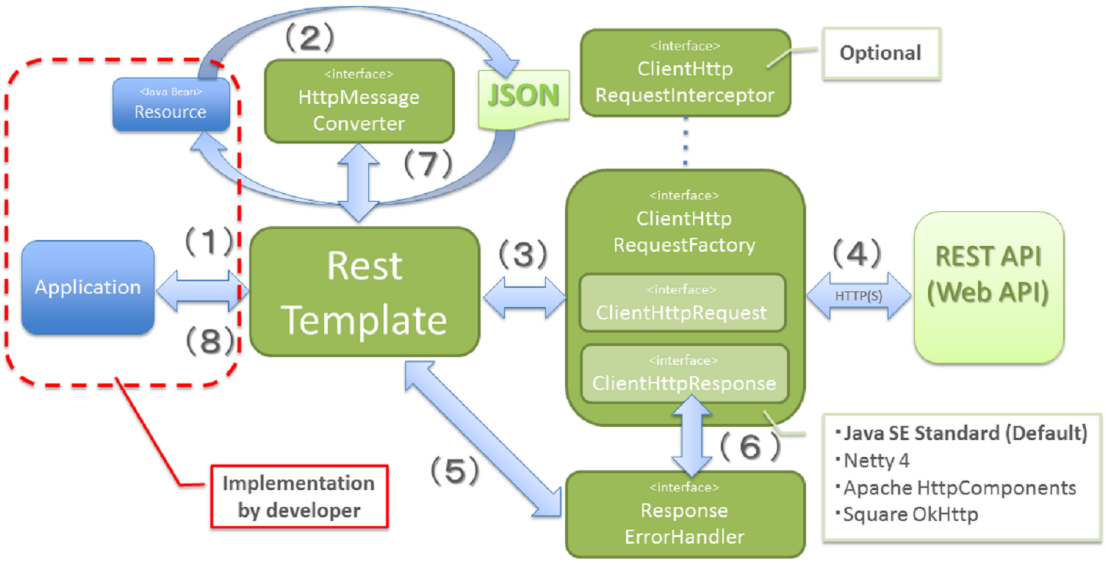
5.2.1 Overview

This chapter explains how to call RESTful Web Service(REST API) by using `org.springframework.web.client.RestTemplate` offered by Spring Framework.

What is `RestTemplate`

`RestTemplate` is a class which offers a method for calling REST API(Web API) and is a HTTP client offered by Spring Framework.

The method by which `RestTemplate` access REST API (Web API) is explained before explaining basic implementation method.



Sr. No.	Component	Description
(1)	Application	Call `RestTemplate` method and request to call REST API (Web API).
(2)	`RestTemplate`	By using `HttpMessageConverter`, convert Java object to message (JSON etc.) which is to be configured in the request body.
(3)		Fetch `ClientHttpRequest` from `ClientHttpRequestFactory` and request to send a message (JSON etc.).
(4)	`ClientHttpRequest`	Configure message (JSON etc) in the request body and carry out request in REST API (Web API) through HTTP.
(5)	`RestTemplate`	Determine errors and perform error handling for HTTP transmission using `ResponseErrorHandler`.
(6)	`ResponseErrorHandler`	Fetch response data from `ClientHttpResponse`, determine errors and perform error handling.
(7)	`RestTemplate`	By using `HttpMessageConverter`, convert message configured in response body (JSON etc) to Java object.
(8)		Return results (Java object) of calling REST API (Web API) to the application.

Note: Handling asynchronous processing

When response received from REST API is to be processed in another thread (asynchronous processing), `org.springframework.web.client.AsyncRestTemplate` should be used instead of

`RestTemplate`. Refer *Asynchronous request* for implementation example of asynchronous processing.

`HttpMessageConverter`

`org.springframework.http.converter.HttpMessageConverter` is an interface which mutually converts Java object handled by application and message (JSON etc) for communicating with server.

When `RestTemplate` is used, implementation class of `HttpMessageConverter` below is registered by default.

Table.5.12: **HttpMessageConverter registered by default**

Sr. No.	Class name	Description	Support type
(1)	`org.springframework.http.converter.ByteArrayHttpMessageConverter`	A class for conversion of "HTTP body (text or binary data) ⇔ Byte array". It supports all media types (`*/*`) by default.	`byte[]`
(2)	`org.springframework.http.converter.StringHttpMessageConverter`	A class for conversion of "HTTP body (text) ⇔ String". It supports all text media types (`text/*`) by default.	`String`
(3)	`org.springframework.http.converter.ResourceHttpMessageConverter`	A class for conversion of "HTTP body (binary data) ⇔ Resource object of Spring". It supports all media types (`*/*`) by default.	`Resource` *1
(4)	`org.springframework.http.converter.xml.SourceHttpMessageConverter`	A class for conversion of "HTTP body (XML) ⇔ XML source object". It supports media types for XML (`text/xml`,`application/xml`,`application/*-xml`) by default.	`Source` *2

次のページに続く

Table. 5.12 – 前のページからの続き

Sr. No.	Class name	Description	Support type
(5)	`org.springframework.http.converter.support.AllEncompassingFormHttpMessageConverter`	A class for conversion of "HTTP body ⇔ `MultiValueMap` object". It is an extension class of `FormHttpMessageConverter` and supports conversion to XML and JSON as a multipart part data. It supports media types for form data (`application/x-www-form-urlencoded`,`multipart/form-data`) by default. • When media type is `application/x-www-form-urlencoded`, the data is read / written as `MultiValueMap<String,String>`. • When media type is `multipart/form-data`, data is written as `MultiValueMap<String,Object>` and `Object` is converted by `HttpMessageConveter` configured separately in `AllEncompassingFormHttpMessageConverter`. (Caution: Refer Note) Refer AllEncompassingFormHttpMessageConverter and FormHttpMessageConverter source for `HttpMessageConveter` used for conversion of part data which is registered by default. Note that, it is also possible to register an arbitrary `HttpMessageConverter`.	`MultiValueMap` [*3]

[*1] `org.springframework.core.io` package class
[*2] `javax.xml.transform` package class
[*3] `org.springframework.util` package class

Note: When media type of AllEncompassingFormHttpMessageConverter is multipart/form-data

When media type is `multipart/form-data`, conversion of "HTTP body from `MultiValueMap` object" can be done, however, conversion "from HTTP body to `MultiValueMap` object" is currently not supported. Hence, an independent implementation is required if conversion "from HTTP body to `MultiValueMap` object" is to be carried out.

Table.5.13: **HttpMessageConverter that is registered when a dependent library exists on the class path**

Sr. No.	Class Name	Description	Support type
(6)	`org.springframework.http.converter.feed.AtomFeedHttpMessageConverter`	A class for conversion of "HTTP body (Atom) ⇔ Atom feed object". It supports media type for ATOM (`application/atom+xml`) by default. (It is registered when ROME exists on the class path)	`Feed` [*4]
(7)	`org.springframework.http.converter.feed.RssChannelHttpMessageConverter`	A class for conversion of "HTTP body (RSS) ⇔ Rss channel object". It supports media type for RSS (`application/rss+xml`) by default. (It is registered when ROME exists on the class path)	`Channel` [*5]
(8)	`org.springframework.http.converter.json.MappingJackson2HttpMessageConverter`	A class for conversion of "HTTP body (JSON) ⇔ JavaBean". It supports media type for JSON (`application/json`,`application/*+json`) by default. (It is registered when Jackson2 exists on the class path)	`Object` (JavaBean) `Map`
(9)	`org.springframework.http.converter.xml.MappingJackson2XmlHttpMessageConverter`	A class for conversion of "HTTP body (XML) ⇔ JavaBean". It supports media type for XML (`text/xml`,`application/xml`,`application/*-xml`) by default. (It is registered when Jackson-dataformat-xml exists on the class path)	`Object` (JavaBean) `Map`

次のページに続く

Table. 5.13 – 前のページからの続き

Sr. No.	Class Name	Description	Support type
(10)	`org.springframework.http.converter.xml.Jaxb2RootElementHttpMessageConverter`	A class for conversion of "HTTP body (XML) ⇔ JavaBean". It supports media type for XML (`text/xml`,`application/xml`,`application/*-xml`) by default. (It is registered when JAXB exists on the class path)	`Object` (JavaBean)
(11)	`org.springframework.http.converter.json.GsonHttpMessageConverter`[*4]	A class for conversion of "HTTP body (JSON) ⇔ JavaBean". It supports media type for JSON (`application/json`,`application/*+json`) by default. (It is registered when Gson exists on the class path)	`Object` (JavaBean) `Map`

[*4] `com.rometools.rome.feed.atom` package class
[*5] `com.rometools.rome.feed.rss` package class

ClientHttpRequestFactory

RestTemplate delegates the process of communicating with the server to implementation class of three interfaces given below.

- org.springframework.http.client.ClientHttpRequestFactory
- org.springframework.http.client.ClientHttpRequest
- org.springframework.http.client.ClientHttpResponse

Of the 3 interfaces, the developers are aware of ClientHttpRequestFactory interface. ClientHttpRequestFactory resolves a class (implementation class of ClientHttpRequest and ClientHttpResponse interface) which communicates with the server.

Note that, the main implementation class of ClientHttpRequestFactory offered by Spring Framework is as given below.

Table.5.14 **Main implementation class of ClientHttpRequestFactory offered by Spring Framework**

Sr. No.	Class Name	Description
(1)	`org.springframework.http.client.SimpleClientHttpRequestFactory`	An implementation class for communication (synchronous, asynchronous) by using HttpURLConnection API of Java SE standard. (Implementation class used as a default)
(2)	`org.springframework.http.client.Netty4ClientHttpRequestFactory`	An implementation class for communication (synchronous, asynchronous) by using Netty 4 API.
(3)	`org.springframework.http.client.HttpComponentsClientHttpRequestFactory`	An implementation class for synchronous communication by using Apache HttpComponents HttpClient API. (HttpClient 4.3 and above versions are required)
(4)	`org.springframework.http.client.HttpComponentsAsyncClientHttpRequestFactory`	An implementation class for asynchronous communication by using Apache HttpComponents HttpAsyncClient API. (HttpAsyncClient 4.0 and above version are required)
(5)	`org.springframework.http.client.OkHttpClientHttpRequestFactory`	An implementation class for communication (synchronous, asynchronous) by using Square OkHttp API.

Note: Regarding implementation class of ClientHttpRequestFactory to be used

Default implementation used by `RestTemplate` is `SimpleClientHttpRequestFactory` and it can also act as an implementation example in this guideline while using `SimpleClientHttpRequestFactory`. If it does not meet requirements in `HttpURLConnection` of Java SE, using libraries like Netty, Apache Http

Components can be explored.

ResponseErrorHandler

`RestTemplate` handles the errors during the communication with the server by delegating to `org.springframework.web.client.ResponseErrorHandler` interface.

- A method to determine errors (`hasError`)
- A method to handle errors (`handleError`)

are defined in `ResponseErrorHandler`. Spring Framework offers `org.springframework.web.client.DefaultResponseErrorHandler` as a default implementation.

`DefaultResponseErrorHandler` carries out error handling as below according to values of HTTP status codes which have been sent as a response from the server.

- When response code is standard (2xx), error handling is not carried out.
- When response code is from client error system (4xx), `org.springframework.web.client.HttpClientErrorE` is generated.
- When response code is from server error system (5xx), `org.springframework.web.client.HttpServerErrorE` is generated.
- When response code is undefined (user defined custom code), `org.springframework.web.client.UnknownHttpStatusCodeException` is generated.

Note: How to fetch response data at the time of error

Response data at the time of error (HTTP status code, response header, response body etc) can be fetched by calling getter method of exception class.

ClientHttpRequestInterceptor

`org.springframework.http.client.ClientHttpRequestInterceptor` is an interface for implementing a common process before and after communicating with the server.

If `ClientHttpRequestInterceptor` is used, the common processes like

- Communication log with server
- Configuration of authentication header

can be applied in `RestTemplate`.

Note: Action specifications for ClientHttpRequestInterceptor

`ClientHttpRequestInterceptor` can be used for multiple times and is executed as a chain in a specified sequence. This operation is similar to working of a servlet filter and the HTTP communication process by `ClientHttpRequest` is registered as a chain destination executed at the end. For example, when you want to cancel communication with the server once it fulfils a certain condition, a chain destination need not be called.

When this system is used, processes like

- Blocking communication with server
- Retrying communication process

can also be applied.

5.2.2 How to use

This chapter explains how to implement a client process which uses `RestTemplate`.

Note: Regarding HTTP method supported by RestTemplate

In this guideline, only the implementation example of client process which use GET method and POST method is introduced, however, `RestTemplate` supports other HTTP methods (PUT, PATCH, DELETE, HEAD, OPTIONS etc) as well and can be used in the similar way. Refer Javadoc of RestTemplate for details.

`RestTemplate` Setup

When `RestTemplate` is used, `RestTemplate` is registered in DI container and injected in the component which uses `RestTemplate`.

Dependent library setup

spring-web library of Spring Framework is added to `pom.xml` for using `RestTemplate`.

In case of multi-project configuration, it is added to `pom.xml` of domain project.

```
<dependencies>

    <!-- (1) -->
```

```
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-web</artifactId>
    </dependency>

</dependencies>
```

Note: In the above setting example, since it is assumed that the dependent library version is managed by the parent project terasoluna-gfw-parent , specifying the version in pom.xml is not necessary. The above dependent library used by terasoluna-gfw-parent is defined by Spring IO Platform.

Sr. No.	Description
(1)	Add `spring-web` library of Spring Framework to dependencies.

Bean definition of `RestTemplate`

Define bean for `RestTemplate` and register in DI container.

Definition example of bean definition file (applicationContext.xml)

```
<bean id="restTemplate" class="org.springframework.web.client.RestTemplate" /> <!--
↪ (1) -->
```

Sr. No.	Description
(1)	When `RestTemplate` is used similar to default configuration, register a bean by using default constructor.

Note: How to customise RestTemplate

When HTTP communication process is to be customised, define a bean as below.

```
<bean id="clientHttpRequestFactory"
      class="org.springframework.http.client.SimpleClientHttpRequestFactory">
↪<!-- (1) -->
    <!-- Set properties for customize a http communication (omit on this␣
↪sample) -->
</bean>

<bean id="restTemplate" class="org.springframework.web.client.RestTemplate">
```

```
    <constructor-arg ref="clientHttpRequestFactory" /> <!-- (2) -->
</bean>
```

Sr. No.	Description
(1)	Define a bean for `ClientHttpRequestFactory`. A method to customise timeout configuration is introduced in this guideline. Refer *Setting communication timeout* for details.
(2)	Register a bean by using a constructor which specifies `ClientHttpRequestFactory` in the argument.

Also, refer

- *How to register an arbitrary HttpMessageConverter*
- *Returning ResponseEntity (Error handler extension)*
- *Application of common process (ClientHttpRequestInterceptor)*

for how to customise `HttpMessageConverter`, `ResponseErrorHandler` and `ClientHttpRequestInterceptor`.

Using `RestTemplate`

When `RestTemplate` is used, `RestTemplate` registered in DI container is injected.

Injection example for RestTemplate

```
@Service
public class AccountServiceImpl implements AccountService {

    @Inject
    RestTemplate restTemplate;

    // ...

}
```

Sending GET request

`RestTemplate` offers multiple methods to send a GET request.

- Usually, `getForObject` method or `getForEntity` method are used.
- When a detailed setting such as setting a header is to be carried out, `org.springframework.http.RequestEntity` and `exchange` methods are used.

Implementation by using `getForObject` method

When only the response body is required to be fetched, `getForObject` method is used.

How to use getForObject method

Field declaration part

```
@Value("${api.url:http://localhost:8080/api}")
URI uri;
```

Internal method

```
User user = restTemplate.getForObject(uri, User.class); // (1)
```

Sr. No.	Description
(1)	When `getForObject` method is used, the response body value is sent as a return value. Response body data is returned after it has been converted to Java class specified in the second argument, by using `HttpMessageConverter`.

Implementation by using `getForEntity` method

When HTTP status code, response header and response body must be fetched, `getForEntity` method is used.

How to use getForEntity method

```
ResponseEntity<User> responseEntity =
        restTemplate.getForEntity(uri, User.class); // (1)
HttpStatus statusCode = responseEntity.getStatusCode(); // (2)
HttpHeaders header = responseEntity.getHeaders(); // (3)
User user = responseEntity.getBody(); // (4)
```

Sr. No.	Description
(1)	When `getForEntity` method is used, `org.springframework.http.ResponseEntity` is sent as a return value.
(2)	Fetch HTTP status code by using `getStatusCode` method.
(3)	Fetch response header by using `getHeaders` method.
(4)	Fetch response body by using `getBody` method.

Note: ResponseEntity

`ResponseEntity` is a class which shows HTTP response and can fetch HTTP status code, response header and response body information. Refer Javadoc of ResponseEntity for details.

Implementation by using `exchange` method

When a request header must be specified, `org.springframework.http.RequestEntity` is generated and `exchange` method is used.

How to use exchange method

import part

```
import org.springframework.http.RequestEntity;
import org.springframework.http.ResponseEntity;
```

Field declaration part

```
@Value("${api.url:http://localhost:8080/api}")
URI uri;
```

Internal method

```
RequestEntity requestEntity = RequestEntity
        .get(uri)//(1)
        .build();//(2)
```

```
ResponseEntity<User> responseEntity =
        restTemplate.exchange(requestEntity, User.class);//(3)

User user = responseEntity.getBody();//(4)
```

Sr. No.	Description
(1)	Use `get` method of `RequestEntity` and generate request builder for GET request. Specify URI in the parameter.
(2)	Use `build` method of `RequestEntity.HeadersBuilder` and create `RequestEntity` object.
(3)	Use `exchange` method and send request. Specify response data type in the second argument. `ResponseEntity<T>` is sent as a response. Specify response data type in Type parameter.
(4)	Use `getBody` method and fetch response body data.

Note: RequestEntity

`RequestEntity` is a class which shows HTTP request and can set connection URI, HTTP method, request header and request body. Refer Javadoc of RequestEntity for details.

Also, refer *Configuration of request header* for how to configure a request header.

Sending POST request

`RestTemplate` offers multiple methods for carrying out POST request.

- Usually, `postForObject` and `postForEntity` are used.
- When a detailed setting like setting any header is to be carried out, `RequestEntity` and `exchange` methods are used.

Implementation by using `postForObject` method

When only response body is required to be fetched as POST results, `postForObject` method is used.

How to use postForObject method

```
User user = new User();

//...

User user = restTemplate.postForObject(uri, user, User.class); // (1)
```

Sr. No.	Description
(1)	`postForObject` method can easily implement a POST request. Specify Java object in the second argument which is converted to request body by using `HttpMessageConverter`. When `postForObject` method is used, response body value is sent as a return value.

Implementation using `postForEntity` method

When HTTP status code, response header and response body are to be fetched as POST results, `postForEntity` method is used.

How to use postForEntity method

```
User user = new User();

//...

ResponseEntity<User> responseEntity =
        restTemplate.postForEntity(uri, user, User.class); // (1)
```

Sr. No.	Description
(1)	`postForEntity` method can implement a POST request easily similar to `getForObject` method. When `postForEntity` method is used, `ResponseEntity` is sent as a return value. Fetch response body value from `ResponseEntity`.

Implementation using `exchange` method

When a request header is to be specified, `RequestEntity` is generated and `exchange` method is used.

How to use exchange method

import part

```
import org.springframework.http.RequestEntity;
import org.springframework.http.ResponseEntity;
```

Field declaration part

```
@Value("${api.url:http://localhost:8080/api}")
URI uri;
```

Internal method

```
User user = new User();

//...

RequestEntity<User> requestEntity = RequestEntity//(1)
        .post(uri)//(2)
        .body(user);//(3)

ResponseEntity<User> responseEntity =
        restTemplate.exchange(requestEntity, User.class);//(4)
```

Sr. No.	Description
(1)	Use `RequestEntity` and generate a request. Specify type of the data specified in the request body, in Type parameter.
(2)	Use `post` method and generate a request builder for POST request. Specify URI in the parameter.
(3)	Use `body` method of `RequestEntity.BodyBuilder` and create `RequestEntity` object. Specify Java object that has been converted to request body, in the parameter.
(4)	Use `exchange` method and send a request.

Note: How to configure a request header

Refer *Configuration of request header* for how to configure a request header.

Fetch data in collection format

When the message of response body received from server as a response is in collection format, the implementation is as below.

How to fetch data in collection format

```
ResponseEntity<List<User>> responseEntity = //(1)
    restTemplate.exchange(requestEntity, new ParameterizedTypeReference<List<User>>
↪(){}); //(2)

List<User> userList = responseEntity.getBody();//(3)
```

Sr. No.	Description
(1)	Specify `List`<Response data type> in `ResponseEntity` Type parameter.
(2)	Specify instance of `org.springframework.core.ParameterizedTypeReference` in the second argument of `exchange` method, and specify `List`<Response data type> in Type parameter.
(2)	Fetch response body data by `getBody` method.

Configuration of request header

If `RequestEntity` and `exchange` methods are used, a specific header or any other header can be set by using `RequestEntity` method. Refer Javadoc of RequestEntity for details.

This guideline explains about

- *Configuration of Content-Type header*
- *Configuration of Accept header*
- *Configuration of an arbitrary request header*

Configuration of Content-Type header

While sending data to server, a usual Content-Type header must be specified.

How to configure Content-Type header

```
User user = new User();

//...

RequestEntity<User> requestEntity = RequestEntity
        .post(uri)
        .contentType(MediaType.APPLICATION_JSON) // (1)
        .body(user);
```

Sr. No.	Description
(1)	Use `contentType` method of `RequestEntity.BodyBuilder` and specify value of Context-Type header. In the implementation example above, "`application/json`" is specified which indicates that the data format is JSON.

Configuration of Accept header

When the format of data to be fetched from server is specified, Accept header must be specified. When the server does not support multiple data format responses, Accept header may not be specified explicitly.

Configuration example of Accept header

```
User user = new User();

//...

RequestEntity<User> requestEntity = RequestEntity
        .post(uri)
        .accept(MediaType.APPLICATION_JSON) // (1)
        .body(user);
```

Sr. No.	Description
(1)	Use `accept` method of `RequestEntity.HeadersBuilder` and specify value of Accept header. In the implementation example above, "`application/json`" is specified which indicates that format of the data that can be fetched is JSON format.

Configuration of an arbitrary request header

A request header must be specified to access server.

Configuration example for an arbitrary header

```
User user = new User();

//...

RequestEntity<User> requestEntity = RequestEntity
        .post(uri)
        .header("Authorization", "Basic " + base64Credentials) // (1)
        .body(user);
```

Sr. No.	Description
(1)	Use `header` method of `RequestEntity.HeadersBuilder` and specify name and value of request header. In the implementation example above, credentials information necessary for Basic authentication is specified in Authorization header.

Error Handling

Exception Handling (Default Behaviour)

Exceptions like

- `HttpClientErrorException` when response code is of client error system (4xx)
- `HttpServerErrorException` when response code is of server error system (5xx)
- `UnknownHttpStatusCodeException` when response code is a undefined code (user defined custom code)

occur in default implementation (`DefaultResponseErrorHandler`) of `RestTemplate`, these exceptions must be handled as and when necessary.

Implementation example of exception handling

Note: An example of exception handling when a server error has occurred is shown below as an implementation example.

Appropriate exception handling must be carried out as per requirements of an application.

Field declaration part

```
@Value("${api.retry.maxCount}")
int retryMaxCount;

@Value("${api.retry.retryWaitTimeCoefficient}")
int retryWaitTimeCoefficient;
```

Internal method

```
int retryCount = 0;
while (true) {
    try {

        responseEntity = restTemplate.exchange(requestEntity, String.class);

        if (log.isInfoEnabled()) {
            log.info("Success({}) ", responseEntity.getStatusCode());
        }

        break;

    } catch (HttpServerErrorException e) { // (1)

        if (retryCount == retryMaxCount) {
            throw e;
        }

        retryCount++;

        if (log.isWarnEnabled()) {
            log.warn("An error ({}) occurred on the server. (The number of retries:
↪{} Times)", e.getStatusCode(),
                retryCount);
        }

        try {
            Thread.sleep(retryWaitTimeCoefficient * retryCount);
```

```
        } catch (InterruptedException ie) {
            Thread.currentThread().interrupt();
        }

        //...
    }

}
```

Sr. No.	Description
(1)	Catch exception and perform error handling. In case of a server error (500 system), catch `HttpServerErrorException`.

Returning `ResponseEntity` (Error handler extension)

By setting implementation class of `org.springframework.web.client.ResponseErrorHandler` interface in `RestTemplate`, an independent error handling process can be carried out.

In the example below, the error handler is extended so as to return `ResponseEntity` even when a server error and a client error has occurred.

How to create an implementation class of error handler

```
import org.springframework.http.client.ClientHttpResponse;
import org.springframework.web.client.DefaultResponseErrorHandler;

public class CustomErrorHandler extends DefaultResponseErrorHandler { // (1)

    @Override
    public void handleError(ClientHttpResponse response) throws IOException {
        //Don't throw Exception.
    }

}
```

Sr. No.	Description
(1)	Create implementation class of `ResponseErrorHandler` interface. In the implementation example above, `DefaultResponseErrorHandler` - an implementation class of default error handler is extended and `ResponseEntity` is returned without generating an exception when a server error and client error has occurred.

Implementation example of bean definition file (applicationContext.xml)

```
<bean id="customErrorHandler" class="com.example.restclient.CustomErrorHandler" />
↪<!-- (1) -->

<bean id="restTemplate" class="org.springframework.web.client.RestTemplate">
    <property name="errorHandler" ref="customErrorHandler" /><!-- (2) -->
</bean>
```

Sr. No.	Description
(1)	Define bean for implementation class of `ResponseErrorHandler`.
(2)	Inject `ResponseErrorHandler` bean in `errorHandler` property.

Implementation example of client process

```
int retryCount = 0;
while (true) {

    responseEntity = restTemplate.exchange(requestEntity, User.class);

    if (responseEntity.getStatusCode() == HttpStatus.OK) { // (1)

        break;

    } else if (responseEntity.getStatusCode() == HttpStatus.SERVICE_UNAVAILABLE) {␣
↪// (2)

        if (retryCount == retryMaxCount) {
            break;
        }

        retryCount++;

        if (log.isWarnEnabled()) {
            log.warn("An error ({}) occurred on the server. (The number of retries:
↪{} Times)",
                    responseEntity.getStatusCode(), retryCount);
        }

        try {
            Thread.sleep(retryWaitTimeCoefficient * retryCount);
        } catch (InterruptedException ie) {
            Thread.currentThread().interrupt();
        }
```

```
        //...
    }
}
```

Sr. No.	Description
(1)	In the implementation example above, since error handler is extended so as to return `ResponseEntity` even at the time of error, it is necessary to check whether process results are normal after fetching HTTP status code from `ResponseEntity` thus returned.
(2)	HTTP status code can be fetched from returned `ResponseEntity` even at the time of error and process can be controlled corresponding to that value.

Setting communication timeout

When a timeout period is to be specified for communicating with server, define a bean as given below.

Implementation example of bean definition file (applicationContext.xml)

```
<bean id="clientHttpRequestFactory"
      class="org.springframework.http.client.SimpleClientHttpRequestFactory">
    <property name="connectTimeout" value="${api.connectTimeout: 2000}" /><!-- (1)␣
↪-->
    <property name="readTimeout" value="${api.readTimeout: 2000}" /><!-- (2) -->
</bean>

<bean id="restTemplate" class="org.springframework.web.client.RestTemplate">
    <constructor-arg ref="clientHttpRequestFactory" />
</bean>
```

Sr. No.	Description
(1)	Specify connection timeout (milliseconds) with server in `connectTimeout` property. When timeout occurs, `org.springframework.web.client.ResourceAccessException` is generated.
(2)	Specify response data read timeout (milliseconds) in `readTimeout` property. When timeout occurs, `ResourceAccessException` is generated.

Note: Cause Exception during Timeout Occurrence

`ResourceAccessException` wraps the cause exception. Cause exception during connection timeout and read timeout occurrence is `java.net.SocketTimeoutException` for both. When default implementation (`SimpleClientHttpRequestFactory`) is used, it must be added that type of timeout occurrence cannot be distinguished by the type of exception class.

Note that, since operation while using `HttpRequestFactory` is not verified, cause exception is likely to be different from the one described above. When other `HttpRequestFactory` is used, appropriate exception handling must be employed after assessing the exception occurred during the timeout.

Using SSL self-signed certificate

Implementation is as given below when a SSL self-signed certificate is to be used in the test environment.

Implementation example of FactoryBean

Implement `org.springframework.beans.factory.FactoryBean` to create `org.springframework.http.client.ClientHttpRequestFactory` to be passed in constructor argument, in Bean definition of `RestTemplate`.

```
import java.security.KeyStore;

import javax.net.ssl.KeyManagerFactory;
import javax.net.ssl.SSLContext;
import javax.net.ssl.TrustManagerFactory;

import org.apache.http.client.HttpClient;
import org.apache.http.impl.client.HttpClientBuilder;
import org.springframework.beans.factory.FactoryBean;
import org.springframework.http.client.ClientHttpRequestFactory;
import org.springframework.http.client.HttpComponentsClientHttpRequestFactory;

public class RequestFactoryBean implements
        FactoryBean<ClientHttpRequestFactory> {

    private String keyStoreFileName;

    private char[] keyStorePassword;

    @Override
    public ClientHttpRequestFactory getObject() throws Exception {

        // (1)
        SSLContext sslContext = SSLContext.getInstance("TLS");
```

```
        KeyStore ks = KeyStore.getInstance(KeyStore.getDefaultType());
        ks.load(this.getClass().getClassLoader()
                .getResourceAsStream(this.keyStoreFileName),
                this.keyStorePassword);

        KeyManagerFactory kmf = KeyManagerFactory.getInstance(KeyManagerFactory
                .getDefaultAlgorithm());
        kmf.init(ks, this.keyStorePassword);

        TrustManagerFactory tmf = TrustManagerFactory
                .getInstance(TrustManagerFactory.getDefaultAlgorithm());
        tmf.init(ks);

        sslContext.init(kmf.getKeyManagers(), tmf.getTrustManagers(), null);

        // (2)
        HttpClient httpClient = HttpClientBuilder.create()
                .setSSLContext(sslContext).build();

        // (3)
        ClientHttpRequestFactory factory = new␣
↪HttpComponentsClientHttpRequestFactory(
                httpClient);

        return factory;
    }

    @Override
    public Class<?> getObjectType() {
        return ClientHttpRequestFactory.class;
    }

    @Override
    public boolean isSingleton() {
        return true;
    }

    public void setKeyStoreFileName(String keyStoreFileName) {
        this.keyStoreFileName = keyStoreFileName;
    }

    public void setKeyStorePassword(char[] keyStorePassword) {
        this.keyStorePassword = keyStorePassword;
    }

}
```

Sr. No.	Description
(1)	Create SSL context based on file name and password of keystore file which is specified in subsequent bean definition. Keystore file of SSL self-signed certificate to be used is placed on the class path.
(2)	Create `org.apache.http.client.HttpClient` which uses SSL context thus created.
(3)	Create `ClientHttpRequestFactory` which uses `HttpClient` thus created.

Apache HttpComponents HttpClient library is required in order to use of `HttpClient` and `HttpClientBuilder`. Add below Apache HttpComponents HttpClient dependency library into `pom.xml`.

- `pom.xml`

```
<dependency>
    <groupId>org.apache.httpcomponents</groupId>
    <artifactId>httpclient</artifactId>
</dependency>
```

Note: In the above setting example, since it is assumed that the dependent library version is managed by the parent project terasoluna-gfw-parent , specifying the version in pom.xml is not necessary. The above dependent library used by terasoluna-gfw-parent is defined by Spring IO Platform.

Implementation example of bean definition file (applicationContext.xml)

Define `RestTemplate`which carries out SSL communication using SSL self-signed certificate.

```
<bean id="httpsRestTemplate" class="org.springframework.web.client.RestTemplate">
    <constructor-arg>
        <bean class="com.example.restclient.RequestFactoryBean"><!-- (1) -->
            <property name="keyStoreFileName" value="${rscl.keystore.filename}" />
            <property name="keyStorePassword" value="${rscl.keystore.password}" />
        </bean>
    </constructor-arg>
</bean>
```

Sr. No.	Description
(1)	Specify created `RequestFactoryBean` in `RestTemplate` constructor. Pass file name and password of keystore file in `RequestFactoryBean`.

How to use RestTemplate

The method to use `RestTemplate` is same as the method when SSL self-signed certificate is not used.

Basic authentication

Implementation is as below when a server requests a basic authentication.

Implementation example of Basic authentication

Field declaration part

```
@Value("${api.auth.userid}")
String userid;

@Value("${api.auth.password}")
String password;
```

Internal method

```
String plainCredentials = userid + ":" + password; // (1)
String base64Credentials = Base64.getEncoder()
        .encodeToString(plainCredentials.getBytes(StandardCharsets.UTF_8)); // (2)

RequestEntity requestEntity = RequestEntity
      .get(uri)
      .header("Authorization", "Basic " + base64Credentials) // (3)
      .build();
```

Sr. No.	Description
(1)	Connect user ID and password with "**:**".
(2)	Convert (1) to byte array and perform Base64 encoding.
(3)	Authorization header specifies credentials information of Basic authentication.

Note: `java.util.Base64` of Java standard is used for Java SE8 and later versions. Earlier, `org.springframework.security.crypto.codec.Base64`of Spring Security is used.

File upload (multi-part request)

Implementation is as below when file is to be uploaded (multi-part request) using `RestTemplate`.

Implementation example for file upload

```
MultiValueMap<String, Object> multiPartBody = new LinkedMultiValueMap<>();//(1)
multiPartBody.add("file", new ClassPathResource("/uploadFiles/User.txt"));//(2)

RequestEntity<MultiValueMap<String, Object>> requestEntity = RequestEntity
        .post(uri)
        .contentType(MediaType.MULTIPART_FORM_DATA)//(3)
        .body(multiPartBody);//(4)
```

Sr. No.	Description
(1)	Generate `MultiValueMap` for storing data sent as a multi-part request.
(2)	Specify parameter name in key and add file to be uploaded in `MultiValueMap`. In the example above, file placed on the class path is added as an uploaded file by specifying parameter name as `file`.
(3)	Specify media type of Content-Type header in `multipart/form-data`.
(4)	Specify `MultiValueMap` in the request body wherein the uploaded file has been stored.

Note: Regarding Resource class offered by Spring Framework

Spring Framework offers `org.springframework.core.io.Resource` as an interface which represents the resource and can be used while uploading a file.

Main implementation classes of `Resource` interface are as below.

- `org.springframework.core.io.PathResource`

- `org.springframework.core.io.FileSystemResource`
- `org.springframework.core.io.ClassPathResource`
- `org.springframework.core.io.UrlResource`
- `org.springframework.core.io.InputStreamResource` (file name cannot be linked to server)
- `org.springframework.core.io.ByteArrayResource` (file name cannot be linked to server)

File download

Implementation is as below when file is to be downloaded using `RestTeamplate`.

Implementation example of file download (when file size is small)

```
RequestEntity requestEntity = RequestEntity
        .get(uri)
        .build();

ResponseEntity<byte[]> responseEntity =
        restTemplate.exchange(requestEntity, byte[].class);//(1)

byte[] downloadContent = responseEntity.getBody();//(2)
```

Sr. No.	Description
(1)	Handle downloaded file with a specified data type. Here, byte array is specified.
(2)	Fetch data of downloaded file from response body.

Warning: Precautions to be taken while downloading a large file

If a large file is fetched in `byte` array using `HttpMessageConverter` registered as default, `java.lang.OutOfMemoryError` is likely to occur. Hence, when a large file is to be downloaded, it is necessary to write downloaded data to the file in parts by fetching `InputStream` from response.

Implementation example of file download (when file size is large)

```
// (1)
final ResponseExtractor<ResponseEntity<File>> responseExtractor =
```

```
        new ResponseExtractor<ResponseEntity<File>>() {

    // (2)
    @Override
    public ResponseEntity<File> extractData(ClientHttpResponse response)
            throws IOException {

        File rcvFile = File.createTempFile("rcvFile", "zip");

        FileCopyUtils.copy(response.getBody(), new FileOutputStream(rcvFile));

        return ResponseEntity.status(response.getStatusCode())
                .headers(response.getHeaders()).body(rcvFile);
    }

};

// (3)
ResponseEntity<File> responseEntity = this.restTemplate.execute(targetUri,
        HttpMethod.GET, null, responseExtractor);
if (HttpStatus.OK.equals(responseEntity.getStatusCode())) {
    File getFile = responseEntity.getBody();

    .....

}
```

Sr. No.	Description
(1)	Create a process to create a return value of `RestTemplate#execute`,from the response fetched from `RestTemplate#execute`.
(2)	Read data from response body (`InputStream`) and create a file. Created file, HTTP header and status code are stored in `ResponseEntity<File>` and returned.
(3)	Download file using `RestTemplate#execute`.

Implementation example of file download (when file size is large (example wherein ResponseEntity is not used))

When status code determination and HTTP header reference are not required, `File` should be returned instead of `ResponseEntity` as given below.

```
final ResponseExtractor<File> responseExtractor = new ResponseExtractor<File>() {

    @Override
    public File extractData(ClientHttpResponse response)
            throws IOException {

        File rcvFile = File.createTempFile("rcvFile", "zip");

        FileCopyUtils.copy(response.getBody(), new FileOutputStream(
                rcvFile));

        return rcvFile;
    }

};

File getFile = this.restTemplate.execute(targetUri, HttpMethod.GET,
        null, responseExtractor);
.....
```

How to handle RESTful URL (URI template) and implementation example

Implementation can be carried out by using URI template for handling RESTful URL.

How to use getForObject method

Field declaration part

```
@Value("${api.serverUrl}/api/users/{userId}") // (1)
String uriStr;
```

Internal method

```
User user = restTemplate.getForObject(uriStr, User.class, "0001"); // (2)
```

Sr. No.	Description
(1)	Variable {userId} of URI template is changed to value specified while using `RestTeamplate`.
(2)	One variable of URI template is replaced with a value specified in third argument of `getForObject` method and processed as "http://localhost:8080/api/users/0001".

How to use exchange method

```
@Value("${api.serverUrl}/api/users/{action}") // (1)
String uriStr;
```

Internal method

```
URI targetUri = UriComponentsBuilder.fromUriString(uriStr).
        buildAndExpand("create").toUri(); //(2)

User user = new User();

//...

RequestEntity<User> requestEntity = RequestEntity
        .post(targetUri)
        .body(user);

ResponseEntity<User> responseEntity = restTemplate.exchange(requestEntity, User.
↪class);
```

Sr. No.	Description
(1)	Variable {action} of URI template is changed to value specified while using `RestTeamplate`.
(2)	By using `UriComponentsBuilder`, first variable of URI template is replaced by value specified in the argument of `buildAndExpand` and "http://localhost:8080/api/users/create" URI is created. Refer Javadoc of UriComponentsBuilder for details.

5.2.3 How to extend

This chapter explains how to extend `RestTemplate`.

How to register an arbitrary `HttpMessageConverter`

If the requirements of message conversion are not met by `HttpMessageConverter` registered as default, an arbitrary `HttpMessageConverter` can be registered. However, since `HttpMessageConverter` registered as default is deleted, the required `HttpMessageConverter` should all be individually registered.

How to define a bean definition file (applicationContext.xml)

```
<bean id="jaxb2CollectionHttpMessageConverter"
      class="org.springframework.http.converter.xml.
↪Jaxb2CollectionHttpMessageConverter" /> <!-- (1) -->
```

```
<bean id="restTemplate" class="org.springframework.web.client.RestTemplate">
    <property name="messageConverters"> <!-- (2) -->
        <list>
            <ref bean="jaxb2CollectionHttpMessageConverter" />
        </list>
    </property>
</bean>
```

Sr. No.	Description
(1)	Define a bean for implementation class of `HttpMessageConverter` to be registered.
(2)	Inject `HttpMessageConverter` bean registered in `messageConverters`property.

Application of common process (`ClientHttpRequestInterceptor`)

By using `ClientHttpRequestInterceptor`, a process can be executed before and after communicating with the server.

Here, implementation example for

- *Logging process*
- *Process to configure a request header for Basic authentication*

is introduced below.

Logging process

When a log for communication with server is to be output, implementation is as below.

Implementation example of communication log output

```
package com.example.restclient;

import org.springframework.http.HttpRequest;
import org.springframework.http.client.ClientHttpRequestExecution;
import org.springframework.http.client.ClientHttpRequestInterceptor;
import org.springframework.http.client.ClientHttpResponse;

public class LoggingInterceptor implements ClientHttpRequestInterceptor { //(1)

    private static final Logger log = LoggerFactory.getLogger(LoggingInterceptor.
↪class);
```

```
    @Override
    public ClientHttpResponse intercept(HttpRequest request, byte[] body,
            ClientHttpRequestExecution execution) throws IOException {

        if (log.isInfoEnabled()) {
            String requestBody = new String(body, StandardCharsets.UTF_8);

            log.info("Request Header {}", request.getHeaders()); //(2)
            log.info("Request Body {}", requestBody);
        }

        ClientHttpResponse response = execution.execute(request, body); //(3)

        if (log.isInfoEnabled()) {
            log.info("Response Header {}", response.getHeaders()); // (4)
            log.info("Response Status Code {}", response.getStatusCode()); // (5)
        }

        return response; // (6)
    }

}
```

Sr. No.	Description
(1)	Implement `ClientHttpRequestInterceptor` interface.
(2)	Implement a common process to be carried out prior to sending a request. In the implementation example above, details of request header and request body are output in a log.
(3)	Run `execute` method of `ClientHttpRequestExecution` received as an argument for `intercept` method and send a request.
(4)	Implement a common process which is to be carried out after receiving a response. In the implementation example above, response header details are output in a log.
(5)	Similar to (4), status code details are output in a log.
(6)	Return the response received in (3).

Process to configure a request header for Basic authentication

When it is necessary to configure a request header for Basic authentication to access the server, implementation is as given below.

Implementation example of request header configuration process for Basic authentication

```
package com.example.restclient;

import org.springframework.http.HttpRequest;
import org.springframework.http.client.ClientHttpRequestExecution;
import org.springframework.http.client.ClientHttpRequestInterceptor;
import org.springframework.http.client.ClientHttpResponse;

public class BasicAuthInterceptor implements ClientHttpRequestInterceptor { //(1)

    private static final Logger log = LoggerFactory.getLogger(BasicAuthInterceptor.
↪class);
```

```
    @Value("${api.auth.userid}")
    String userid;

    @Value("${api.auth.password}")
    String password;

    @Override
    public ClientHttpResponse intercept(HttpRequest request, byte[] body,
            ClientHttpRequestExecution execution) throws IOException {

        String plainCredentials = userid + ":" + password;
        String base64Credentials = Base64.getEncoder()
                .encodeToString(plainCredentials.getBytes(StandardCharsets.UTF_8));
        request.getHeaders().add("Authorization", "Basic " + base64Credentials); //
 (1)

        ClientHttpResponse response = execution.execute(request, body);

        return response;
    }

}
```

Sr. No.	Description
(1)	Add a request header for Basic authentication in `intercept` method.

Applying `ClientHttpRequestInterceptor`

When `ClientHttpRequestInterceptor` created in `RestTemplate` is to be applied, define a bean as given below.

How to define a bean definition file (applicationContext.xml)

```
<!-- (1) -->
<bean id="basicAuthInterceptor" class="com.example.restclient.BasicAuthInterceptor
 ↪" />
<bean id="loggingInterceptor" class="com.example.restclient.LoggingInterceptor" />

<bean id="restTemplate" class="org.springframework.web.client.RestTemplate">
    <property name="interceptors"><!-- (2) -->
        <list>
            <ref bean="basicAuthInterceptor" />
            <ref bean="loggingInterceptor" />
        </list>
    </property>
```

```
</bean>
```

Sr. No.	Description
(1)	Define a bean for implementation class of `ClientHttpRequestInterceptor`.
(2)	Inject `ClientHttpRequestInterceptor` bean in `interceptors` property. When multiple beans are to be injected, execute the process in a chain sequence starting from top of the list. In the example above, processes prior to request are implemented in the sequence - `BasicAuthInterceptor` -> `LoggingInterceptor` -> `ClientHttpRequest`. (the sequence will be reversed for the processes after receiving a response)

Asynchronous request

When an asynchronous request is to be carried out, `org.springframework.web.client.AsyncRestTemplate` is used.

Bean definition for `AsyncRestTemplate`

Define a bean for `AsyncRestTemplate`.

How to define a bean definition file (applicationContext.xml)

```
<bean id="asyncRestTemplate" class="org.springframework.web.client.
↪AsyncRestTemplate" /> <!-- (1) -->
```

Sr. No.	Description
(1)	When `AsyncRestTemplate` is to be used as per default setup, register a bean by using a default constructor. In case of default configuration, `SimpleClientHttpRequestFactory` which has set `org.springframework.core.task.SimpleAsyncTaskExecutor` is set as `org.springframework.core.task.AsyncListenableTaskExecutor` in `org.springframework.http.client.AsyncClientHttpRequestFactory` of `AsyncRestTemplate`.

Note: How to customise AsyncRestTemplate

`SimpleAsyncTaskExecutor` set as default generates threads without using a thread pool and there is no restriction on number of concurrent execution of threads. Hence, when the number of threads to be used concurrently is very large, OutOfMemoryError is likely to occur.

By setting a Bean of `org.springframework.core.task.AsyncListenableTaskExecutor` interface, in the constructor of `AsyncRestTemplate`, upper limit for thread pool count can be specified. An example of setting `org.springframework.scheduling.concurrent.ThreadPoolTaskExecutor`is given below.

```
<!-- (1) -->
<bean id="asyncTaskExecutor" class="org.springframework.scheduling.concurrent.
↪ThreadPoolTaskExecutor">
    <property name="maxPoolSize" value="100" />
</bean>

<!-- (2) -->
<bean id="asyncRestTemplate" class="org.springframework.web.client.
↪AsyncRestTemplate" >
    <constructor-arg index="0" ref="asyncTaskExecutor" />
</bean>
```

Sr. No.	Description
(1)	Define a bean for `AsyncTaskExecutor`. Thread operation using a thread pool is carried out by using `ThreadPoolTaskExecutor`. Further, number of threads can be controlled by setting `maxPoolSize` property.
(2)	Define a bean for `AsyncRestTemplate`. Register a bean by using a constructor which specifies `ThreadPoolTaskExecutor`in the argument.

This guideline introduces an implementation example to customise the task execution process only, however HTTP communication process can also be customised for `AsyncRestTemplate`. Refer Javadoc of AsyncRestTemplate for details.

Also, customisation for other than thread pool size is possible for `ThreadPoolTaskExecutor`as well. Refer Javadoc of ThreadPoolTaskExecutor for details.

Implementation of asynchronous request

Implementation example of asynchronous request

Field declaration part

```
@Inject
AsyncRestTemplate asyncRestTemplate;
```

Internal method

```
ListenableFuture<ResponseEntity<User>> responseEntity =
        asyncRestTemplate.getForEntity(uri, User.class); // (1)

responseEntity.addCallback(new ListenableFutureCallback<ResponseEntity<User>>() { //
↪ (2)
    @Override
    public void onSuccess(ResponseEntity<User> entity) {
        //...
    }

    @Override
    public void onFailure(Throwable t) {
      //...
    }
});
```

Sr. No.	Description
(1)	Send asynchronous request by using each method of `AsyncRestTemplate`. In the implementation example above, `getForEntity` method is used. `ResponseEntity` wrapped in `org.springframework.util.concurrent.ListenableFuture`is sent as return value. How to use each method is similar to `RestTemplate`.
(2)	Register `org.springframework.util.concurrent.ListenableFutureCallback` in `ListenableFuture` and implement a process when a response has returned. Implement the process in `onSuccess` method when a successful response has returned and implement a process in `onFailure`when an error has occurred.

Implementation of common processing for asynchronous request

Any arbitrary process can be executed before and after communicating with the server by using `org.springframework.http.client.AsyncClientHttpRequestInterceptor`.

An example of login process is introduced here.

Implementation example of communication log output

```
package com.example.restclient;

import java.io.IOException;
import java.nio.charset.StandardCharsets;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.http.HttpRequest;
import org.springframework.http.client.AsyncClientHttpRequestExecution;
import org.springframework.http.client.AsyncClientHttpRequestInterceptor;
import org.springframework.http.client.ClientHttpResponse;
import org.springframework.util.concurrent.ListenableFuture;
import org.springframework.util.concurrent.ListenableFutureCallback;

public class AsyncLoggingInterceptor implements
                                    AsyncClientHttpRequestInterceptor { // (1)
    private static final Logger log = LoggerFactory.getLogger(
            AsyncLoggingInterceptor.class);

    @Override
    public ListenableFuture<ClientHttpResponse> intercept(HttpRequest request,
            byte[] body,
            AsyncClientHttpRequestExecution execution) throws IOException {
        // (2)
        if (log.isInfoEnabled()) {
            String requestBody = new String(body, StandardCharsets.UTF_8);

            log.info("Request Header {}", request.getHeaders());
            log.info("Request Body {}", requestBody);
        }

        // (3)
        ListenableFuture<ClientHttpResponse> future = execution.executeAsync(
                request, body);
        if (log.isInfoEnabled()) {
            // (4)
            future.addCallback(new ListenableFutureCallback<ClientHttpResponse>() {

                @Override
                public void onSuccess(ClientHttpResponse response) {
                    try {
                        log.info("Response Header {}", response
                                .getHeaders());
                        log.info("Response Status Code {}", response
                                .getStatusCode());
                    } catch (IOException e) {
                        log.warn("I/O Error", e);
                    }
                }
```

```
                @Override
                public void onFailure(Throwable e) {
                    log.info("Communication Error", e);
                }
            });
        }

        return future; // (5)
    }
}
```

Sr. No.	Description
(1)	Implement `AsyncClientHttpRequestInterceptor` interface.
(2)	Implement a process which is executed prior to sending an asynchronous request. In the implementation example above, contents of request header and request body are output in the log.
(3)	Send an asynchronous request by using `executeAsync` method of `AsyncClientHttpRequestExecution` which is received as an argument of `intercept` method.
(4)	Register `org.springframework.util.concurrent.ListenableFutureCallback` in `ListenableFuture` which is received in (3) and implement the process once the response is received. When the response is received, `onSuccess` method is called. Also, when an exception occurs at the time of asynchronous request, `onFailure` method is called. A specific example is shown below. • Unable to connect a specified host （`ConnectException`） • Timeout occurred for reading of response data （`SocketTimeoutException`）
(5)	Return `ListenableFuture` returned in (3).

Definition example of a bean definition file (applicationContext.xml)

```
<!-- (1) -->
<bean id="asyncLoggingInterceptor" class="com.example.restclient.
↪AsyncLoggingInterceptor" />

<bean id="asyncRestTemplate" class="org.springframework.web.client.
↪AsyncRestTemplate">
    <property name="interceptors"><!-- (2) -->
        <list>
            <ref bean="asyncLoggingInterceptor" />
        </list>
    </property>
</bean>
```

Sr. No.	Description
(1)	Define a bean for implementation class of `AsyncClientHttpRequestInterceptor`.
(2)	Inject a bean of `AsyncClientHttpRequestInterceptor` in `interceptors` property. When multiple beans are injected, they are executed in the sequence starting from top of the list, similar to `RestTemplate`.

5.2.4 Appendix

How to configure HTTP Proxy server

When the server is to be accessed via HTTP Proxy server, HTTP Proxy server must be configured in system property or JVM starting argument, or Bean definition of `RestTemplate`. When the server is configured in system property or JVM starting argument, it impacts the overall application. Hence, an example wherein HTTP Proxy server is configured for each `RestTemplate` is introduced.

HTTP Proxy server for each `RestTemplate` can be configured for `SimpleClientHttpRequestFactory` which is a default implementation of `ClientHttpRequestFactory` interface. However, since credentials cannot be configured in `SimpleClientHttpRequestFactory`, `HttpComponentsClientHttpRequestFactory` is used while authenticating Proxy. `HttpComponentsClientHttpRequestFactory` is an implementation class of `ClientHttpRequestFactory` interface which generates a request using `Apache HttpComponents HttpClient`.

How to specify a HTTP Proxy server

Connection destination of HTTP Proxy server for which credentials are essential is specified for `RestTemplate` by using `HttpComponentsClientHttpRequestFactory`.

pom.xml

```
<!-- (1) -->
<dependency>
    <groupId>org.apache.httpcomponents</groupId>
    <artifactId>httpclient</artifactId>
</dependency>
```

Sr. No.	Description
(1)	Add `Apache HttpComponents Client` to dependent library of `pom.xml` in order to use `Apache HTTP Client` which is used in `HttpComponentsClientHttpRequestFactory`.

Note: In the above setting example, since it is assumed that the dependent library version is managed by the parent project terasoluna-gfw-parent , specifying the version in pom.xml is not necessary. The above dependent library used by terasoluna-gfw-parent is defined by Spring IO Platform.

Bean definition file

```
<!-- (1) -->
<bean id="proxyHttpClientBuilder" class="org.apache.http.impl.client.
↪HttpClientBuilder" factory-method="create" >
    <!-- (2) -->
    <property name="proxy">
        <bean class="org.apache.http.HttpHost" >
            <constructor-arg index="0" value="${rscl.http.proxyHost}" />    <!--␣
↪(3) -->
            <constructor-arg index="1" value="${rscl.http.proxyPort}" />    <!--␣
↪(4) -->
        </bean>
    </property>
</bean>

<!-- (5) -->
<bean id="proxyRestTemplate" class="org.springframework.web.client.RestTemplate" >
    <constructor-arg>
        <!-- (6) -->
        <bean class="org.springframework.http.client.
↪HttpComponentsClientHttpRequestFactory">
            <!-- (7) -->
            <constructor-arg>
```

```
                <bean factory-bean="proxyHttpClientBuilder" factory-method="build"␣
↪/>
            </constructor-arg>
        </bean>
    </constructor-arg>
</bean>
```

Sr. No.	Description
(1)	Use `org.apache.http.impl.client.HttpClientBuilder` and configure `org.apache.http.client.HttpClient`.
(2)	Configure `org.apache.http.HttpHost` which performs HTTP Proxy server setting, in `proxy` property of `HttpClientBuilder`.
(3)	Set value of key `rscl.http.proxyHost` configured in property file in the first argument of `HttpHost` constructor, as a host name of HTTP Proxy server.
(4)	Set value of key `rscl.http.proxyPort` configured in property file in the second argument of `HttpHost` constructor, as a port number of HTTP Proxy server.
(5)	Define a Bean for `RestTemplate`.
(6)	Configure `HttpComponentsClientHttpRequestFactory` in the argument of `RestTemplate` constructor.
(7)	Configure `HttpClient` generated from `HttpClientBuilder` in the argument of `HttpComponentsClientHttpRequestFactory` constructor.

How to specify credentials information of HTTP Proxy server

When credentials (user name and password) are required for accessing HTTP Proxy server, credentials are set by using `org.apache.http.impl.client.BasicCredentialsProvider`.

Since `setCredentials` method of `BasicCredentialsProvider` contains two arguments, a Bean cannot be generated by using a setter injection. Hence, a Bean is generated by using `org.springframework.beans.factory.FactoryBean`.

FactoryBean class

```
import org.apache.http.auth.AuthScope;
import org.apache.http.auth.UsernamePasswordCredentials;
import org.apache.http.impl.client.BasicCredentialsProvider;
import org.springframework.beans.factory.FactoryBean;
import org.springframework.beans.factory.annotation.Value;

// (1)
public class BasicCredentialsProviderFactoryBean implements FactoryBean
↪<BasicCredentialsProvider> {

    // (2)
    @Value("${rscl.http.proxyHost}")
    String host;

    // (3)
    @Value("${rscl.http.proxyPort}")
    int port;

    // (4)
    @Value("${rscl.http.proxyUserName}")
    String userName;

    // (5)
    @Value("${rscl.http.proxyPassword}")
    String password;

    @Override
    public BasicCredentialsProvider getObject() throws Exception {

        // (6)
        AuthScope authScope = new AuthScope(this.host, this.port);

        // (7)
        UsernamePasswordCredentials usernamePasswordCredentials =
                new UsernamePasswordCredentials(this.userName, this.password);

        // (8)
        BasicCredentialsProvider credentialsProvider = new␣
↪BasicCredentialsProvider();
```

```
        credentialsProvider.setCredentials(authScope, usernamePasswordCredentials);

        return credentialsProvider;
    }

    @Override
    public Class<?> getObjectType() {
        return BasicCredentialsProvider.class;
    }

    @Override
    public boolean isSingleton() {
        return true;
    }
}
```

Sr. No.	Description
(1)	Define a `BasicCredentialsProviderFactoryBean` class which implements `org.springframework.beans.factory.FactoryBean`. Configure `BasicCredentialsProvider` in the type of Bean.
(2)	Set value of key `rscl.http.proxyHost` set in property file in instance variable, as a host name of HTTP Proxy server.
(3)	Set value of key `rscl.http.proxyPort` set in property file in instance variable, as a port number of HTTP Proxy server.
(4)	Set value of key `rscl.http.proxyUserName` set in property file in instance variable, as a user name of HTTP Proxy server.
(5)	Set value of key `rscl.http.proxyPassword` set in property file in instance variable, as a password of HTTP Proxy server.
(6)	Create `org.apache.http.auth.AuthScope` and configure scope of credentials. This example specifies host name and port number of HTTP Proxy server. For other configuration methods, refer AuthScope (Apache HttpClient API).
(7)	Create `org.apache.http.auth.UsernamePasswordCredentials` and configure credentials.
(8)	Create `org.apache.http.impl.client.BasicCredentialsProvider` and configure credentials and its scope by using `setCredentials` method.

Bean definition file

```
<bean id="proxyHttpClientBuilder" class="org.apache.http.impl.client.
↪HttpClientBuilder" factory-method="create">
    <!-- (1) -->
    <property name="defaultCredentialsProvider">
        <bean class="com.example.restclient.BasicCredentialsProviderFactoryBean" />
    </property>
    <property name="proxy">
        <bean id="proxyHost" class="org.apache.http.HttpHost">
            <constructor-arg index="0" value="${rscl.http.proxyHost}" />
            <constructor-arg index="1" value="${rscl.http.proxyPort}" />
        </bean>
    </property>
</bean>

<bean id="proxyRestTemplate" class="org.springframework.web.client.RestTemplate">
    <constructor-arg>
        <bean class="org.springframework.http.client.
↪HttpComponentsClientHttpRequestFactory">
            <constructor-arg>
                <bean factory-bean="proxyHttpClientBuilder" factory-method="build"␣
↪/>
            </constructor-arg>
        </bean>
    </constructor-arg>
</bean>
```

Sr. No.	Description
(1)	Configure `BasicCredentialsProvider`in `defaultCredentialsProvider`property of `HttpClientBuilder`. `BasicCredentialsProvider` creates a Bean by using `BasicCredentialsProviderFactoryBean`which implements `FactoryBean`.

Configuration while using JSR-310 Date and Time API in JSON

For configuration while using JSR-310 Date and Time API as a property of JavaBean which represents a resource (Resource class), refer "*Configuration while using JSR-310 Date and Time API / Joda Time* ".

5.3 SOAP Web Service (Server/Client)

5.3.1 Overview

This chapter explains fundamental concepts of SOAP Web Service and development of both SOAP server and client which use JAX-WS.

For basic idea of how to implement, refer

- "*How to use*"

  It explains application configuration and how to implement API for SOAP Web Service which use JAX-WS.

SOAP

SOAP is a protocol which sends/receives XML messages between computer networks.

Originally, it was an abbreviation of "**S**imple **O**bject **A**ccess **P**rotocol".

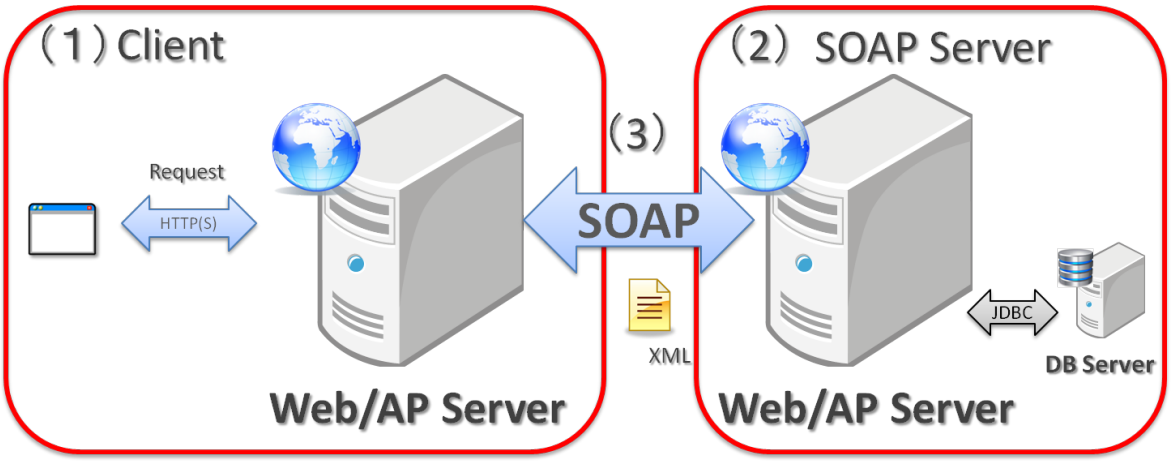
However, now "SOAP" is not considered as an abbreviation and declared as a proper noun by W3C.

Specifications of SOAP1.1 and SOAP1.2 are defined by W3C.

For details, refer W3C -SOAP Specifications-.

How to use SOAP Web Service with the configuration given in following figure is explained in this guideline.

However, SOAP Web Service can also be used with a configuration different from the configuration given below. (Example: when a client is a batch etc)



Sr. No.	Description
(1)	A Web application which communicates with another SOAP server is assumed as a client. Although it is referred as a client, precautions must be taken since it is envisaged as a Web application.
(2)	SOAP server publishes a Web service and performs a process by receiving XML through SOAP Web Service from client. Operations like accessing database etc and performing business process are assumed.
(3)	In SOAP Web Service, information is exchanged by using XML. Here, both SOAP server and client are assumed to be in Java, however, communication is possible in other platforms as well without any issues.

JAX-WS

JAX-WS is an abbreviation of "**J**ava **A**PI for **X**ML-Based **W**eb **S**ervices" and is a Java standard API for handling Web service using SOAP etc.

Using JAX-WS, Java object can be sent by converting the same to XML in conformance with SOAP specifications.

Therefore, although information is exchanged in SOAP Web Service using XML, the user can handle the data without being aware of XML structure.

Main Java EE servers like Oracle WebLogic Server or JBoss Enterprise Application Platform use JAX-WS implementation on server side and can easily publish Web service by using the function without adding a specific library.

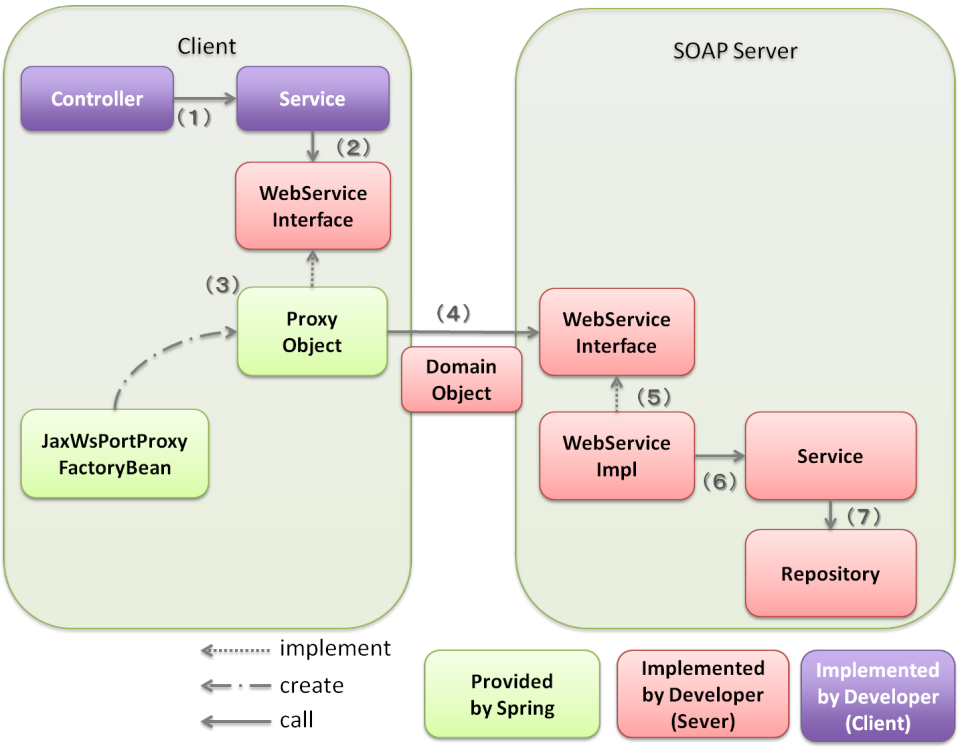
However, since Tomcat does not implement JAX-WS, a separate JAX-WS implementation library must be added while using Tomcat.

For details, refer "*Web service development on Tomcat*".

JAX-WS linkage function of Spring Framework

Spring Framework supports JAX-WS linkage function and implementation can be easily done for both SOAP server and client using this function.

Overview of the recommended access flow using this function is given below. Here, it is assumed that the Web application acting as a SOAP client (Fig. on the left) access SOAP server (fig. on the right).



Sr. No.	Description
(1)	[Client] Controller calls the Service. No specific changes are observed during normal calling.
(2)	[Client] Service calls WebService interface offered by SOAP server side. In the Fig., Service calls the WebService interface, however, WebService interface can also be called directly from Controller if required.
(3)	[Client] If WebService interface is called, Proxy Object is called as an entity. The Proxy Object is an implementation class of WebService interface generated in `org.springframework.remoting.jaxws.JaxWsPortProxyFactoryBean`. The implementation class is injected in the Service, Service can carry out the process using SOAP Web Service only by calling WebService interface method.
(4)	ProxyObject calls WebService interface of SOAP server. Values are exchanged between SOAP server and client by using Domain Object. **Note:** Strictly speaking, SOAP server and client communicate using XML. Although Domain Object and XML are mutually converted using JAXB during sending and receiving, SOAP Web Service creator can carry out development without being aware of XML.
(5)	[Server] If WebService interface gets called, WebService implementation class is called as an entity. A WebService implementation class is provided as an implementation class of WebService interface in SOAP server. The WebService implementation class can inject the Bean of Spring DI container by `@Inject` etc., by inheriting `org.springframework.web.context.support.SpringBeanAutowiringSupport`.
(6)	[Server] Call Service for carrying out business process in WebService implementation class.

次のページに続く

Table. 5.15 – 前のページからの続き

Sr. No.	Description
(7)	[Server] Run business process by using Repository etc in Service. No specific changes are observed during normal calling.

Note: Although a document driven Spring Web Service which develops a Web service is provided in the Spring, it is not addressed in this guideline. For details, refer Spring Web Services.

Note: For details of JAX-WS implementation in Spring, refer Spring Framework Reference Documentation -Remoting and web services using Spring(Web services)-.

Development of Web service using JAX-WS

It is recommended to develop Web service in TERASOLUNA Server Framework for Java (5.x) using JAX-WS implementation of AP server and Spring function.

Note: Deploying to AP server

SOAP Web Service can be implemented for SOAP server or client by deploying a WAR file created by a web project in the blank project, to AP server, similar to a normal Web application.

Configuration of Web service module which uses JAX-WS

When Web service using JAX-WS is to be created, it is recommended to separately add two projects given below besides the existing blank project.

- model project
- webservice project

model project stores Domain Object used in argument and return value of Web service.

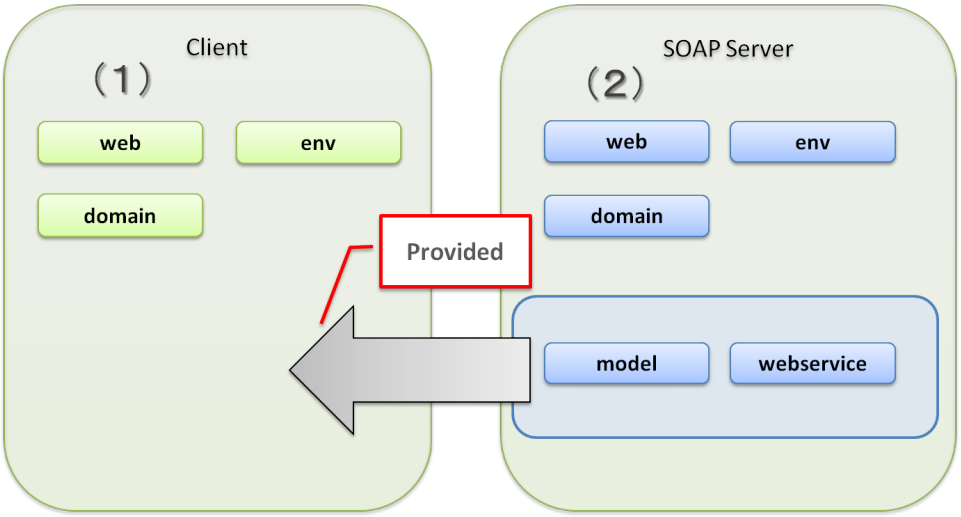
webservice project stores an interface which calls the Web service.

Both the projects store only the classes that must be distributed from SOAP server to client.

It is recommended to use another project to clearly identify the scope of distribution.

In this guideline, following configuration is used in the multi-projects.

Although client is again assumed to be a Web application, the basic idea for calling from desktop application or command line interface remains the same.



Sr. No.	Description
(1)	Add model project and webservice project offered by SOAP server to a conventional multi-project while creating a client. Here, it is assumed that both the server and the client are developed together. The project details are explained in "*How to create SOAP server*". Refer "*Project configuration is changed for SOAP server.*" for how to add. When server and client are not developed separately, and model project and webservice project are not provided, or SOAP server is created in other than Java, Domain Object of model project and Web service interface in webservice project must be created on their own. Domain Object and Web service interface can be easily created from WSDL by using wsimport. For details, refer "*wsimport*".
(2)	Add a model project and webservice project besides a conventional multi-project while creating a SOAP server. Publish these two projects to the client. It is assumed that the model and webservice projects of the client are added in the Maven dependencies.

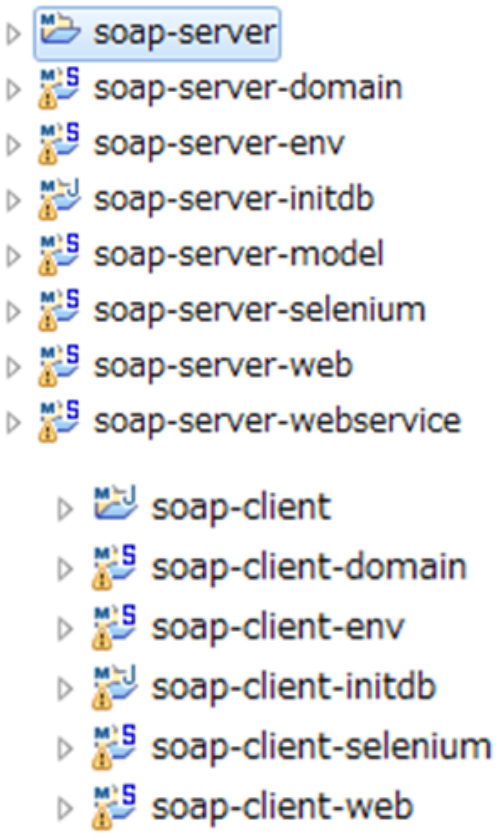
As a result, project is configured as below.

SOAP server project configuration is as given below.

Client project configuration is as given below.

URL to be published as Web service

If SOAP Web Service is created, definition of Web service interface called WSDL（**W**eb **S**ervices **D**escription **L**anguage） is published and the client then implements SOAP Web Service based on the definition.



For WSDL details, refer W3C -Web Services Description Language (WSDL)-.

Access URL, method name, argument and return value for Web service implementation are defined in WSDL.
If SOAP Web Service is created as per this guideline, WSDL is published under following URL.
The URL must be specified at the client side.

- *http://AAA.BBB.CCC.DDD:XXXX/ Context root/Web service name?wsdl*

End point address defined in WSDL consists of following URL..

- *http://AAA.BBB.CCC.DDD:XXXX/ Context root/Web service name*

Note: In the guideline, it is assumed that a WAR file is used for deploying a web project of multi-project configuration in AP server. In this case, [server projectName]-web basically acts as a context root. However, it must be noted that root changes depending on AP server.

Note: In this guideline, since it is assumed that SOAP server and client are together published as a Web application, WSDL URL is specified in the client. The client can also be created by providing WSDL as a file instead of

URL. For details, refer *Implementation of Web service client*.

Warning: In this guideline, AP server (library to be used in case of Tomcat) is configured to change mapping of context root and access by following URL.

- *http://AAA.BBB.CCC.DDD:XXXX/[server projectName]-web/ws/TodoWebService?wsdl*

How to map Web service in URL which is not under context root differs according to AP server. Refer following for details.

Sr. No.	AP server name	Description
(1)	Apache Tomcat	*Web service development on Tomcat*
(2)	Oracle WebLogic Server	TBD
(3)	JBoss Enterprise Application Platform	TBD

5.3.2 How to use

This section specifically explains how to create SOAP Web Service.

How to create SOAP server

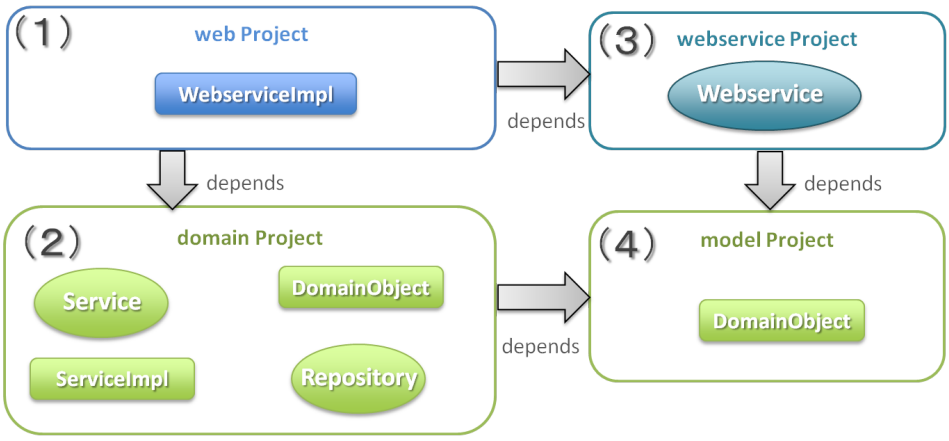
Project configuration

Dependency for each project

model project and webservice project are added as described in "*Development of Web service using JAX-WS*".

Refer "*Project configuration is changed for SOAP server.*" for how to add.

Also note that a dependency must be added to an existing project accordingly.



Sr. No.	Project Name	Description
(1)	web project	Deploy a Web service implementation class.
(2)	domain project	Deploy Service which is called from WebService implementation class. Repository etc are same as used in the conventional project.
(3)	webservice project	Deploy interface of WebService to be published here. Client runs Web service using this interface.
(4)	model project	Deploy only the class that is used in SOAP Web Service from the classes that belong to the domain layer. Input value and return results from the client use the class in the project.

Application configuration

Default configuration while publishing Web service

When Tomcat is to be used as AP server, "*Web service development on Tomcat*" must be implemented.

Besides, since the method to publish Web service is different according to AP server, refer manual of each AP server for details.

Note: AP server manual is explained below as the reference material. It must be always checked that appropriate version of the manual is being referred.

Oracle WebLogic Server 12.2.1: Oracle(R) Fusion Middleware Understanding WebLogic Web Services for Oracle WebLogic Server Features and Standards Supported by WebLogic Web Services

JBoss Enterprise Application Platform 7.0: DEVELOPING JAX-WS WEB SERVICES

JBoss Enterprise Application Platform 7.0: DEVELOPMENT GUIDE JAX-WS WEB SERVICES

Configuration of component scan of package

`[server projectName]-ws.xml` is created for scanning the component to be used by Web service, component scan is defined and injection in the Web service is enabled.

[server projectName]-web/src/main/resources/META-INF/spring/[server projectName]-ws.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans" xmlns:xsi="http://www.
↪w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="
         http://www.springframework.org/schema/beans
         http://www.springframework.org/schema/beans/spring-beans.xsd
         http://www.springframework.org/schema/context
         http://www.springframework.org/schema/context/spring-context.xsd">
    <!-- (1) -->
    <context:component-scan base-package="com.example.ws" />
</beans>
```

Sr. No.	Description
(1)	Specify a package wherein the component to be used in Web service is stored.

[server projectName]-web/src/main/webapp/WEB-INF/web.xml

```
<context-param>
    <param-name>contextConfigLocation</param-name>
    <!-- Root ApplicationContext -->
    <!-- (1) -->
    <param-value>
        classpath*:META-INF/spring/applicationContext.xml
        classpath*:META-INF/spring/spring-security.xml
        classpath*:META-INF/spring/[server projectName]-ws.xml
    </param-value>
</context-param>
```

Sr. No.	Description
(1)	Add `[server projectName]-ws.xml` to reading target while generating a root `ApplicationContext`.

Definition for input check

Following definition is added to input check for using method validation.

For input check details, refer *Input check implementation*.

[server projectName]-web/src/main/resources/META-INF/spring/applicationContext.xml

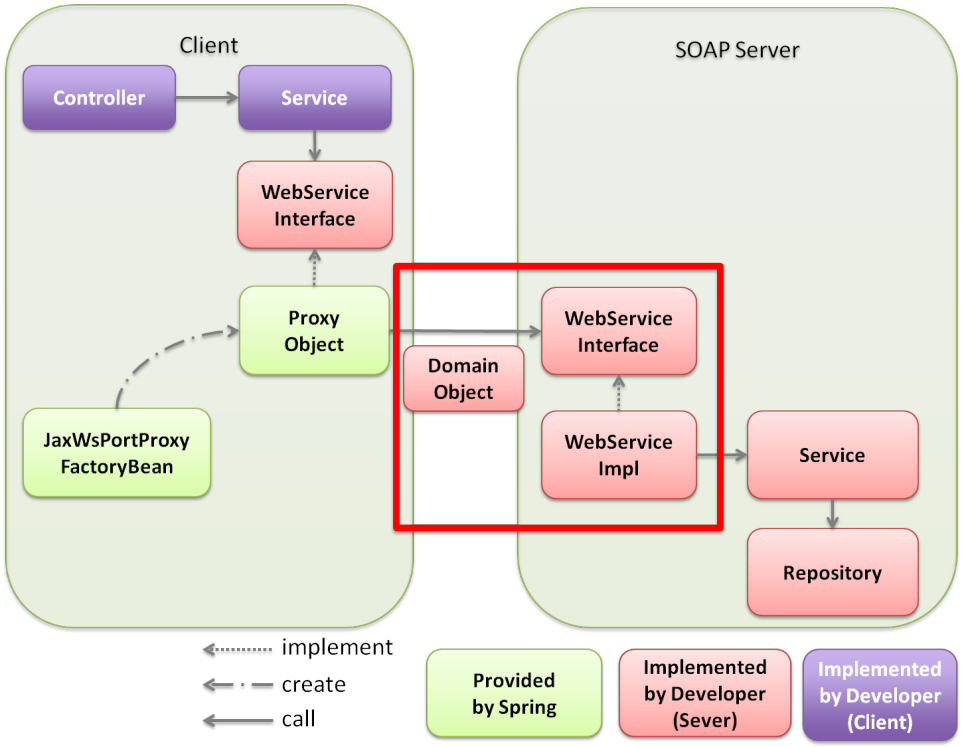
```
<bean class="org.springframework.validation.beanvalidation.
↪MethodValidationPostProcessor">
    <property name="validator" ref="validator" />
</bean>
```

```
<bean id="validator" class="org.springframework.validation.beanvalidation.
↪LocalValidatorFactoryBean" />
```

Web service implementation

Following are created.

- Creating Domain Object
- Creating WebService interface
- Creating WebService implementation class



Creating Domain Object

Domain Object used in the argument and return value of Web service is created in model project.

It is not specifically different from general JavaBean which implements `java.io.Serializable` interface.

[server projectName]-model/src/main/java/com/example/domain/model/Todo.java

```
package com.example.domain.model;

import java.io.Serializable;
import java.util.Date;

public class Todo implements Serializable {

    private String todoId;

    private String title;

    private String description;

    private boolean finished;

    private Date createdAt;

    // omitted setter and getter

}
```

Creating WebService interface

An interface to call Web service is created in webservice project.

[server projectName]-webservice/src/main/java/com/example/ws/todo/TodoWebService.java

```
package com.example.ws.todo;

import java.util.List;

import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebResult;
import javax.jws.WebService;

import com.example.domain.model.Todo;
import com.example.ws.webfault.WebFaultException;
```

```
@WebService(targetNamespace = "http://example.com/todo") // (1)
public interface TodoWebService {

    @WebMethod // (2)
    @WebResult(name = "todo") // (3)
    Todo getTodo(@WebParam(name = "todoId") /* (4) */ String todoId) throws
↪WebFaultException;

}
```

Sr. No.	Description
(1)	WebService interface is declared by applying `@WebService`. Although namespace is defined in the `targetNamespace` attribute, it is recommended to match it with the package name of the Web service used for creating it. **Warning:** Value of `targetNamespace` attribute should be unique. Therefore, it must be changed while diverting the source in the guideline. **Note:** Value of `targetNamespace` attribute is defined in WSDL. Namespace of the Web service is determined and is used for unique identification.
(2)	Apply `@WebMethod` to the method which is published as a Web service method. Method can be published on WSDL and used externally by applying this annotation.
(3)	Apply `@WebResult` to return value and specify name in `name` attribute. It is not required in the absence of a return value. It is published as a return value on WSDL by applying this annotation.
(4)	Apply `@WebParam` in the argument and specify name in `name` attribute. Argument is published on WSDL and defined as a parameter required for external calling, by applying this annotation. For details of `WebFaultException`, refer "*Implementing exception handling*".

Note: How to apply package name and namespace

When the package name is in the following format

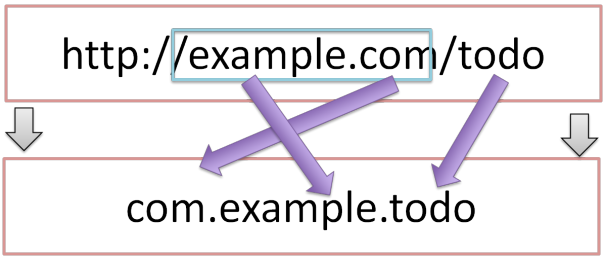
- [Domain].[Application name (System name)].ws.[Used case name]

In this guideline, it is recommended to use namespace as given below..

- http:// [Domain]/[Application name (System name)]

Note: Relation between namespace and package name

When com.example is used as a domain and todo is used as an application name, Namespace is linked with Java package as below.



Although it is not specified, naming of Namespace and package is summarised in XML Namespace Mapping(Red Hat JBoss Fuse).

Creating WebService implementation class

Create an implementation class of WebService interface in web project.

[server projectName]-web/src/main/java/com/example/ws/todo/TodoWebServiceImpl.java

```
package com.example.ws.todo;

import java.util.List;

import javax.inject.Inject;
import javax.jws.HandlerChain;
import javax.jws.WebService;
import javax.xml.ws.BindingType;
import javax.xml.ws.soap.SOAPBinding;

import org.springframework.web.context.support.SpringBeanAutowiringSupport;

import com.example.domain.model.Todo;
import com.example.domain.service.TodoService;
import com.example.ws.webfault.WebFaultException;
import com.example.ws.exception.WsExceptionHandler;
import com.example.ws.todo.TodoWebService;


@WebService(
        portName = "TodoWebPort",
```

```
        serviceName = "TodoWebService",
        targetNamespace = "http://example.com/todo",
        endpointInterface = "com.example.ws.todo.TodoWebService") // (1)
@BindingType(SOAPBinding.SOAP12HTTP_BINDING) // (2)
public class TodoWebServiceImpl extends SpringBeanAutowiringSupport implements␣
↪TodoWebService { // (3)

    @Inject // (4)
    TodoService todoService;

    @Override // (5)
    public Todo getTodo(String todoId) throws WebFaultException {
        return todoService.getTodo(todoId);
    }

}
```

Sr. No.	Description
(1)	WebService implementation class is declared by applying `@WebService`. `portName` attribute is published as a port name on WSDL. `serviceName` attribute is published as a service name on WSDL. `targetNamespace` attribute is a namespace used on WSDL. `endpointInterface` attribute defines an interface name of Web service implemented by this class. **Note:** `portName` attribute, `serviceName` attribute and `endpointInterface` attribute should not be set in `TodoWebService` interface as `@WebService` attribute. This is because the interface corresponds to `portType` element on WSDL and is not the element for describing Web service details.
(2)	Specify binding method by applying `@BindingType`. When `SOAPBinding.SOAP12HTTP_BINDING` is defined, it acts as a binding in SOAP1.2. If annotation is not applied, binding in SOAP1.1 is used.
(3)	Implement `TodoWebService` interface created earlier. Enable DI for Spring Bean by inheriting `org.springframework.web.context.support.SpringBeanAutowiringSupport`.
(4)	Inject Service. Same as while calling the service in normal Controller.
(5)	Run business process by calling Service. Same as while calling the service in normal Controller.

Note: It is recommended to summarise Web service related class under ws package. This is to differentiate it from application layer class which is placed under app package.

Input check implementation

A method validation provided by Spring is used in the input check of parameters sent by SOAP Web Service.

For the details of method validation, refer *How to define for the method for Method Validation target*.

Input check details are defined in Service interface as given below.

[server projectName]-domain/src/main/java/com/example/domain/service/todo/TodoService.java

```
package com.example.domain.service.todo;

import java.util.List;

import javax.validation.Valid;
import javax.validation.constraints.NotNull;
import javax.validation.groups.Default;

import org.springframework.validation.annotation.Validated;

import com.example.domain.model.Todo;

@Validated // (1)
public interface TodoService {

    Todo getTodo(@NotNull String todoId); // (2)

    Todo createTodo(@Valid Todo todo); // (3)

    @Validated({ Default.class, Todo.Update.class }) // (4)
    Todo updateTodo(@Valid Todo todo);

}
```

Sr. No.	Description
(1)	Implementation class of this interface is declared as a target for input check by applying `@Validated`.
(2)	Apply annotation to argument itself while checking the argument.
(3)	Apply `@Valid` in the argument while carrying out input check of JavaBean.
(4)	Input check can also be carried out by specifying `@Validated` in the group and narrowing down the specific conditions. Group details are explained in the JavaBean described next.

[server projectName]-model/src/main/java/com/example/domain/model/Todo.java

```
package com.example.domain.model;

import javax.validation.constraints.NotNull;
import javax.validation.constraints.Null;
import java.io.Serializable;
import java.util.Date;

// (1)
public class Todo implements Serializable {

    // (2)
    public interface Create {
    }

    public interface Update {
    }

    @Null(groups = Create.class)
    @NotNull(groups = Update.class)
    private String todoId;
```

```
    @NotNull
    private String title;

    private String description;

    private boolean finished;

    @Null(groups = Create.class)
    private Date createdAt;

    // omitted setter and getter
}
```

Sr. No.	Description
(1)	Define JavaBean input check in the Bean Validation. For details, refer "*Input Validation*".
(2)	Define an interface used for grouping of validation.

Security measures

Authentication process

How to carry out Basic authentication in Spring Security and how to authorize in Service are introduced in the guideline as the methods of authentication and authorization for SOAP.

WS-Security is not addressed.

For details of how to use, refer "*Authentication*" and "*Authorization*".

A configuration example of Spring Security carrying out Basic authentication for SOAP Web Service is shown below.

[server projectName]-web/src/main/resources/META-INF/spring/spring-security.xml

```
<sec:http pattern="/ws/**"
          create-session="stateless">
   <sec:csrf disabled="true" />
   <sec:http-basic />  <!-- (1) -->
</sec:http>

<!-- (2) -->
<sec:authentication-manager>
   <sec:authentication-provider
       user-service-ref="sampleUserDetailsService">
       <sec:password-encoder ref="passwordEncoder" />
   </sec:authentication-provider>
</sec:authentication-manager>
```

Sr. No.	Description
(1)	Basic authentication can be carried out if `sec:http-basic` tag is described. Authentication is carried out only for the Web service execution by using `pattern` attribute.
(2)	Define authentication method by using `authentication-provider`. Actual authentication and fetching user information must be implemented by creating `UserDetailsService`. For details, refer "*Authentication*".

Authorization process

Authorization is carried out by applying annotation for each Service.

For details, refer access authorization of "*Authorization*(method)".

[server projectName]-web/src/main/resources/META-INF/spring/spring-security.xml

```
<sec:global-method-security pre-post-annotations="enabled" /> <!-- (1) -->
```

Sr. No.	Description
(1)	Specify `pre-post-annotations` attribute of `<sec:global-method-security>` element in `enabled`.

[server projectName]-domain/src/main/java/com/example/domain/service/todo/TodoServiceImpl.java

```
public class TodoServiceImpl implements TodoService {

    // omitted

    // (1)
    @PreAuthorize("isAuthenticated()")
    public List<Todo> getTodos() {
        // omitted
    }

    @PreAuthorize("hasRole('ROLE_ADMIN')")
    public Todo createTodo(Todo todo) {
        // omitted
    }

}
```

Sr. No.	Description
(1)	Specify `org.springframework.security.access.prepost.PreAuthorize` annotation in the method which carries out authorization.

CSRF measures

SOAP Web Service should be used in the stateless communication without using a session.

Therefore, a configuration method wherein CSRF measures using the session are not employed is explained below.

For details of CSRF, refer "*CSRF Countermeasures*".

CSRF measures are enabled in the default configuration of Blank project.

Therefore, CSRF measures processing is disabled for SOAP Web Service request by adding following configuration.

[server projectName]-web/src/main/resources/META-INF/spring/spring-security.xml

```
<!-- (1) -->
<sec:http pattern="/ws/**"
    create-session="stateless">
    <sec:http-basic />
    <sec:csrf disabled="true" />
</sec:http>
```

Sr. No.	Description
(1)	Add definition of Spring Security for SOAP Web Service. Specify URL pattern of request path for SOAP Web Service in `pattern` attribute of `<sec:http>` element. In this code example, request path starting at `/ws/` acts as a request path for SOAP Web Service. Further, the session can no longer be used in Spring Security process by making `create-session` attribute `stateless`. Set `disabled` attribute of `<sec:csrf>` element to `true` for disabling CSRF measures.

Implementation of exception handling

An exclusive exception class must be thrown for communicating with the client when an exception occurs in SOAP server.

Implementation is described below.

Exception occurred in SOAP server

Exception occurred at SOAP server can determine the notification message to the client by using class (SOAP-Fault) which implements exception described henceforth.

Basically the class given below is created.

Sr. No.	Class Name	Overview
(1)	`ErrorBean`	A class which retains code and message of occurred exception.
(2)	`WebFaultType`	Enum type used to determine the type of exception.
(3)	`WebFaultBean`	A class which retains `ErrorBean` and `WebFaultType`. Multiple exception information can be retained by retaining `ErrorBean` in `List`.
(4)	`WebFaultException`	Exception class which retains `WebFaultBean`.

These exceptions are placed on [server projectName]-webservice since these are shared by SOAP server and client.

[server projectName]-webservice/src/main/java/com/example/ws/webfault/ErrorBean.java

```
package com.example.ws.webfault;

public class ErrorBean { // (1)
    private String code;
    private String message;
    private String path;

    // omitted setter and getter
}
```

Sr. No.	Description
(1)	Create a class which retains exception messages etc.

[server projectName]-webservice/src/main/java/com/example/ws/webfault/WebFaultType.java

```
package com.example.ws.webfault;

public enum WebFaultType { // (2)
    AccessDeniedFault,
    BusinessFault,
    ResourceNotFoundFault,
    ValidationFault,
}
```

Sr. No.	Description
(1)	Define an enum type used to identify type of exception.

[server projectName]-webservice/src/main/java/com/example/ws/webfault/WebFaultBean.java

```
package com.example.ws.webfault;

import java.util.ArrayList;
import java.util.List;

public class WebFaultBean { // (3)

    private WebFaultType type;

    private List<ErrorBean> errors = new ArrayList<ErrorBean>();

    public WebFaultBean(WebFaultType type) {
        this.type = type;
    }

    public void addError(String code, String message) {
```

```
        addError(code, message, null);
    }

    public void addError(String code, String message, String path) {
        errors.add(new ErrorBean(code, message, path));
    }

    // omitted setter and getter
}
```

Sr. No.	Description
(1)	Create a class which retains `ErrorBean` and `WebFaultType`.

[server projectName]-webservice/src/main/java/com/example/ws/webfault/WebFaultException.java

```
package com.example.ws.webfault;

import java.util.List;

import javax.xml.ws.WebFault;

@WebFault(name = "WebFault", targetNamespace = "http://example.com/todo") // (1)
public class WebFaultException extends Exception {
    private WebFaultBean faultInfo; // (2)

    public WebFaultException() {
    }

    public WebFaultException(String message, WebFaultBean faultInfo) {
        super(message);
        this.faultInfo = faultInfo;
    }

    public WebFaultException(String message, WebFaultBean faultInfo, Throwable e) {
        super(message, e);
        this.faultInfo = faultInfo;
    }

    public List<ErrorBean> getErrors() {
        return this.faultInfo.getErrors();
    }
```

```
    public WebFaultType getType() {
        return this.faultInfo.getType();
    }
    // omitted setter and getter
}
```

Sr. No.	Description
(1)	Declare SOAPFault by applying `@WebFault` to Exception inheritance class. Specify `name` attribute of SOAPFault sent to client in `name` attribute. Specify namespace to be used in `targetNamespace` attribute. It must be same as in Web service.
(2)	It consists of constructor and method below, as shown in code example besides retaining the faultInfo in the field. • Constructor wherein message string and faultInfo are considered as arguments • Constructor wherein message string, faultInfo and cause exception are considered as arguments • getFaultInfo method

Note: Reason of inheriting Exception in WebFaultException instead of RuntimeException

If parent class of `WebFaultException` is set to `RuntimeException`, exception process can be further simplified. However, parent class should not be set to `RuntimeException`. it is also declared that it cannot be defined in JSR 224: JavaTM API for XML-Based Web Services as well. Although, it depends on JAS-WS implementation of AP server during an actual attempt, exception class (`WebFaultException`) wherein `@WebFault` is applied in the client cannot be fetched resulting in inability to fetch error types and message. Inheriting `Exception`also results in non-implementation of exception process using AOP.

Warning: Constructor and field of WebFaultException

A setter corresponding to each field and default constructor is mandatory in `WebFaultException`. This is an internal process of client and is used while creating `WebFaultException`. Therefore, it is also not possible to consider each field as final.

This `WebFaultException` is inherited, and types to be communicated to the client and child class are created.
For example, child classes are created as given below.

- Business error exception
- Input error exception
- Resource not detected exception
- Exclusive error exception
- Authorization exception
- System error exception

Following is an example of business error exception.

[server projectName]-webservice/src/main/java/com/example/ws/webfault/BusinessFaultException.java

```
package com.example.ws.webfault;

import javax.xml.ws.WebFault;

@WebFault(name = "BusinessFault", targetNamespace = "http://example.com/todo") //␣
↪(1)
public class BusinessFaultException extends WebFaultException {

    public BusinessFaultException(String message, WebFaultBean faultInfo) {
        super(message, faultInfo);
    }

    public BusinessFaultException(String message, WebFaultBean faultInfo,␣
↪Throwable e) {
        super(message, faultInfo, e);
    }

}
```

Sr. No.	Description
(1)	Inherit `WebFaultException` and create only constructor. Field and other methods are not required to be described since parent class method is used.

Exception handler which wraps exceptions that have occurred by SOAPFault

Exception handler class is created for wrapping the run-time exceptions which occur in Service by SOAPFault. This guideline adopts a policy wherein WebService implementation class converts and throws exceptions using this handler.

Exception thrown from Service assumes the following. It should be added when required.

Exception name	Details
`org.springframework.security.access.AccessDeniedException`	Exception at the time of authorization error
`javax.validation.ConstraintViolationException`	Exception at the time of input check error
`org.terasoluna.gfw.common.exception.ResourceNotFoundException`	Exception when resource is not detected
`org.terasoluna.gfw.common.exception.BusinessException`	Business exception

[server projectName]-web/src/main/java/com/example/ws/exception/WsExceptionHandler.java

```
package com.example.ws.exception;

import java.util.Iterator;
import java.util.Locale;
import java.util.Set;

import javax.inject.Inject;
import javax.validation.ConstraintViolation;
import javax.validation.ConstraintViolationException;
import javax.validation.Path;

import org.springframework.context.MessageSource;
import org.springframework.security.access.AccessDeniedException;
import org.springframework.stereotype.Component;
import org.terasoluna.gfw.common.exception.BusinessException;
import org.terasoluna.gfw.common.exception.ExceptionCodeResolver;
import org.terasoluna.gfw.common.exception.ExceptionLogger;
import org.terasoluna.gfw.common.exception.ResourceNotFoundException;
import org.terasoluna.gfw.common.exception.SystemException;
import org.terasoluna.gfw.common.message.ResultMessage;
```

```
import org.terasoluna.gfw.common.message.ResultMessages;

import com.example.ws.webfault.WebFaultBean;
import com.example.ws.webfault.WebFaultException;
import com.example.ws.webfault.WebFaultType;

@Component  // (1)
public class WsExceptionHandler {

    @Inject
    MessageSource messageSource; // (2)

    @Inject
    ExceptionCodeResolver exceptionCodeResolver; // (3)

    @Inject
    ExceptionLogger exceptionLogger; // (4)

    // (5)
    public void translateException(Exception e) throws WebFaultException {
        loggingException(e);
        WebFaultBean faultInfo = null;

        if (e instanceof AccessDeniedException) {
            faultInfo = new WebFaultBean(WebFaultType.AccessDeniedFault);
            faultInfo.addError(e.getClass().getName(), e.getMessage());
        } else if (e instanceof ConstraintViolationException) {
            faultInfo = new WebFaultBean(WebFaultType.ValidationFault);
            this.addErrors(faultInfo, ((ConstraintViolationException) e).
↪getConstraintViolations());
        } else if (e instanceof ResourceNotFoundException) {
            faultInfo = new WebFaultBean(WebFaultType.ResourceNotFoundFault);
            this.addErrors(faultInfo, ((ResourceNotFoundException) e).
↪getResultMessages());
        } else if (e instanceof BusinessException) {
            faultInfo = new WebFaultBean(WebFaultType.BusinessFault);
            this.addErrors(faultInfo, ((BusinessException) e).getResultMessages());
        } else {
            // not translate.
            throw new SystemException("e.ex.fw.9001", e);
        }

        throw new WebFaultException(e.getMessage(), faultInfo, e.getCause());
    }

    private void loggingException(Exception e) {
        exceptionLogger.log(e);
    }

    private void addErrors(WebFaultBean faultInfo, Set<ConstraintViolation<?>>␣
↪constraintViolations) {
```

```
        for (ConstraintViolation<?> v : constraintViolations) {
            Iterator<Path.Node> pathIt = v.getPropertyPath().iterator();
            pathIt.next(); // method name node (skip)
            Path.Node methodArgumentNameNode = pathIt.next();
            faultInfo.addError(
                v.getConstraintDescriptor().getAnnotation().annotationType().
↪getSimpleName(),
                v.getMessage(),
                pathIt.hasNext() ? pathIt.next().toString() :␣
↪methodArgumentNameNode.toString());
        }

    }

    private void addErrors(WebFaultBean faultInfo, ResultMessages resultMessages) {
        Locale locale = Locale.getDefault();
        for (ResultMessage message : resultMessages) {
            faultInfo.addError(
                message.getCode(),
                messageSource.getMessage(message.getCode(), message.getArgs(),␣
↪message.getText(), locale));
        }
    }

}
```

Sr. No.	Description
(1)	Apply `@Component` for managing the class in DI container.
(2)	Use `MessageSource` to fetch the message to be output.
(3)	Use `ExceptionCodeResolverMessageSource` provided by common library and map exception types and exception codes. For details, refer "*Exception Handling*".
(4)	Use `ExceptionLogger` provided by common library and output exception information in the exception. For details, refer "*Exception Handling*".
(5)	Each exception occurring in Service is wrapped in `SOAPFault`. Refer table at the beginning for exception mapping.

Note: Other exception handling

In case of other exceptions (else part of `translateException` method described above), detailed exception details are not notified to the client and only `com.sun.xml.internal.ws.fault.ServerSOAPFaultException` is thrown.Exception can also be wrapped like other exceptions and notified to the client side.

Exception occurred in the Service is wrapped in Web service by calling exception handler

Exception handler is called in Web service class. Example is given below.

[server projectName]-web/src/main/java/com/example/ws/todo/TodoWebServiceImpl.java

```
@WebService(
        portName = "TodoWebPort",
        serviceName = "TodoWebService",
        targetNamespace = "http://example.com/todo",
        endpointInterface = "com.example.ws.todo.TodoWebService")
@BindingType(SOAPBinding.SOAP12HTTP_BINDING)
public class TodoWebServiceImpl extends SpringBeanAutowiringSupport implements␣
↪TodoWebService {
    @Inject
    TodoService todoService;
    @Inject
    WsExceptionHandler handler; // (1)

    @Override
    public Todo getTodo(String todoId) throws WebFaultException /* (2) */ {
        try {
            return todoService.getTodo(todoId);
        } catch (RuntimeException e) {
            handler.translateException(e); // (3)
        }
    }
}
```

Sr. No.	Description
(1)	Inject exception handler.
(2)	Apply throws clause since the exception is thrown after wrapping in `WebFaultException`.
(3)	In case of run-time exception, delegate the process to exception handler class.

How to handle large binary data using MTOM

Sending and receiving process can be carried out in SOAP by mapping Byte array while handling binary data.

However, while handling binary data of large volume, issues like memory exhaustion are likely to occur.

Accordingly, binary data can be handled as attached file in the optimised state by carrying out implementation in compliance with MTOM（Message Transmission Optimization Mechanism).

For detailed definition, refer W3C -SOAP Message Transmission Optimization Mechanism-.

The method is described below.

[server projectName]-webservice/src/main/java/com/example/ws/todo/TodoWebService.java

```
package com.example.ws.todo;

import java.util.List;

import javax.activation.DataHandler;
import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebResult;
import javax.jws.WebService;
import javax.xml.bind.annotation.XmlMimeType;

import com.example.domain.model.Todo;
import com.example.ws.webfault.WebFaultException;

@WebService(targetNamespace = "http://example.com/todo")
public interface TodoWebService {

    // omitted

    @WebMethod
    void uploadFile(@XmlMimeType("application/octet-stream") /* (1) */ DataHandler␣
↪dataHandler) throws WebFaultException;

}
```

Sr. No.	Description
(1)	Apply `@XmlMimeType` for `javax.activation.DataHandler` which processes binary data.

[server projectName]-web/src/main/java/com/example/ws/todo/TodoWebServiceImpl.java

```
package com.example.ws.todo;

import java.io.IOException;
import java.io.InputStream;
import java.util.List;
```

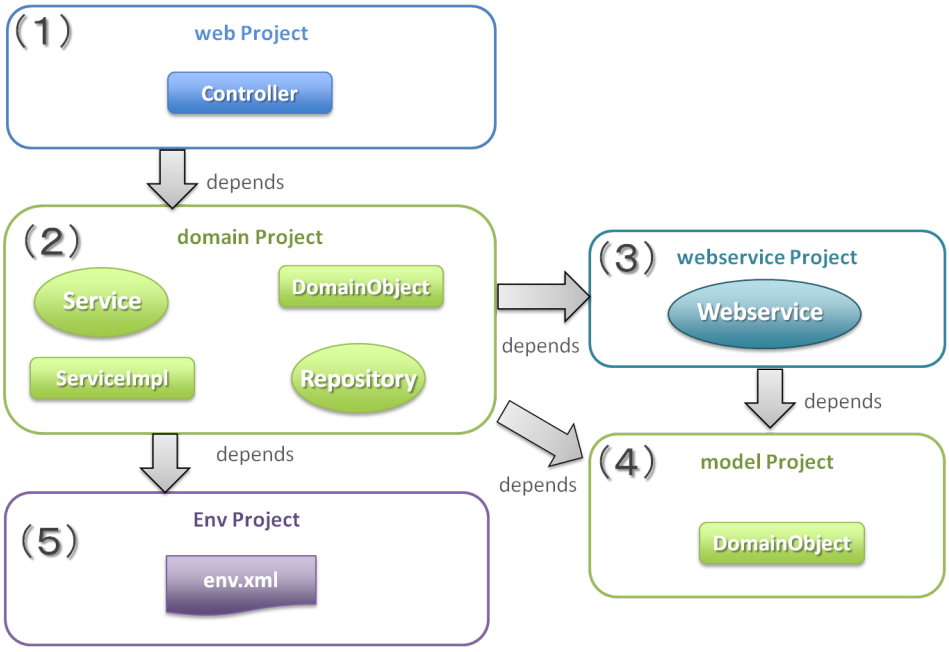
```
import javax.activation.DataHandler;
import javax.inject.Inject;
import javax.jws.HandlerChain;
import javax.jws.WebService;
import javax.xml.ws.BindingType;
import javax.xml.ws.soap.MTOM;
import javax.xml.ws.soap.SOAPBinding;

import org.springframework.web.context.support.SpringBeanAutowiringSupport;
import org.terasoluna.gfw.common.exception.SystemException;

import com.example.domain.model.Todo;
import com.example.domain.service.TodoService;
import com.example.ws.webfault.WebFaultException;
import com.example.ws.exception.WsExceptionHandler;

// (1)
@MTOM
@WebService(
        portName = "TodoWebPort",
        serviceName = "TodoWebService",
        targetNamespace = "http://example.com/todo",
        endpointInterface = "com.example.ws.todo.TodoWebService")
@BindingType(SOAPBinding.SOAP12HTTP_BINDING)
public class TodoWebServiceImpl extends SpringBeanAutowiringSupport implements␣
↪TodoWebService {

    @Inject
    TodoService todoService;

    // omitted

    @Override
    public void uploadFile(DataHandler dataHandler) throws WebFaultException {

        try (InputStream inputStream = dataHandler.getInputStream()){ // (2)
            todoService.uploadFile(inputStream);
        } catch (Exception e) {
            handler.translateException(e);
        }
    }

}
```

Sr. No.	Description
(1)	Apply `@MTOM` and declare the use of implementation in compliance with MTOM.
(2)	Fetch `java.io.InputStream` from `javax.activation.DataHandler` and handle the file.

Creation of client

Project configuration

As described in "*Development of Web service using JAX-WS*", model project and webservice project are assumed to be received by SOAP server.



Sr. No.	Project name	Description
(1)	web project	Create Controller. No specific change in the Controller during normal screen transition.
(2)	domain project	Call Web service by using WebService interface which is provided in webservice project from Service class.
(3)	webservice project	Configure data same as SOAP server. Client uses this interface to implement Web service.
(4)	model project Configure data same as SOAP server.	Use class in the project for input value and return results passed to SOAP server.
(5)	env project	Define a proxy class which implements WebService interface used while communicating with SOAP server. Since proxy class definition is often environment dependent, it is defined in env project.

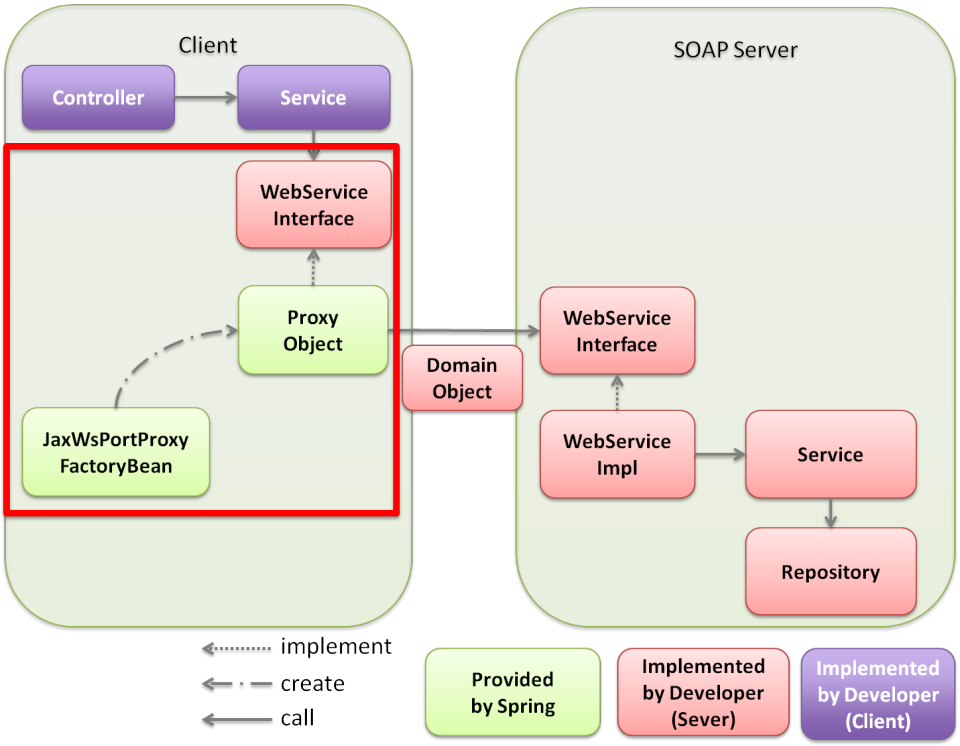
Implementation of Web service client

Implement class as below.

- Define a proxy class which implements WebService interface
- Call Web service from Service class through WebService interface.

Creating proxy class which implements WebService interface

Define `org.springframework.remoting.jaxws.JaxWsPortProxyFactoryBean` generating a



proxy class which implements a WebService interface.

[client projectName]-env/src/main/resources/META-INF/spring/[client projectName]-env.xml

```
<bean id="todoWebService"
    class="org.springframework.remoting.jaxws.JaxWsPortProxyFactoryBean"><!-- (1) -
↪->
    <property name="serviceInterface" value="com.example.ws.todo.TodoWebService" />
↪<!-- (2) -->
    <!-- (3) -->
    <property name="serviceName" value="TodoWebService" />
    <property name="portName" value="TodoWebPort" />
    <property name="namespaceUri" value="http://example.com/todo" />
    <property name="wsdlDocumentResource" value="${webservice.todoWebService.
↪wsdlDocumentResource}" /><!-- (4) -->
</bean>
```

[client projectName]-env/src/main/resources/META-INF/spring/[client projectName]-infra.properties

```
# (5)
webservice.todoWebService.wsdlDocumentResource=http://AAA.BBB.CCC.DDD:XXXX/[server␣
↪projectName]-web/ws/TodoWebService?wsdl
```

Sr. No..	Description
(1)	Define `org.springframework.remoting.jaxws.JaxWsPortProxyFactoryBean`.SOAP server can be accessed through proxy class generated by this class.
(2)	Define an interface that should be implemented by Web service in `serviceInterface` property.
(3)	Details same as defined on the server side must be defined in `serviceName` ,`portName` and `namespaceUri` property respectively.
(4)	Specify URL of WDSL published in `wsdlDocumentResource` property. Property key is specified since URL is described in the property file described later.
(5)	Specify value of property key defined in `[client projectName]-env.xml`.Describe URL of WSDL. **Note: Specify other than URL of WSDL file to wsdlDocumentResource** In the example above, it is assumed that SOAP server publishes WSDL file. A static file can be specified as well by using `classpath:` or `file:` prefix. Refer Spring Framework Reference Documentation -Resources(The ResourceLoader)- for strings that can be specified.

Note: Overwriting end point address

Access URL configuration is not required in the client since access URL at the time of executing Web service (end point address) is described in WSDL file. However, when a URL not described in WSDL file is to be accessed, end point address can be overwritten by configuring `endpointAddress` property of `org.springframework.remoting.jaxws.JaxWsPortProxyFactoryBean`. It should preferably be used while changing the environment in tests etc. Configuration example is as below.

[client projectName]-env/src/main/resources/META-INF/spring/[client projectName]-env.xml

```
<bean id="todoWebService"
    class="org.springframework.remoting.jaxws.JaxWsPortProxyFactoryBean">
    <property name="serviceInterface" value="com.example.ws.todo.
↪TodoWebService" />
    <property name="serviceName" value="TodoWebService" />
    <property name="portName" value="TodoWebPort" />
    <property name="namespaceUri" value="http://example.com/todo" />
    <property name="wsdlDocumentResource" value="${webservice.todoWebService.
↪wsdlDocumentResource}" />
    <property name="endpointAddress" value="${webservice.todoWebService.
↪endpointAddress}" /><!-- (1) -->
</bean>
```

[client projectName]-env/src/main/resources/META-INF/spring/[client projectName]-infra.properties

```
# (2)
webservice.todoWebService.endpointAddress=http://AAA.BBB.CCC.DDD:XXXX/[server␣
↪projectName]-web/ws/TodoWebService
```

Sr. No.	Description
(1)	Specify end point address. A property key is specified for describing URL in property file described later.
(2)	Specify value of property key defined in `[client projectName]-env.xml`. Describe end point address.

Call Web service from Service

Inject Web service created above by Service and run Web service.

[client projectName]-domain/src/main/java/com/example/domain/service/todo/TodoServiceImpl.java

```
package com.example.soap.domain.service.todo;

import java.util.List;

import javax.inject.Inject;
```

```
import org.springframework.stereotype.Service;

import com.example.domain.model.Todo;
import com.example.ws.webfault.WebFaultException;
import com.example.ws.todo.TodoWebService;

@Service
public class TodoServiceImpl implements TodoService {

    @Inject
    TodoWebService todoWebService;

    @Override
    public void createTodo(Todo todo) {
        // (1)
        try {
            todoWebService.createTodo(todo);
        } catch (WebFaultException e) {
            // (2)
            // handle exception…
        }
    }
}
```

Sr. No.	Description
(1)	Inject `TodoWebService` and call Service to be run.
(2)	When an exception occurs at the server side, it is wrapped in `WebFaultException` and sent. Carry out process depending on the details. For details of exception process, refer "*Implementing exception handling*".

Note: Defining proxy class

It is recommended to define proxy class in env project. This is to enable changing implementation class of Web service by changing maven profile. When sending destination of SOAP server for testing is to be changed or when the SOAP server is not ready, the testing can be carried out without changing another source by creating a stub class.

Note: Fetch response information

When the response information is to be fetched by the client for example retry, it can be fetched by casting in `javax.xml.ws.BindingProvider`class as given below.

```
BindingProvider provider = (BindingProvider) todoWebService;
int status = (int) provider.getResponseContext().get(MessageContext.HTTP_
↪RESPONSE_CODE);
```

For details of `BindingProvider`, refer The Java API for XML-Based Web Services(JAX-WS) 2.2 -4.2 javax.xml.ws.BindingProvider-.

However, when Apatch CXF library is included in the dependency relation of the client, it is not possible to fetch response information by the method given above at the time of communication error. This is because Apatch CXF proxy is automatically used when Apatch CXF library is included in the dependency relation and Apache CXF proxy does not retain response information in response context at the time of communication error. For error handling of Apache CXF, refer Apache CXF Software Architecture Guide -Fault Handling-.

According to Web service and relay service which includes client of another Web service, if dependency relation of Apache CXF library is necessarily included in the client, it must be considered as a restricted item and adequate care must be taken.

Security measures

Authentication process

When the communication is to be established with SOAP server which uses Basic authentication while using `org.springframework.remoting.jaxws.JaxWsPortProxyFactoryBean`, authentication can be done only if user name and password are added to bean definition.

[client projectName]-env/src/main/resources/META-INF/spring/[client projectName]-env.xml

```
<bean id="todoWebService"
    class="org.springframework.remoting.jaxws.JaxWsPortProxyFactoryBean">
    <property name="serviceInterface" value="com.example.ws.todo.TodoWebService" />
    <property name="serviceName" value="TodoWebService" />
    <property name="portName" value="TodoWebPort" />
    <property name="namespaceUri" value="http://example.com/todo" />
    <property name="wsdlDocumentResource" value="${webservice.todoWebService.
↪wsdlDocumentResource}" />
    <!-- (1) -->
    <property name="username" value="${webservice.todoWebService.username}" />
    <property name="password" value="${webservice.todoWebService.password}" />
</bean>
```

[client projectName]-env/src/main/resources/META-INF/spring/[client projectName]-infra.properties

```
# (2)
webservice.todoWebService.username=testuser
webservice.todoWebService.password=password
```

Sr. No.	Description
(1)	Authentication information for Basic authentication can be sent by adding user name and password in bean definition of `org.springframework.remoting.jaxws.JaxWsPortProxyFactoryBean`. It is a sample wherein user name and password are transferred to the property file.
(2)	Specify value of property key defined in `[client projectName]-env.xml`. Describe user name and password used for authentication.

Implementing exception handling

In SOAP server, it is recommended to wrap exception in `WebFaultException`and throw the same.

Client catches `WebFaultException`, determines the cause for exception and carry out respective processing.

```
@Override
public void createTodo(Todo todo) {

    try {
        // (1)
        todoWebService.createTodo(todo);
    } catch (WebFaultException e) {
        // (2)
        switch (e.getFaultInfo().getType()) {
        case ValidationFault:
            // handle exception…
            break;
        case BusinessFault:
            // handle exception…
            break;
```

```
        default:
            // handle exception…
            break;
        }
    }

}
```

Sr.No.	Description
(1)	Call Web service. `WebFaultException` must be caught since throws clause is applied.
(2)	Determine exception by `faultInfo`type and describe respective process (sending a message to the screen, throwing an exception etc)

Timeout configuration

Timeout that can be specified by client is broadly classified into following two types.

- Connection timeout for each SOAP server
- Request timeout for each SOAP server

Both the configurations must be specified in the custom property of `org.springframework.remoting.jaxws.JaxWsPortProxyFactoryBean`.
How to configure is as given below.

[client projectName]-env/src/main/resources/META-INF/spring/[client projectName]-env.xml

```
<bean id="todoWebService"
    class="org.springframework.remoting.jaxws.JaxWsPortProxyFactoryBean">
    <property name="serviceInterface" value="com.example.ws.todo.TodoWebService" />
    <property name="serviceName" value="TodoWebService" />
    <property name="portName" value="TodoWebPort" />
    <property name="namespaceUri" value="http://example.com/todo" />
    <property name="wsdlDocumentResource" value="${webservice.todoWebService.
↪wsdlDocumentResource}" />
```

```
    <!-- (1) -->
    <property name="customProperties">
        <map>
            <!-- (2) -->
            <entry key="com.sun.xml.internal.ws.connect.timeout" value-type="java.
↪lang.Integer" value="${webservice.connect.timeout}"/>
            <entry key="com.sun.xml.internal.ws.request.timeout" value-type="java.
↪lang.Integer" value="${webservice.request.timeout}"/>
        </map>
    </property>
</bean>
```

[client projectName]-env/src/main/resources/META-INF/spring/[client projectName]-infra.properties

```
# (3)
webservice.request.timeout=3000
webservice.connect.timeout=3000
```

Sr. No.	Description
(1)	Define a custom property by specifying Map in `customProperties` property.
(2)	Define connection timeout and request timeout. It is a sample wherein the respective values are transferred to a property file. **Warning: Key used for defining timeout** It is necessary to specify a different value based on JAX-WS implementation as a key to define respective timeout. For details, refer JAX_WS-1166 Standardize timeout settings.
(3)	Specify value of property key defined in `[client projectName]-env.xml`. Connection timeout and request timeout are described.

5.3.3 Appendix

Project configuration is changed for SOAP server.

It is recommended to add model project and webservice project to the blank project while creating SOAP server.

The method is described below.

Blank project default configuration is as below.

Note that, artifactId specified while creating a blank project is configured in artifactId.

```
artifactId
├- pom.xml
├- artifactId-domain
├- artifactId-env
├- artifactId-initdb
├- artifactId-selenium
└- artifactId-web
```

Project configuration is as below.

```
artifactId
├- pom.xml
├- artifactId-domain
├- artifactId-env
├- artifactId-initdb
├- artifactId-selenium
├- artifactId-web
├- artifactId-model
└- artifactId-webservice
```

Changing existing project

A simple implementation of Web application like Controller etc is included in the default state of the blank project.

SOAP web service can be used as it is, however, it is recommended to delete the same since it is not required.

For deletion target, refer "*Multi-project structure* of *Create Web application development project* ".

Creating model project

model project configuration is explained.

```
artifactId-model
    ├- pom.xml  ... (1)
```

Sr. No.	Description
(1)	A POM (Project Object Model) file which defines model module configuration. Following are defined in this file. • Definition of plug-ins for dependent libraries and build • Definition for creating jar file

`pom.xml` is as shown in the image below. It must be edited when required.

Actually, the values specified while creating a blank project must be configured for “artifactId” and “groupId”.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/
↪2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0␣
↪http://maven.apache.org/maven-v4_0_0.xsd">

    <modelVersion>4.0.0</modelVersion>
    <artifactId>artifactId-model</artifactId>
    <packaging>jar</packaging>
    <parent>
        <groupId>groupId</groupId>
        <artifactId>artifactId</artifactId>
        <version>1.0.0-SNAPSHOT</version>
        <relativePath>../pom.xml</relativePath>
    </parent>
    <dependencies>
        <!-- == Begin TERASOLUNA == -->
        <dependency>
            <groupId>org.terasoluna.gfw</groupId>
            <artifactId>terasoluna-gfw-common-dependencies</artifactId>
        </dependency>
        <dependency>
            <groupId>org.terasoluna.gfw</groupId>
```

```
            <artifactId>terasoluna-gfw-jodatime-dependencies</artifactId>
            <type>pom</type>
        </dependency>
        <dependency>
            <groupId>org.terasoluna.gfw</groupId>
            <artifactId>terasoluna-gfw-security-core-dependencies</artifactId>
            <type>pom</type>
        </dependency>

        <dependency>
            <groupId>org.terasoluna.gfw</groupId>
            <artifactId>terasoluna-gfw-recommended-dependencies</artifactId>
            <type>pom</type>
        </dependency>
        <!-- == End TERASOLUNA == -->
    </dependencies>
</project>
```

Creating webservice project

webservice project configuration is explained.

```
artifactId-webservice
    ├- pom.xml  ... (1)
```

Sr. No.	Description
(1)	A POM (Project Object Model) file which defines webservice module configuration. Following are defined in this file. • Definition of plug-ins for dependent libraries and builds • Definition for creating a jar file

`pom.xml` is as shown in the image below. It must be edited when required.

Actually, the values specified while creating a blank project must be configured for “artifactId” and “groupId”.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/
↪2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0␣
↪http://maven.apache.org/maven-v4_0_0.xsd">

    <modelVersion>4.0.0</modelVersion>
    <artifactId>artifactId-webservice</artifactId>
    <packaging>jar</packaging>
    <parent>
        <groupId>groupId</groupId>
        <artifactId>artifactId</artifactId>
        <version>1.0.0-SNAPSHOT</version>
        <relativePath>../pom.xml</relativePath>
    </parent>
    <dependencies>
        <dependency>
            <groupId>${project.groupId}</groupId>
            <artifactId>artifactId-model</artifactId>
        </dependency>
        <!-- == Begin TERASOLUNA == -->
        <dependency>
            <groupId>org.terasoluna.gfw</groupId>
            <artifactId>terasoluna-gfw-common-dependencies</artifactId>
            <type>pom</type>
        </dependency>
        <dependency>
            <groupId>org.terasoluna.gfw</groupId>
            <artifactId>terasoluna-gfw-jodatime-dependencies</artifactId>
            <type>pom</type>
        </dependency>
        <dependency>
            <groupId>org.terasoluna.gfw</groupId>
            <artifactId>terasoluna-gfw-security-core-dependencies</artifactId>
            <type>pom</type>
        </dependency>

        <dependency>
            <groupId>org.terasoluna.gfw</groupId>
            <artifactId>terasoluna-gfw-recommended-dependencies</artifactId>
            <type>pom</type>
        </dependency>
        <!-- == End TERASOLUNA == -->
    </dependencies>
</project>
```

Package configuration of SOAP server

Recommended configuration while creating SOAP server is explained.

Following configuration is obtained if the project is added in conformance with the guidelines.

Project name	Description
[server projectName]-domain	Project which stores class and configuration file related to domain layer of SOAP server
[server projectName]-web	Project which stores class and configuration file related to application layer of SOAP server
[server projectName]-env	Project which stores files dependent on the environment of SOAP server
[server projectName]-model	Project which stores the class to be shared with the client and used while executing Web service, from the classes related to domain layer of SOAP server
[server projectName]-webservice	Project which stores interface of Web service offered by SOAP server

[server projectName]-domain

Following is added to `pom.xml` for adding dependency of [server projectName]-model.

```
<dependency>
    <groupId>${project.groupId}</groupId>
    <artifactId>artifactId-model</artifactId>
</dependency>
```

Refer "*Project structure* of *Application Layering* " since package configuration besides these is not different from

the usual domain project.

[server projectName]-web

Following is added to `pom.xml` for adding dependency of [server projectName]-webservice.

```
<dependency>
    <groupId>${project.groupId}</groupId>
    <artifactId>artifactId-webservice</artifactId>
</dependency>
```

Note: How to resolve a dependency

It is not necessary to define dependency of [server projectName]-model because a transitive dependency is added since dependency to [server projectName]-model is defined from [server projectName]-webservice.

Recommended configuration for [server projectName]-web project is shown below.

```
[server projectName]-web
  └src
      └main
          ├java
          |  └com
          |      └example
          |          ├app...(1)
          |          └ws...(2)
          |            ├exception...(3)
          |            |  └WsExceptionHandler.java
          |            ├abc
          |            |  └AbcWebServiceImpl.java
          |            └def
          |               └DefWebServiceImpl.java
          ├resources
          |  ├META-INF
          |  |  └spring
          |  |     ├applicationContext.xml...(4)
          |  |     ├application.properties...(5)
```

```
|  |      ├spring-mvc.xml ...(6)
|  |      ├spring-security.xml...(7)
|  |      └[server projectName]-ws.xml...(8)
|  └i18n
|      └application-messages.properties...(9)
└webapp
    ├resources...(10)
    └WEB-INF
        ├views ...(11)
        └web.xml...(12)
```

Sr. No.	Description
(1)	Package which stores configuration elements of application layer. It may be deleted if only Web service is required to be created.
(2)	Package which stores related class of Web service.
(3)	Package which stores exception handler etc of Web service.
(4)	Defines a bean related to overall application.
(5)	Define a property to be used in the application.
(6)	Define a Bean for configuring Spring MVC. It may be deleted if only Web service is required to be created.
(7)	Define a Bean for configuring Spring Security.
(8)	Define a Bean for Web service.
(9)	Define a message (internationalization) for screen display.

次のページに続く

Table. 5.18 – 前のページからの続き

Sr. No.	Description
(10)	Stores static resources (css, js, image etc). It may be deleted if only Web service is required to be created.
(11)	Stores View (jsp). It may be deleted if only Web service is required to be created.
(12)	Define Servlet deployment.

Note: Files not required for SOAP server

When only Web service is to be created in SOAP server, Spring MVC configuration file existing in the blank project is not required, hence can be deleted.

[server projectName]-env

Since [server projectName]-env does not differ from normal env project, refer "*Project structure* of *Application Layering*".

[server projectName]-model

Recommended project configuration of [server projectName]-model is shown below.

```
[server projectName]-model
  └src
     └main
        └java
           └com
              └example
                 └domain ...(1)
                    └model ...(2)
                       ├Xxx.java
                       ├Yyy.java
                       └Zzz.java
```

Sr. No.	Description
(1)	Package which stores configuration elements of domain layer.
(2)	Package which stores the class to be used while implementing Web service in the Domain Object.

[server projectName]-webservice

Recommended project configuration of [server projectName]-webservice is shown below.

```
[server projectName]-webservice
  └src
      └main
          └java
              └com
                  └example
                      └ws...(1)
                        ├webfault...(2)
                        ├abc
                        |  └AbcWebService.java
                        └def
                            └DefWebService.java
```

Sr. No.	Description
(1)	Package which stores Web service interface.
(2)	Package which stores webfault of Web service.

Package configuration of client

Recommended configuration while creating a client is explained.

Project when provided from SOAP Server in accordance with the guidelines is of following configuration.

Project name	Description
[client projectName]-domain	Project which stores class and configuration file related to domain layer of client
[client projectName]-web	Project which stores class and configuration file related to application layer of client
[client projectName]-env	Project which stores files dependent on the client environment

Note: For [server projectName]-model and [server projectName]-webservice, refer " *Package configuration of SOAP server*" described earlier.

[client projectName]-domain

Following is added to `pom.xml` for adding dependency of [server projectName]-webservice offered from SOAP server.

```
<dependency>
    <groupId>${project.groupId}</groupId>
    <artifactId>artifactId-webservice</artifactId>
</dependency>
```

Note: how to resolve dependency

Similar to [server projectName]-web, it is not necessary to define dependency of [server projectName]-model in `pom.xml` because a transitive dependency is added since dependency relation to [server projectName]-model is defined from [server projectName]-webservice.

Since package configuration other than this is not different from the usual domain project, refer "*Project structure* of *Application Layering* ".

[client projectName]-web

Since [client projectName]-web is not different from the usual web project, refer "*Project structure* of *Application Layering*".

[client projectName]-env

Recommended project configuration of [client projectName]-env project is shown below.

```
[projectName]-env
  ├configs ...(1)
  |   └[envName] ...(2)
  |       └resources ...(3)
  └src
      └main
          └resources ...(4)
             ├META-INF
             |  └spring
             |      ├[projectName]-env.xml ...(5)
             |      └[projectName]-infra.properties ...(6)
             ├dozer.properties
             ├log4jdbc.properties
             └logback.xml ...(7)
```

Sr. No.	Description
(1)	Directory for managing environment dependent files of overall environment.
(2)	Directory for managing environment dependent files for each environment. Specify a name which identifies the environment, as a directory name.
(3)	Directory for managing configuration files for each environment. Subdirectory configuration and configuration files to be managed are same as (4).
(4)	Directory for managing configuration files for local development environment.
(5)	Define a Bean for local development environment. Specify a proxy class of Web service in this file.
(6)	Define a property for local development environment. Specify the value that can be changed for each environment like URL of WSDL.
(7)	Define log output for local development environment.

wsimport

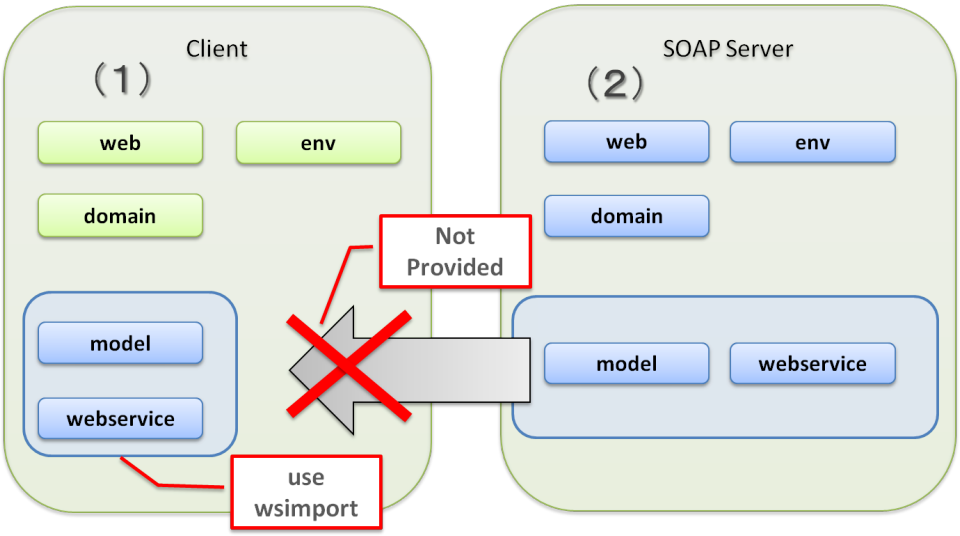
wsimport is a command line tool that is included in Java SE.

It outputs a Java class (source as well depending on the options) which can call Web service by reading WSDL file.

Using wsimport

In this guideline, wsimport has been recommended to be used in the cases given in the image below..

Web service can be implemented by using wsimport while creating a client even when Domain Object or Web service interface which are used in SOAP server cannot be used.



How to use wsimport

It is stored in the bin folder of JDK and can be used only by going through the path.

When the command is executed on the command line as given below, the source file is created in the current directory.

```
# (1)
wsimport -keep -p [Package name of the source to be output] -s [Location which␣
↪stores source to be output] [URL of wsdl]
```

Sr. No.	Description
(1)	Specify URL of WSDL as an argument of wsimport. Following are used as an option. • -keep Source is output as well. • -p Specify the package of the source to be output. • -s Specify the location which stores the source to be output. For other options, refer Java Platform, Standard Edition Tools Reference -Web Services(wsimport)-.

Note: wsimport outputs only class file as the default behaviour. No action is required only for the moving operation, however when a debug operation is to be carried out, it is recommended to apply 'keep' option and store source as well.

For example, the commands are as below.

```
wsimport -keep -p com.example.ws.todo -s c:/tmp http://AAA.BBB.CCC.DDD:XXXX/soap-
↪web/ws/TodoWebService?wsdl
```

Although the source created is dependent on Web service to be published, Java class used in the guideline is output.

- Web service interface (`TodoWebService.java` in the source example)
- Domain Object (`Todo.java` in the source example)

When the class generated by wsimport is to be used in only one client project, it should be placed in the domain project.

Although generated class belongs to infrastructure layer(*Integration System Connector*), it can also be included in the normal domain project as shown in Note of *Project structure*.

When the generated class is to be used for multiple clients, it is preferable to create model project and webservice project based on *Project configuration is changed for SOAP server.*and use by referring the same from respective clients.

Note: Java class to be output is also output in the cases other than above. A client can be created only by using source that has been output. However, as a policy in this guideline, since the client uses `org.springframework.remoting.jaxws.JaxWsPortProxyFactoryBean`, it is recommended to not to use another Java class.

Web service development on Tomcat

Although JAS-VX on Java EE server is described in the guideline, JAX-WS is not implemented in case of Tomcat.

Therefore, when SOAP server is Tomcat, Apache CXF is used as an implementation product of JAX-WS. It is necessary to use `CXFServlet` by changing the configuration.

When Apache CXF is used, a couple of implementation methods of WebService class exist as given below.

1. A method wherein Web service implementation class is described in POJO
2. A method wherein Web service implementation class is created by inheriting `SpringBeanAutowiringSupport`. (method that has been described so far)

In case of 1, since POJO is used as a Web service implementation class, unit testing can be easily carried out. However, this method may not work well for AP servers other than Tomcat. Therefore, implementation by using second method is described in the guideline instead of using first method. However, when only Tomcat is used, using the first method is recommended due to a number of advantages.

In case of 2, implementation can be done similar to other AP servers. Operations are carried out on Java EE server, however, this method is used when Tomcat must be mandatorily used during development.

Configuration while using CXFServlet

Library configuration is described in `pom.xml` to use `CXFServlet`.

```
<!-- (1) -->
<dependency>
```

```
    <groupId>org.apache.cxf</groupId>
    <artifactId>cxf-rt-frontend-jaxws</artifactId>
    <version>3.1.4</version>
</dependency>
<dependency>
    <groupId>org.apache.cxf</groupId>
    <artifactId>cxf-rt-transports-http</artifactId>
    <version>3.1.4</version>
</dependency>
```

Sr. No.	Description
(1)	Add dependency to Apache CXF library for using `CXFServlet`.

Next, `CXFServlet`which receives SOAP Web Service in `web.xml` is defined.

```
<!-- (1) -->
<servlet>
    <servlet-name>cxfServlet</servlet-name>
    <servlet-class>org.apache.cxf.transport.servlet.CXFServlet</servlet-class>
    <init-param>
        <param-name>config-location</param-name>
        <param-value>classpath:/META-INF/spring/cxf-servlet.xml</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
</servlet>
<!-- (2) -->
<servlet-mapping>
    <servlet-name>cxfServlet</servlet-name>
    <url-pattern>/ws/*</url-pattern>
</servlet-mapping>
```

Sr. No.	Description
(1)	Define servlet for `org.apache.cxf.transport.servlet.CXFServlet`. Specify path of `cxf-servlet.xml`to be described later, in `config-location`.
(2)	Define mapping for the servlet that has been defined. In this case, Web service is created under Context name/ws.

Configuration required in POJO method

Specify Web service implementation class as an endpoint.

[server projectName]-web/src/main/resources/META-INF/spring/cxf-servlet.xml

```
<beans xmlns="http://www.springframework.org/schema/beans" xmlns:xsi="http://www.
↪w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:jaxws="http://cxf.apache.org/jaxws" xmlns:soap="http://cxf.apache.org/
↪bindings/soap"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
         http://www.springframework.org/schema/beans/spring-beans.xsd
         http://www.springframework.org/schema/context
         http://www.springframework.org/schema/context/spring-context.xsd
         http://cxf.apache.org/jaxws
         http://cxf.apache.org/schemas/jaxws.xsd
         http://cxf.apache.org/bindings/soap
         http://cxf.apache.org/schemas/configuration/soap.xsd">

    <!-- (1) -->
    <jaxws:endpoint id="todoWebEndpoint" implementor="#todoWebServiceImpl"
        address="/TodoWebService" />

</beans>
```

Sr. No.	Description
(1)	Define the endpoint to be published. Specify bean name (“bean name” format) of Web service class which is registered in DI container, in the `implementor` attribute. Specify address which publishes Web service, in `address` attribute. Address describes only the path of end point to be published. For attribute details, refer Apache CXF JAX-WS Configuration.

Create `TodoWebServiceImpl` as POJO.

[server projectName]-web/src/main/java/com/example/ws/todo/TodoWebServiceImpl.java

```
package com.example.ws.todo;

import java.util.List;

import javax.inject.Inject;
import javax.jws.HandlerChain;
import javax.jws.WebService;
import javax.xml.ws.BindingType;
import javax.xml.ws.soap.SOAPBinding;

import org.springframework.web.context.support.SpringBeanAutowiringSupport;

import org.springframework.stereotype.Component;

import com.example.domain.model.Todo;
import com.example.domain.service.TodoService;
import com.example.ws.webfault.WebFaultException;
import com.example.ws.exception.WsExceptionHandler;
import com.example.ws.todo.TodoWebService;

// (1)
@Component
@WebService(
      portName = "TodoWebPort",
      serviceName = "TodoWebService",
      targetNamespace = "http://example.com/todo",
      endpointInterface = "com.example.ws.todo.TodoWebService")
@BindingType(SOAPBinding.SOAP12HTTP_BINDING)
// (2)
```

```
public class TodoWebServiceImpl implements TodoWebService {

    // omitted

}
```

Sr. No.	Description
(1)	Apply `@Component` and register to DI container.
(2)	Create as POJO since registration to DI container is possible by component scan. Basically, inheriting `org.springframework.web.context.support.SpringBeanAutowiringSupport` is not necessary.

Configuration required for the method that inherits SpringBeanAutowiringSupport

Class name and address acting as SOAP end points are defined in the Bean definition file for CXFServlet.

[server projectName]-web/src/main/resources/META-INF/spring/cxf-servlet.xml

```
<beans xmlns="http://www.springframework.org/schema/beans" xmlns:xsi="http://www.
↪w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:jaxws="http://cxf.apache.org/jaxws" xmlns:soap="http://cxf.apache.org/
↪bindings/soap"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
         http://www.springframework.org/schema/beans/spring-beans.xsd
         http://www.springframework.org/schema/context
         http://www.springframework.org/schema/context/spring-context.xsd
         http://cxf.apache.org/jaxws
         http://cxf.apache.org/schemas/jaxws.xsd
         http://cxf.apache.org/bindings/soap
         http://cxf.apache.org/schemas/configuration/soap.xsd">
    <!-- (1) -->
    <jaxws:endpoint id="todoWebEndpoint" implementor="com.example.ws.todo.
↪TodoWebServiceImpl"
        address="/TodoWebService" />
```

```
</beans>
```

Sr. No.	Description
(1)	Define endpoint to be published. Specify implementation class of Web service to be published in `implementor` attribute. Specify address which publishes Web service in `address` attribute. Address describes only the path of end point to be published. For attribute details, refer Apache CXF JAX-WS Configuration.

6

Data Access

The architecture adopted in this guideline is explained in detail.

6.1 Database Access (Common)

Todo

TBD

The following topics in this chapter are currently under study.

- About multiple datasources

  For details, *About multiple datasources of Overview* and *Settings for using multiple datasources of How to extends*.

- About handling of unique constraint errors and pessimistic exclusion errors when using JPA

  For details, refer to *Overview - About exception handling*.

6.1.1 Overview

This chapter explains the method of accessing the data stored in RDBMS.

For the O/R Mapper dependent part, refer to

- *Database Access (MyBatis3)*
- *Database Access (JPA)*

About JDBC DataSource

RDBMS can be accessed from the application by referring to JDBC datasource.

JDBC datasource can be used to exclude settings such as JDBC driver load, connection information (URL, user, password etc.) from application.

Therefore, RDBMS to be used and environment in which the application is to be deployed need not be taken into account.

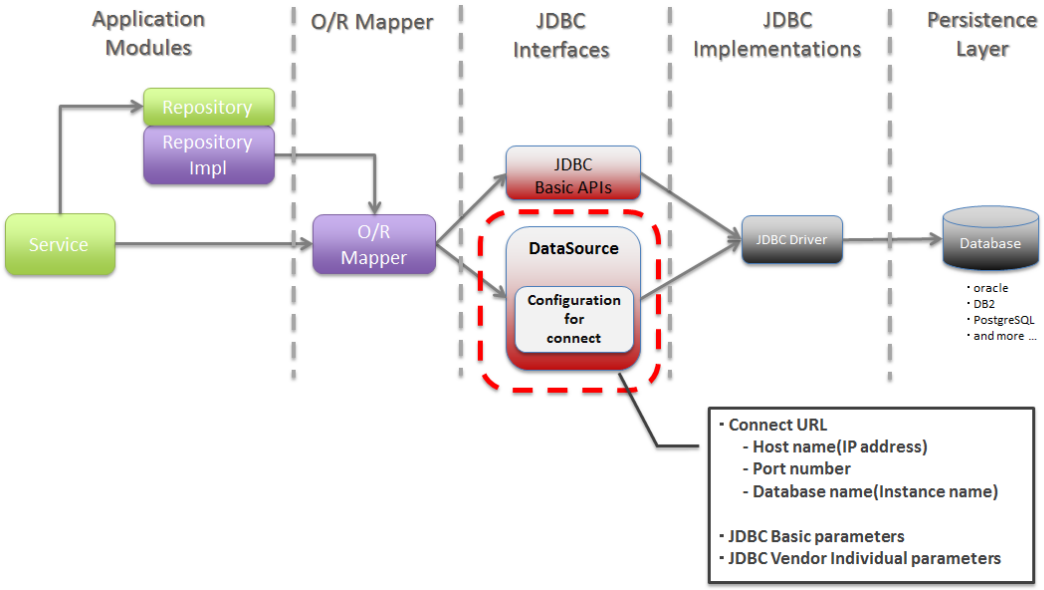


Figure.6.1 **Picture - About JDBC DataSource**

JDBC datasource is implemented from Application Server, OSS library, Third-Party library, Spring Framework etc.; hence it is necessary to select the datasource based on project requirements and deployment environment.

The typical datasources are introduced below.

- *JDBC datasource provided by Application Server*
- *JDBC datasource provided by OSS/Third-Party library*
- *JDBC datasource provided by Spring Framework*

JDBC datasource provided by Application Server

When datasource is to be used in Web application, normally JDBC datasource provided by Application Server is used.

JDBC datasource of Application Server provides functionalities required in web application such as Connection Pooling as standard functionalities.

Table.6.1 **Datasources provided by Application Server**

Sr. No.	Application Server	Reference page
1.	Apache Tomcat 8.5	Refer to Apache Tomcat 8.5 User Guide(The Tomcat JDBC Connection Pool). Refer to Apache Tomcat 8.5 User Guide(JNDI Datasource HOW-TO) (Apache Commons DBCP 2).
2.	Apache Tomcat 8.0	Refer to Apache Tomcat 8.0 User Guide(The Tomcat JDBC Connection Pool). Refer to Apache Tomcat 8.0 User Guide(JNDI Datasource HOW-TO)(Apache Commons DBCP 2).
3.	Apache Tomcat 7	Refer to Apache Tomcat 7 User Guide (The Tomcat JDBC Connection Pool). Refer to Apache Tomcat 7 User Guide (JNDI Datasource HOW-TO) (Apache Commons DBCP).
4.	Oracle WebLogic Server 12c	Refer to Oracle WebLogic Server Product Documentation.
5.	IBM WebSphere Application Server Version 9.0	Refer to WebSphere Application Server Online information center.
6.	JBoss Enterprise Application Platform 7.0	Refer JBoss Enterprise Application Platform 7.0 Product Documentation.
7.	JBoss Enterprise Application Platform 6.4	Refer JBoss Enterprise Application Platform 6.4 Product Documentation.

JDBC datasource provided by OSS/Third-Party library

When JDBC datasource of Application Server is not used, JDBC datasource of OSS/Third-Party library should be used.

This guideline introduces only Apache Commons DBCP; however other libraries can also be used.

Table.6.2 **JDBC datasource provided by OSS/Third-Party library**

Sr. No.	Library name	Description
1.	Apache Commons DBCP	Refer to Apache Commons DBCP.

JDBC datasource provided by Spring Framework

Implementation class of JDBC datasource of Spring Framework cannot be used as datasource of Web application since it does not provide connection pooling.

In Spring Framework, implementation class and adapter class of JDBC datasource are provided; however they are introduced as *JDBC datasource classes provided by Spring Framework* of Appendix, since usage is restricted.

About transaction management

When transactions are to be stored using Spring Framework functionality, PlatformTransactionManager needs to be selected based on project requirements and deployment environment.

For details, refer to *Settings for using transaction management* of *Domain Layer Implementation*.

About declaration of transaction boundary/attribute

Transaction boundary and transaction attributes should be declared by specifying `@Transactional` annotation in Service.

For details, refer to *Regarding transaction management* of *Domain Layer Implementation* .

About exclusion control of data

When updating data, it is necessary to execute exclusion control to ensure data consistency and integrity.

For details on exclusion control of data, refer to *Exclusive Control*.

About exception handling

In Spring Framework, a function is provided to convert JDBC exception (`java.sql.SQLException`) and O/R Mapper specific exception to data access exception (subclass of (`org.springframework.dao.DataAccessException`) provided by Spring Framework.

For the class which is converted to data access exception of Spring Framework, refer to *Classes provided by Spring Framework for converting to data access exception* of Appendix.

The converted data access exception need not be handled in application code; however, some errors (such as unique constraint violation, exclusion error etc.) need to be handled as per the requirements.

When handling data access exception, exception of subclass notifying error details should be caught instead of `DataAccessException`.

Typical subclasses which are likely to be handled in application code are as follows:

Table.6.3 **Subclasses of DB access exception, which are likely to be handled**

Sr. No.	Class name	Description
1.	org.springframework.dao. DuplicateKeyException	Exception that occurs in case of unique constraint violation.
2.	org.springframework.dao. OptimisticLockingFailureException	Exception that occurs in case of optimistic locking failure. It occurs when same data is updated with different logic. This exception occurs when JPA is used as O/R Mapper. MyBatis does not have optimistic locking function; hence this exception does not occur from O/R Mapper.
3.	org.springframework.dao. PessimisticLockingFailureException	Exception that occurs in case of pessimistic locking failure. It occurs when same data is locked with different logic and the lock is not released even after “waiting for unlocking” timeout period has elapsed.

Note: When optimistic locking is to be implemented using MyBatis in O/R Mapper, it should be implemented as Service or Repository process.

As a method of notifying the optimistic locking failure to Controller, this guideline recommends generation of `OptimisticLockingFailureException` and exception of its child class.

This is to make implementation of application layer (implementation of Controller) independent of O/R Mapper to be used.

Todo

It has been recently found that using JPA (Hibernate) results in occurrence of unexpected errors.

- In case of unique constraint violation, `org.springframework.dao.DataIntegrityViolationException` occurs and not `DuplicateKeyException`.

See the example below for handling unique constraint violation as business exception.

```
try {
    accountRepository.saveAndFlash(account);
} catch(DuplicateKeyException e) { // (1)
    throw new BusinessException(ResultMessages.error().add("e.xx.xx.0002"),␣
↪e); // (2)
}
```

Sr. No.	Description
(1)	Exception (DuplicateKeyException) that occurs in case of unique constraint violation is caught.
(2)	Business exception indicating that there is duplicate data is thrown. When exception is caught, make sure to specify the cause of exception (`e`) in business exception.

About multiple datasources

Multiple datasources may be required depending on the application.

Typical cases wherein multiple datasources are required, are shown below.

Table.6.4 **Typical case where multiple datasources are required**

Sr. No.	Case	Example	Feature
1.	When database and schema are divided according to data (tables).	When group of tables maintaining customer information and group of tables maintaining invoice information are stored in separate database and schema.	The data to be handled in the process is fixed; hence the datasource to be used can be defined statically.
2.	When database and schema to be used are divided according to users (login users).	When database and schema are divided according to users (Multitenant etc.).	The datasource to be used differs depending on users; hence the datasource to be used dynamically can be defined.

Todo

TBD

The following details will be added in future.

- Conceptual diagram

About common library classes

Common library provides classes that carry out following processes.
For more details about common library, refer to links given below.

- *Escaping during LIKE search*
- *About Sequencer*

6.1.2 How to use

Datasource settings

Settings when using DataSource defined in Application Server

When using datasource defined in Application Server, it is necessary to perform settings in Bean definition file to register the object fetched through JNDI as a bean.

Settings when PostgreSQL is used as database and Tomcat7 is used as Application Server are given below.

- `xxx-context.xml` (Tomcat config file)

```
<!-- (1) -->
<Resource
   type="javax.sql.DataSource"
   name="jdbc/SampleDataSource"
   driverClassName="org.postgresql.Driver"
   url="jdbc:postgresql://localhost:5432/terasoluna"
   username="postgres"
   password="postgres"
   defaultAutoCommit="false"
   /> <!-- (2) -->
```

- `xxx-env.xml`

```
<jee:jndi-lookup id="dataSource" jndi-name="jdbc/SampleDataSource" /> <!--␣
↪(3) -->
```

Sr. No.	Attribute name	Description
(1)	-	Define datasource.
	type	Specify resource type. Specify `javax.sql.DataSource`.
	name	Specify resource name. The name specified here is JNDI name.
	driverClassName	Specify JDBC driver class. In the example, JDBC driver class provided by PostgreSQL is specified.
	url	Specify URL. [Needs to be changed as per environment]
	username	Specify user name. [Needs to be changed as per environment]
	password	Specify password of user. [Needs to be changed as per environment]
	defaultAutoCommit	Specify default value of auto commit flag. Specify 'false'. It is forcibly set to 'false' when it is under Transaction Management.
(2)	-	In case of Tomcat7, tomcat-jdbc-pool is used if factory attribute is omitted. For more details about settings, refer to Attributes of The Tomcat JDBC Connection Pool.
(3)	-	Specify JNDI name of datasource. In case of Tomcat, specify the value specified in resource name "(1)-name" at the time of defining datasource.

Settings when using DataSource for which Bean is defined

When using datasource of OSS/Third-Party library or JDBC datasource of Spring Framework without using the datasource provided by Application Server,
bean for DataSource class needs to be defined in Bean definition file.

Settings when PostgreSQL is used as database and Apache Commons DBCP is used as datasource are given below.

- xxx-env.xml

```
<bean id="dataSource" class="org.apache.commons.dbcp2.BasicDataSource"
    destroy-method="close">                                          <!--␣
↪(1) (8) -->
    <property name="driverClassName" value="org.postgresql.Driver" /> <!--␣
↪(2) -->
    <property name="url" value="jdbc:postgresql://localhost:5432/terasoluna" /
↪> <!-- (3) -->
    <property name="username" value="postgres" />                    <!--␣
↪(4) -->
    <property name="password" value="postgres" />                    <!--␣
↪(5) -->
    <property name="defaultAutoCommit" value="false"/>               <!-- (6)␣
↪-->
    <!-- (7) -->
</bean>
```

Sr. No.	Description
(1)	Specify implementation class of datasource. In the example, datasource class (`org.apache.commons.dbcp2.BasicDataSource`) provided by Apache Commons DBCP is specified.
(2)	Specify JDBC driver class. In the example, JDBC driver class provided by PostgreSQL is specified.
(3)	Specify URL. [Needs to be changed as per environment]
(4)	Specify user name. [Needs to be changed as per environment]
(5)	Specify password of user. [Needs to be changed as per environment]
(6)	Specify default value of auto commit flag. Specify 'false'. It is forcibly set to 'false' when it is under Transaction Management.
(7)	In BasicDataSource, configuration values common in JDBC, JDBC driver specific properties values, connection pooling configuration values can be specified other than the values mentioned above. For more details about settings, refer to DBCP Configuration.
(8)	In the example, values are specified directly; however, for fields where configuration values change with the environment, actual configuration values should be specified in properties file using Placeholder(${...}). For Placeholder, refer to `PropertyPlaceholderConfigurer` of Spring Reference Document.

Settings to enable transaction management

For basic settings to enable transaction management, refer to *Settings for using transaction management* of *Domain Layer Implementation*.

For PlatformTransactionManager, the class to be used changes depending on the O/R Mapper used; hence for detailed settings, refer to:

- *Database Access (MyBatis3)*
- *Database Access (JPA)*

JDBC debug log settings

When more detailed information than the log output using O/R Mapper(MyBatis, Hibernate) is required, the information output using log4jdbc(log4jdbc-remix) can be used.

For details on log4jdbc, refer to log4jdbc project page.

For details on log4jdbc-remix, refer to log4jdbc-remix project page.

> **Warning: When Log4jdbcProxyDataSource offered by log4jdbc-remix is used, substantial overheads are likely to occur even if the log level is set in the configuration other than debug. Therefore, it is recommended to use this setting for debugging, and connect to database without passing through Log4jdbcProxyDataSource while its release during performance test enviroment and commercial environment.**

Settings related to datasource provided by log4jdbc

- `xxx-env.xml`

```
<jee:jndi-lookup id="dataSourceSpied" jndi-name="jdbc/SampleDataSource" /> <!-
↪- (1) -->

<bean id="dataSource" class="net.sf.log4jdbc.Log4jdbcProxyDataSource"> <!--␣
↪(2) -->
    <constructor-arg ref="dataSourceSpied" /> <!-- (3) -->
</bean>
```

Sr. No.	Description
(1)	Define actual datasource. In the example, the datasource fetched through JNDI from Application Server is being used.
(2)	Specify `net.sf.log4jdbc.Log4jdbcProxyDataSource` provided by log4jdbc.
(3)	In constructor, specify bean which is an actual datasource.

Warning: When the application is to be released in performance test environment or production environment, Log4jdbcProxyDataSource should not be used as datasource.

Specifically, exclude settings of (2) and (3) and change bean name of `"dataSourceSpied"` to `"dataSource"`.

log4jdbc logger settings

- `logback.xml`

```
<!-- (1) -->
<logger name="jdbc.sqltiming">
    <level value="debug" />
</logger>

<!-- (2) -->
<logger name="jdbc.sqlonly">
    <level value="warn" />
</logger>

<!-- (3) -->
<logger name="jdbc.audit">
    <level value="warn" />
</logger>

<!-- (4) -->
<logger name="jdbc.connection">
    <level value="warn" />
</logger>

<!-- (5) -->
<logger name="jdbc.resultset">
    <level value="warn" />
```

```
</logger>

<!-- (6) -->
<logger name="jdbc.resultsettable">
    <level value="debug" />
</logger>
```

Sr. No.	Description
(1)	Logger to output SQL execution time and SQL statement wherein the value is set in bind variable. Since this SQL contains values for bind variables, it can be executed using DB access tool.
(2)	Logger to output SQL statement wherein the value is set in bind variable. The difference with (1) is that SQL execution time is not output.
(3)	Logger to exclude ResultSet interface, call methods of JDBC interface and to output arguments and return values. This log is useful for analyzing the JDBC related issues; however volume of the output log is large.
(4)	Logger to output connected/disconnected events and number of connections in use. This log is useful for analyzing connection leak, but it need not be output unless there is connection leak issue.
(5)	Logger to call methods of ResultSet interface and output arguments and return values. This log is useful during analysis when actual result differs from expected result; however volume of the output log is large.
(6)	Logger to output the contents of ResultSet by converting them into a format so that they can be easily verified. This log is useful during analysis when actual result differs from expected result; however volume of the output log is large.

Warning: Large amount of log is output depending on the type of logger; hence only the required logger should be defined or output.

In the above sample, log level for loggers which output very useful logs during development, is set to `"debug"`. As for other loggers, the log level needs to be set to `"debug"` whenever required.

When the application is to be released in performance test environment or production environment, log using log4jdbc logger should not be output at the time of normal end of process.

Typically log level should be set to `"warn"`.

Settings of log4jdbc option

Default operations of log4jdbc can be customized by placing properties file `log4jdbc.properties`under class path.

- `log4jdbc.properties`

```
# (1)
log4jdbc.dump.sql.maxlinelength=0
# (2)
```

Sr. No.	Description
(1)	Specify word-wrap setting for SQL statement. If '0' is specified, SQL statement is not wrapped.
(2)	For option details, refer to log4jdbc project page -Options-.

6.1.3 How to extend

Settings for using multiple datasources

Todo

TBD

Following details will be added in future.

- Transaction management method may change depending on the processing pattern (like Update for multiple datasources, Update for a single datasource, Only for reference, No concurrent access etc.), hence breakdown is planned focusing on that area.

Settings to switch the datasource dynamically

In order to define multiple datasources and then to switch them dynamically, it is necessary to create a class that inherits `org.springframework.jdbc.datasource.lookup.AbstractRoutingDataSource` and implement the conditions by which datasource is switched.

This is to be implemented by mapping the key which is a return value of `determineCurrentLookupKey` method with the datasource. For selecting the key, usually context information like authenticated user information, time and locale etc. is to be used.

Implementation of AbstractRoutingDataSource

The datasource can be switched dynamically by using the `DataSource` which is created by extending `AbstractRoutingDataSource`in a same way as the normal datasource.

The example of switching the datasource based on time is given below.

- Example of implementing a class that inherits `AbstractRoutingDataSource`

```
package com.examples.infra.datasource;

import javax.inject.Inject;

import org.joda.time.DateTime;
import org.springframework.jdbc.datasource.lookup.AbstractRoutingDataSource;
import org.terasoluna.gfw.common.date.jodatime.JodaTimeDateFactory;

public class RoutingDataSource extends AbstractRoutingDataSource { // (1)

    @Inject
    JodaTimeDateFactory dateFactory; // (2)

    @Override
    protected Object determineCurrentLookupKey() { // (3)

        DateTime dateTime = dateFactory.newDateTime();
        int hour = dateTime.getHourOfDay();

        if (7 <= hour && hour <= 23) { // (4)
            return "OPEN"; // (5)
        } else {
```

```
            return "CLOSE";
        }
    }
}
```

Sr. No.	Description
(1)	Inherit `AbstractRoutingDataSource`.
(2)	Use `JodaTimeDateFactory`to fetch time. For details, refer to *System Date*.
(3)	Implement `determineCurrentLookupKey` method. The datasource to be used is defined by mapping the return value of this method and the `key` defined in `targetDataSources` of the bean definition file described later.
(4)	In the method, refer to the context information (here Time) and switch the key. Here the implementation should be in accordance with the business requirements. This sample is being implemented so that the time returns different keys as "From 7:00 to 23:59" and "From 0:00 to 6:59".
(5)	Return the `key` to be mapped with `targetDataSources` of the bean definition file described later.

Datasource definition

Define the `AbstractRoutingDataSource`extended class which was created, in bean definition file.

- `xxx-env.xml`

```
<bean id="dataSource"
    class="com.examples.infra.datasource.RoutingDataSource">  <!-- (1) -->
    <property name="targetDataSources">  <!-- (2) -->
        <map>
            <entry key="OPEN" value-ref="dataSourceOpen" />
            <entry key="CLOSE" value-ref="dataSourceClose" />
        </map>
    </property>
    <property name="defaultTargetDataSource" ref="dataSourceDefault" />  <!--␣
↪(3) -->
</bean>
```

Sr. No.	Description
(1)	Define a class that inherits `AbstractRoutingDataSource` created earlier.
(2)	Define the datasource to be used. As for `key`, define the value that can be returned using `determineCurrentLookupKey` method. In `value-ref` , specify the datasource to be used for each `key`. Define in accordance with the number of datasources to be switched based on *Datasource settings*.
(3)	This datasource is used, when `key` specified in `determineCurrentLookupKey` method does not exist in `targetDataSources`. In case of implementation example, default setting is not used; however, this time `defaultTargetDataSource` is being used for description purpose.

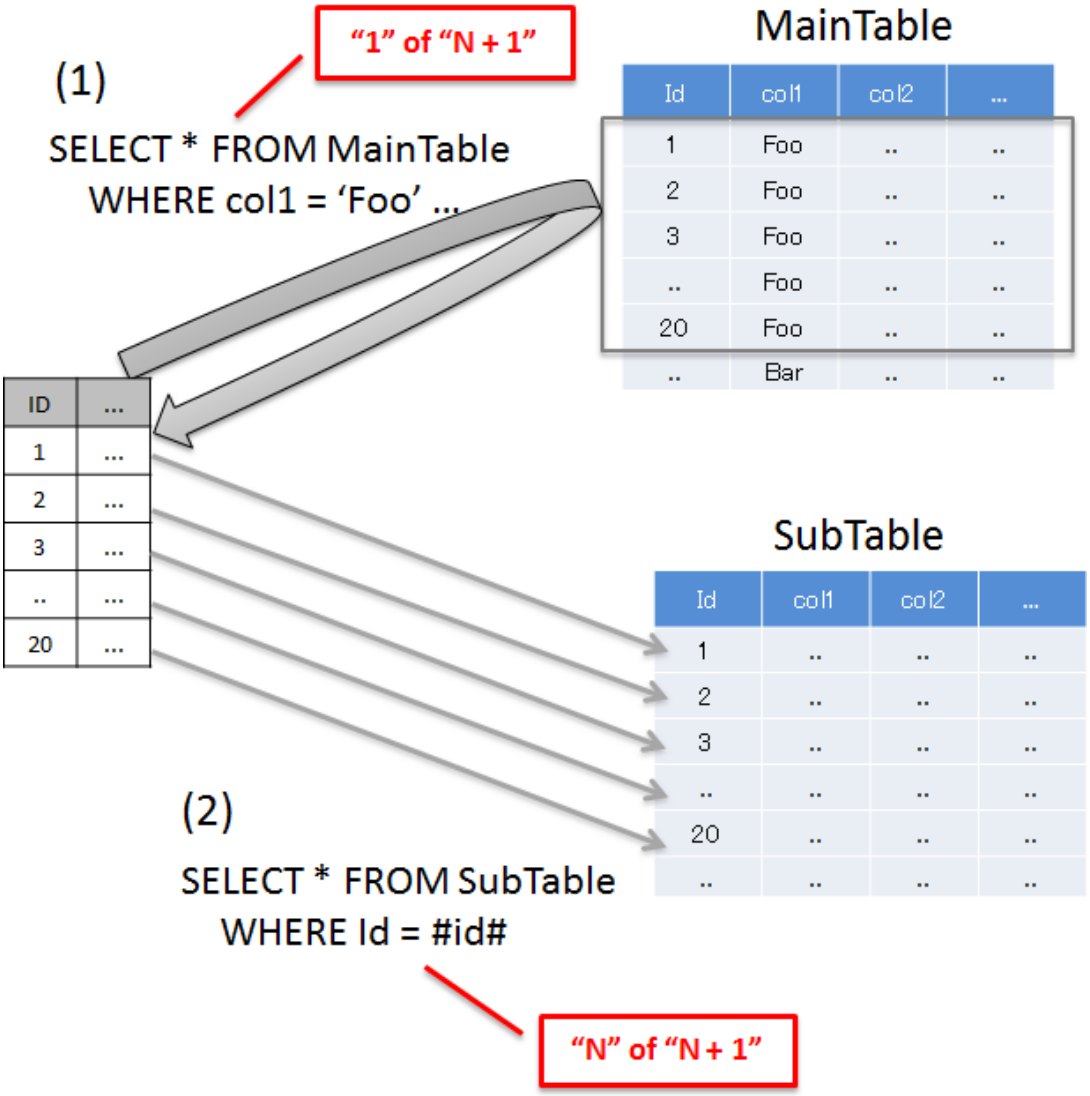
6.1.4 how to solve the problem

How to resolve N+1

N+1 occurs when more number of SQL statements need to be executed in accordance with the number of records to be fetched from the database. This problem causes high load on the database and deteriorates response time.

Details are given below.

Sr. No.	Description
(1)	Search the records matching the search conditions from MainTable. In the above example, col1 of MainTable fetches `'Foo'` records and the total records fetched are 20.
(2)	For each record searched in (1), related records are fetched from SubTable. In the above example, the id column of SubTable fetches the same records as the id column of records fetched in (1). **This SQL is executed for number of records fetched in (1).**



In the above example, **SQL is executed totally 21 times.**

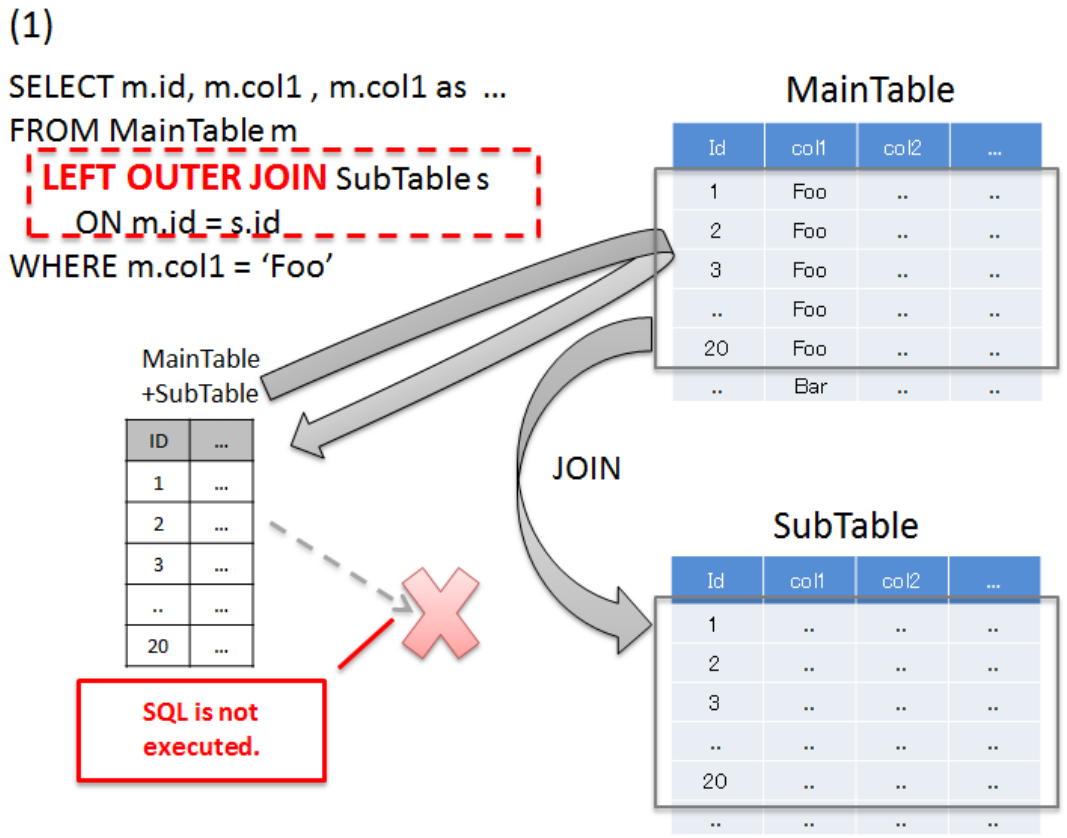
Supposing there are 3 SubTables, **SQL is executed totally 61 times; hence countermeasures are required.**

Typical example to resolve N+1 is given below.

Resolving N+1 using JOINs (Join Fetch)

By performing JOIN on SubTable and MainTable, records of MainTable and SubTable are fetched by executing SQL once.

When relation of MainTable and SubTable is 1:1, check whether N+1 can be resolved using this method.



Sr. No.	Description
(1)	When searching records matching the search conditions, the records are fetched in batch from MainTable and SubTable, by performing JOIN on SubTable. In the above example, col1 of MainTable collectively fetches `'Foo'` records and records of SubTable that match the id of the records matching with search conditions. When there are duplicate column names, it is necessary to assign alias name in order to identify the table to which that column belongs.

If JOIN (Join Fetch) is used, **all the required data can be fetched by executing SQL once.**

Note: When performing JOIN by JPQL

For example of performing JOIN using JPQL, refer to *JOIN FETCH*.

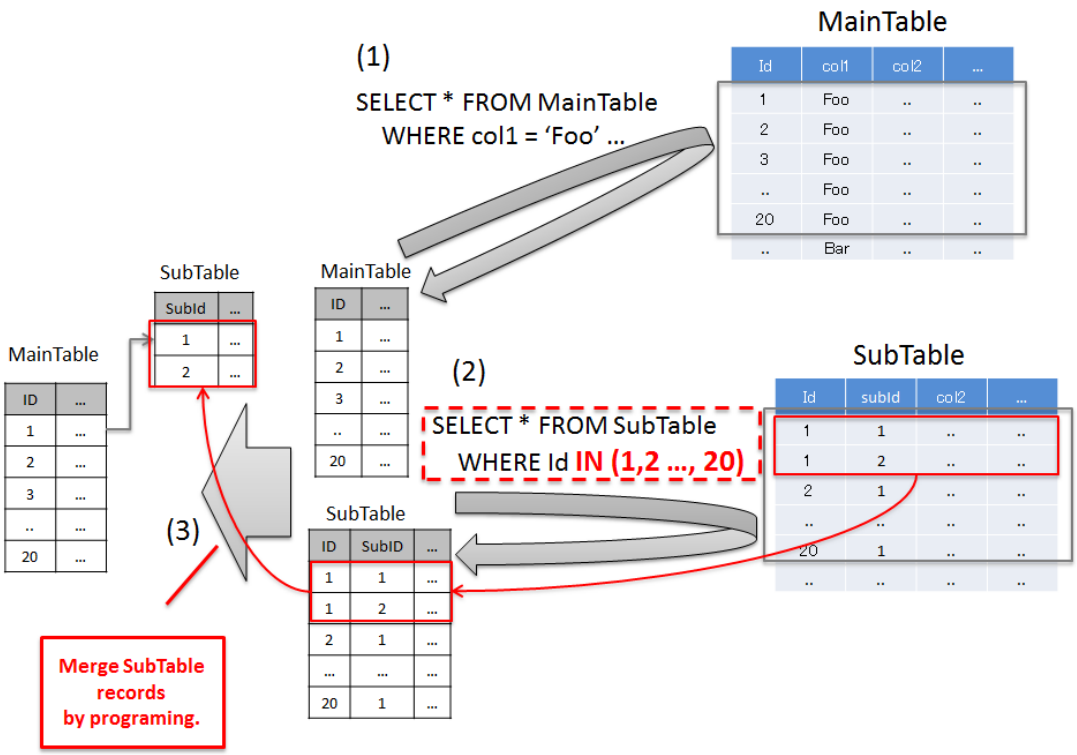
Warning: When relation with SubTable is 1:N, the problem can be resolved using JOIN (Join Fetch); however the following points should be noted.

- When JOIN is performed on records having 1:N relation, unnecessary data is fetched depending on the number of records in SubTable. For details, refer to *Notes during collective fetch*.
- When using JPA (Hibernate), if N portions in 1:N are multiple, then it is necessary to use `java.util.Set` instead of `java.util.List` as a collection type storage N portion.

Resolving N+1 by fetching related records in batch

There are cases where, it has proved better when the related records are fetched in batch for patterns with multiple 1:N relations etc.; and then sorted by programming.

When relation with SubTable is 1:N, analyze whether the problem can be resolved using this method.



Sr. No.	Description
(1)	Search the records matching the search conditions from MainTable. In the above example, col1 of MainTable fetches `'Foo'` records and the total records fetched are 20.
(2)	For each record searched in (1), related records are fetched from SubTable. Related records are not fetched one by one; but the records matching the foreign key of each record fetched in (1), are fetched in batch. In the above example, id column of SubTable collectively fetches same records as id column of records fetched in (1) using IN clause.
(3)	SubTable records fetched in (2) sorted and merged with records fetched in (1).

In the above example, **all the required data can be fetched by executing SQL twice.**
Even supposing there are 3 SubTables, **SQL needs to be executed totally 4 times.**

Note: This method has a special feature. It can fetch only the required data by optimizing SQL execution. It is necessary to sort SubTable records by programming; however when there are many SubTables or when number of N records in 1:N is more, there are cases wherein it is better to resolve the problem using this method.

6.1.5 Appendix

Escaping during LIKE search

When performing LIKE search, the values to be used as search conditions need to be escaped.

Following class is provided by common library as a component to perform escaping for LIKE search.

Sr. No.	Class	Description
1.	org.terasoluna.gfw.common.query. QueryEscapeUtils	Utility class that provide methods to perform escaping of SQL and JPQL. In this class, • method to perform escaping for LIKE search is provided.
2.	org.terasoluna.gfw.common.query. LikeConditionEscape	Class to perform escaping for LIKE search.

Note: `LikeConditionEscape` is a class added from terasoluna-gfw-common 1.0.2.RELEASE to fix "Bugs related to handling of wildcard characters for LIKE search".

`LikeConditionEscape` class plays a role in absorbing the differences in wildcard characters that occur due to difference in database and database versions.

Specifications of escaping of common library

Specifications of escaping provided by common library are as follows:

- Escape character is `"~"` .
- Characters to be escaped by default are 2, namely `"%"` , `"_"` .

Note: Till terasoluna-gfw-common 1.0.1.RELEASE, the characters to be escaped were 4, namely `"%"` , `"_"` , `"％"` , `"＿"` ; however, it is changed to 2 characters namely `"%"` , `"_"` from terasoluna-gfw-common 1.0.2.RELEASE in order to fix the "Bugs related to handling of wildcard characters for LIKE search ".

In addition, a method for escaping that includes double byte characters `"％"` , `"＿"` as characters to be escaped, is also provided.

See the example of escaping below.

[Example of escaping with default specifications]

Example of escaping when default values used as characters to be escaped is given below.

Sr. No.	Target String	After escaping String	Escaping Flag	Description
1.	"a"	"a"	OFF	Escaping not done as the string does not contain character to be escaped.
2.	"a~"	"a~~"	ON	Escaping done as the string contains escape character.
3.	"a%"	"a~%"	ON	Escaping done as the string contains character to be escaped.
4.	"a_"	"a~_"	ON	Similar to No.3.
5.	"_a%"	"~_a~%"	ON	Escaping done as the string contains characters to be escaped. When there are multiple characters to be escaped, escaping is done for all characters.
6.	"a％"	"a％"	OFF	Similar to No.1. From terasoluna-gfw-common 1.0.2.RELEASE, "％" is handled as character out of escaping scope in default specifications.
7.	"a＿"	"a＿"	OFF	Similar to No.1. From terasoluna-gfw-common 1.0.2.RELEASE, "＿" is handled as character out of escaping scope in default specifications.
8.	"　"	"　"	OFF	Similar to No.1.
9.	""	""	OFF	Similar to No.1.
10.	null	null	OFF	Similar to No.1.

[Example of escaping when double byte characters are included]

Example of escaping when double byte characters included as characters to be escaped is given below.

For other than Sr. No. 6 and 7, refer to escaping example of default specifications.

Sr. No.	Target String	After escaping String	Escaping Flag	Description
6.	"a％"	"a~％"	ON	Escaping done as string contains characters to be escaped.
7.	"a＿"	"a~＿"	ON	Similar to No.6.

About escaping methods provided by common library

List of escaping methods for LIKE search of `QueryEscapeUtils` class and `LikeConditionEscape` class provided by common library is given below.

Sr. No.	Method name	Description
1.	toLikeCondition(String)	String passed as an argument is escaped for LIKE search. When specifying type of matching (Forward match, Backward match and Partial match) at SQL or JPQL side, perform only escaping using this method.
2.	toStartingWithCondition(String)	After escaping a string passed as an argument for LIKE search, assign `"%"` at the end of the string after escaping. This method is used in order to convert into a value for Forward match search.
3.	toEndingWithCondition(String)	After escaping a string passed as an argument for LIKE search, assign `"%"` at the beginning of the string after escaping. This method is used in order to convert into a value for Backward match search.
4.	toContainingCondition(String)	After escaping a string passed as an argument for LIKE search, assign `"%"` at the beginning and end of the string after escaping. This method is used in order to convert into a value for Partial match search.

Note: Methods of No.2, 3, 4 are used when specifying the type of matching (Forward match, Backward match and Partial match) at program side and not at SQL or JPQL side.

How to use common library

For example of escaping at the time of LIKE search, refer to the document for O/R Mapper to be used.

- When using MyBatis3, refer to *Escape during LIKE search* of *Database Access (MyBatis3)*.
- When using JPA (Spring Data JPA), refer to *Escaping at the time of LIKE search*of *Database Access (JPA)*.

Note: API for escaping should be used as per wildcard characters supported by database to be used.

[In case of database that supports only "%" , "_" (single byte characters) as wildcard]

```
String escapedWord = QueryEscapeUtils.toLikeCondition(word);
```

Sr. No.	Description
(1)	Escaping is done by directly using method of `QueryEscapeUtils` class.

[In case of database that also supports "%" , "_" (double byte characters) as wildcard]

```
String escapedWord = QueryEscapeUtils.withFullWidth()   // (2)
                         .toLikeCondition(word);        // (3)
```

Sr. No.	Description
(2)	Fetch instance of `LikeConditionEscape` class by calling `withFullWidth()` method of `QueryEscapeUtils` method.
(3)	Perform escaping by using method of `LikeConditionEscape` class instance fetched in (2).

About Sequencer

Sequencer is a common library for fetching sequence value.

Use the sequence value fetched from Sequencer as a configuration value of primary key column of the database.

Note: Reason for creating Sequencer as a common library

The reason for creating Sequencer is that there is no mechanism to format the sequence value as string in ID generator functionality of JPA. In actual application development, sometimes the formatted string is also set as primary key; hence Sequencer is provided as common library.

When value set as primary key is number, it is recommended to use ID generator functionality of JPA. For ID generator functionality of JPA, refer to *How to add entities* of *Database Access (JPA)*.

The primary objective of creating Sequencer is to supplement functions which are not supported by JPA; but it can also be used when sequence value is required in the processes not relating to JPA.

About classes provided by common library

List of classes of Sequencer functionality of common library is as follows:

For usage example, refer to *How to use common library* of How to use.

Sr. No.	Class name	Description
1.	org.terasoluna.gfw.common.sequencer.Sequencer	Interface that defines the method to fetch subsequent sequence value (getNext) and method to return current sequence value (getCurrent).
2.	org.terasoluna.gfw.common.sequencer.JdbcSequencer	Implementation class of `Sequencer` interface for JDBC. This class is used to fetch sequence value by executing SQL in the database. For this class, it is assumed that values are fetched from sequence object of the database; however it is also possible to fetch the values from other than sequence object by calling function stored in the database.

How to use common library

Define a bean for Sequencer.

- `xxx-infra.xml`

```
<!-- (1) -->
<bean id="articleIdSequencer" class="org.terasoluna.gfw.common.sequencer.
↪JdbcSequencer">
    <!-- (2) -->
    <property name="dataSource" ref="dataSource" />
    <!-- (3) -->
    <property name="sequenceClass" value="java.lang.String" />
    <!-- (4) -->
    <property name="nextValueQuery"
        value="SELECT TO_CHAR(NEXTVAL('seq_article'),'AFM0000000000')" />
    <!-- (5) -->
    <property name="currentValueQuery"
        value="SELECT TO_CHAR(CURRVAL('seq_article'),'AFM0000000000')" />
</bean>
```

Sr. No.	Description
(1)	Define a bean for class that implements `org.terasoluna.gfw.common.sequencer.Sequencer`. In the above example, (`JdbcSequencer`) class for fetching sequence value by executing SQL is specified.
(2)	Specify the datasource for executing the SQL to fetch sequence value.
(3)	Specify the type of sequence value to be fetched. In the above example, since conversion to string is done using SQL; `java.lang.String` type is specified.
(4)	Specify SQL for fetching subsequent sequence value. In the above example, sequence value fetched from sequence object of (PostgreSQL) database is formatted as string. When sequence value fetched from the database is `1`, `"A0000000001"` is returned as return value of `Sequencer#getNext()` method.
(5)	Specify SQL for fetching current sequence value. When sequence value fetched from the database is `2`, `"A0000000002"` is returned as return value of `Sequencer#getCurrent()` method.

Fetch sequence value from Sequencer for which bean is defined.

- Service

```
// omitted

// (1)
@Inject
@Named("articleIdSequencer") // (2)
Sequencer<String> articleIdSequencer;

// omitted

@Transactional
```

```
public Article createArticle(Article inputArticle) {

    String articleId = articleIdSequencer.getNext(); // (3)
    inputArticle.setArticleId(articleId);

    Article savedArticle = articleRepository.save(inputArticle);

    return savedArticle;
}
```

Sr. No.	Description
(1)	Inject `Sequencer` object for which bean is defined. In the above example, since sequence value is fetched as formatted string, `java.lang.String` type is specified as generics type of `Sequencer`.
(2)	Specify bean name of the bean to be injected in value attribute of `@javax.inject.Named` annotation. In the above example, bean name (`"articleIdSequencer"`) defined in `xxx-infra.xml` is specified.
(3)	Call `Sequencer#getNext()` method and fetch the subsequent sequence value. In the above example, fetched sequence value is used as Entity ID. When fetching current sequence value, call `Sequencer#getCurrent()` method.

Tip: When `Sequencer` for which bean is defined is 1, `@Named` annotation can be omitted. When specifying multiple sequencers, bean name needs to be specified using `@Named` annotation.

Classes provided by Spring Framework for converting to data access exception

Classes of Spring Framework which play a role in converting an exception to data access exception, are as follows:

Table.6.6 **Classes of Spring Framework for converting to data access exception**

Sr. No.	Class name	Description
1.	org.springframework.jdbc.support.SQLErrorCodeSQLExceptionTranslator	When MyBatis or `JdbcTemplate` is used, JDBC exception is converted to data access exception of Spring Framework using this class. Conversion rules are mentioned in XML file. XML file used by default is `org/springframework/jdbc/support/sql-error-codes.x` in `spring-jdbc.jar`. It is also possible to change the default behavior by placing XML file (`sql-error-codes.xml`) just below class path.
2.	org.springframework.orm.jpa.vendor.HibernateJpaDialect	When JPA (Hibernate implementation) is used, O/R Mapper exception (Hibernate exception) is converted to data access exception of Spring Framework using this class.
3.	org.springframework.orm.jpa.EntityManagerFactoryUtils	If an exception that cannot be converted by `HibernateJpaDialect` has occurred, JPA exception is converted to data access exception of Spring Framework using this class.
4.	Sub classes of org.hibernate.dialect.Dialect	When JPA (Hibernate implementation) is used, exceptions are converted to JDBC exception and O/R Mapper exception using this class.

JDBC datasource classes provided by Spring Framework

Spring Framework provides implementation of JDBC datasource. However since they are very simple classes, they are rarely used in production environment.

These classes are mainly used during Unit Testing.

Table.6.7 **JDBC datasource classes provided by Spring Framework**

Sr. No.	Class name	Description
1.	org.springframework.jdbc.datasource.DriverManagerDataSource	Datasource class for creating new connection by calling `java.sql.DriverManager#getConnection` when connection fetch request is received from the application. When connection pooling is required, Application Server datasource or datasource of OSS/Third-Party library should be used.
2.	org.springframework.jdbc.datasource.SingleConnectionDataSource	Child class of `DriverManagerDataSource`.This class provides implementation of single shared connection. This is a datasource class for unit test which works with single thread. Even in case of Unit Testing, if this class is used when datasource is to be accessed with multithread, care needs to be taken as it may not show the expected behavior.
3.	org.springframework.jdbc.datasource.SimpleDriverDataSource	Datasource class for creating new connection by calling `java.sql.Driver#getConnection` when connection fetch request is received from the application. When connection pooling is required, Application Server datasource or datasource of OSS/Third-Party library should be used.

Spring Framework provides adapter classes with extended JDBC datasource operations.

Specific adapter classes are introduced below.

Table.6.8 **JDBC datasource adapter classes provided by Spring Framework**

Sr. No.	Class name	Description
1.	org.springframework.jdbc.datasource.TransactionAwareDataSourceProxy	Adapter class for converting a datasource which does not store transactions, into a datasource storing Spring Framework transactions.
2.	org.springframework.jdbc.datasource.lookup.IsolationLevelDataSourceRoute	Adapter class for switching the datasource to be used based on independence level of an active transaction.

6.2 Database Access (MyBatis3)

6.2.1 Overview

This chapter describes how to access database using MyBatis3.

This guideline presumes the use of Mapper interface of MyBatis3 as a Repository interface. Refer to "*Implementation of Repository*" for Repository interface.

The architecture to access database by using MyBatis3 and MyBatis-Spring is explained in the Overview.
For more information, refer "*How to use*".

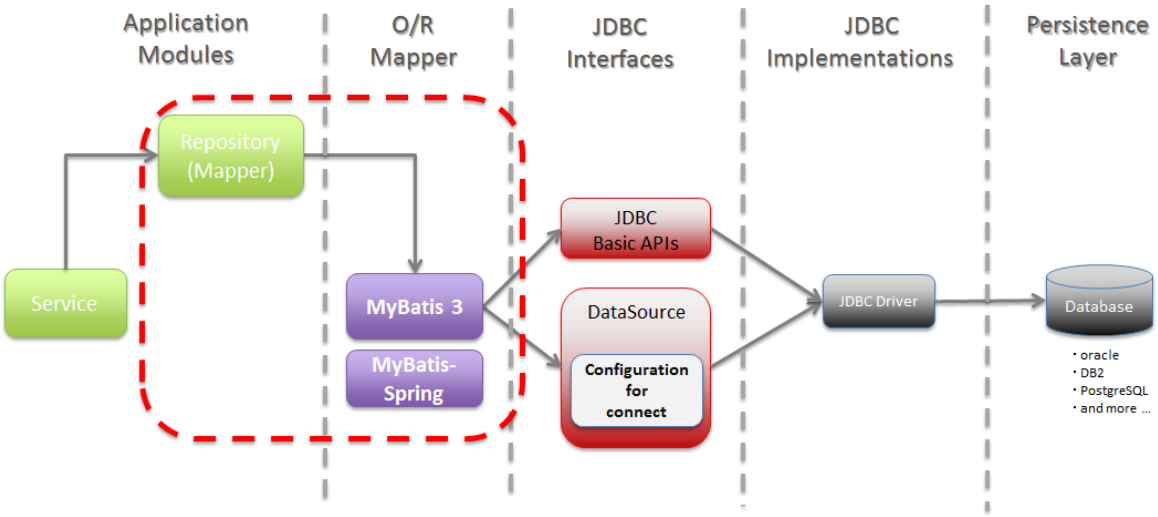


Figure.6.2 **Picture - Scope of description**

About MyBatis3

MyBatis3 is a type of O/R mapper which is developed for mapping SQL to objects. and not for mapping the records stored in database to objects.

Thus, it is an effective O/R mapper to access denormalized databases or to obtain full control of SQL execution in an application without entrusting the SQL statement execution to the O/R mapper.

In this guideline, CRUD operation of Entity is performed by using Mapper interface added from MyBatis3. Refer to "*Mapper interface mechanism*" for details of Mapper interface.

This guideline does not cover explanation of all the functionalities of MyBatis3, Hence, it is recommended to refer "MyBatis 3 REFERENCE DOCUMENTATION " .

Component structure of MyBatis3

The explanation about main components of MyBatis3 (configuration file) is given below.

In MyBatis3, SQL execution and O/R mapping is implemented by integrating the following components with each other based on the definition of configuration file.

Sr. No.	Component/Configuration file	Description
1.	MyBatis configuration file	XML file that explains operation settings of MyBatis3. It is a file that explains details such as connecting destination for database, path of mapping file, operation settings of MyBatis and so on. It is not necessary to specify connecting destination of database and mapping file path settings in the configuration file, when using it by integrating with Spring. However, settings are performed when changing or extending default operations of MyBatis3.
2.	`org.apache.ibatis.session.SqlSessionFactoryBuilder`	A component to read MyBatis configuration file and generate `SqlSessionFactory`. This component is not directly handled by the application class when used by integrating with Spring.
3.	`org.apache.ibatis.session.SqlSessionFactory`	A component to generate `SqlSession`. This component is not directly handled by the application class when used by integrating with Spring.
4.	`org.apache.ibatis.session.SqlSession`	A component to provide API for SQL execution and transaction control. It is the component that plays the most important role when accessing database using MyBatis3. When this component is used by integrating with Spring, it is not directly handled by the application class.
5.	Mapper interface	An interface to call the SQL defined in mapping file in typesafe. Developer needs to create only the interface, as MyBatis3 automatically generates an implementation class for the Mapper interface.
6.	Mapping file	XML file that explains SQL and O/R mapping settings.

Flow by which main components of MyBatis3 access the database, is explained below.

Process for accessing database can be broadly divided into 2 types.

- Processes that are performed at the start of the application. Processes (1) to (3) mentioned below correspond to this type.
- Processes that are performed for each request from the client. Processes (4) to (10) mentioned below

correspond to this type.

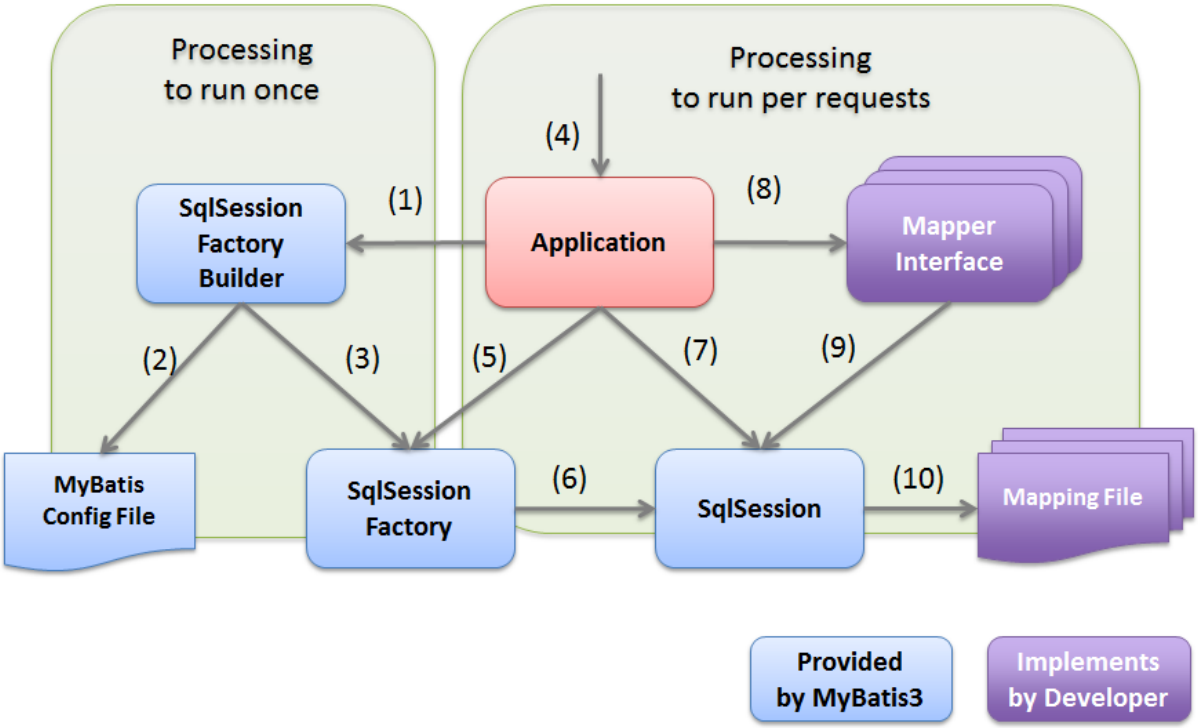


Figure.6.3 **Picture - Relationship of MyBatis3 components**

Processes that are performed at the start of the application are executed with following flow.

Refer to "*Component structure of MyBatis-Spring*" for the flow when integrating with Spring.

Sr. No.	Description
1.	Application requests building `SqlSessionFactory` for `SqlSessionFactoryBuilder`.
2.	`SqlSessionFactoryBuilder` reads MyBatis configuration file for generating `SqlSessionFactory`.
3.	`SqlSessionFactoryBuilder` generates `SqlSessionFactory` based on the definition of MyBatis configuration file.

Processes that are performed for each request from the client are executed with the following flow.

Refer to "*Component structure of MyBatis-Spring*" for the flow when integrating with Spring.

Sr. No.	Description
4.	Client requests a process for the application.
5.	Application fetches `SqlSession` from `SqlSessionFactory` that is built by using `SqlSessionFactoryBuilder` .
6.	`SqlSessionFactory` generates `SqlSession` and returns it to the application.
7.	Application fetches the implementation object of Mapper interface from `SqlSession` .
8.	Application calls Mapper interface method. Refer to "*Mapper interface mechanism*" for mechanism of Mapper interface.
9.	Implementation object of Mapper interface calls `SqlSession` method and requests SQL execution.
10.	`SqlSession` fetches the SQL to be executed from mapping file and executes SQL.

Tip: Transaction control

Commit and rollback for the transaction are performed by calling `SqlSession` API from application code. However, it is not described in the above flow.

When integrated with Spring, Spring transaction control functionality performs commit and rollback. As a result, the API controlling `SqlSession` is not called directly from application class.

Regarding MyBatis3 and Spring integration

MyBatis-Spring is provided by MyBatis as the library to integrate MyBatis3 and Spring.

MyBatis3 components can be stored on Spring DI container by using this library.

There are following advantages of using MyBatis-Spring.

- MyBatis3 SQL can be executed in the transactions managed by Spring. Hence, it is not necessary to control the transactions that are dependent on MyBatis3 API.
- MyBatis3 exception is converted to a generic exception (‘‘org.springframework.dao.DataAccessException‘‘) provided by Spring. Hence, exception process that is not dependent on MyBatis3 API can be implemented.
- The overall initialization process for using MyBatis3 is performed by MyBatis-Spring API. Hence, MyBatis3 API need not be used directly.
- Mapper object can be injected in Singleton Service class to generate a thread safe Mapper object.

This guideline presumes the use of MyBatis-Spring.

This guideline does not cover explanation of all the functionalities of MyBatis-Spring, Hence, it is recommended to refer “Mybatis-Spring REFERENCE DOCUMENTATION ” .

Component structure of MyBatis-Spring

Main components of MyBatis-Spring are explained here.

In MyBatis-Spring, MyBatis3 and Spring are integrated by integrating the following components.

Sr. No.	Component/Configuration file	Description
1.	`org.mybatis.spring.SqlSessionFactoryBean`	Component that builds `SqlSessionFactory` and stores objects on Spring DI container. In standard MyBatis3, `SqlSessionFactory` is built based on the information defined in MyBatis configuration file. However, By using `SqlSessionFactoryBean` , `SqlSessionFactory` can be built even in the absence of MyBatis configuration file. It can also be used in combination.
2.	`org.mybatis.spring.mapper.MapperFactoryBean`	Component that builds Singleton Mapper object and stores objects on Spring DI container. Mapper object generated by MyBatis3 standard mechanism is not thread safe. Hence, it was necessary to assign an instance for each thread. Mapper object created by MyBatis-Spring component can generate a thread safe Mapper object. As a result, DI can be applied to Singleton components like Service etc.
3.	`org.mybatis.spring.SqlSessionTemplate`	`SqlSession` component of Singleton version that implements `SqlSession` interface. `SqlSession` object generated by MyBatis3 standard mechanism is not thread safe. Hence, it was necessary to assign an instance for each thread. `SqlSession` object generated by MyBatis-Spring component can generate a thread safe `SqlSession` object. As a result, DI can be applied to Singleton components like Service etc. However, this guideline does not assume handling `SqlSession` directly.

The flow by which the main components of MyBatis-Spring access the database is explained below. Processes to access database can be broadly divided into two types.

- Processes that are performed at the start of the application. Processes (1) to (4) mentioned below correspond to this type.
- Processes that are performed for each request from the client. Processes (5) to (11) mentioned below correspond to this type.

Processes that are performed at the start of the application are executed by the following flow.

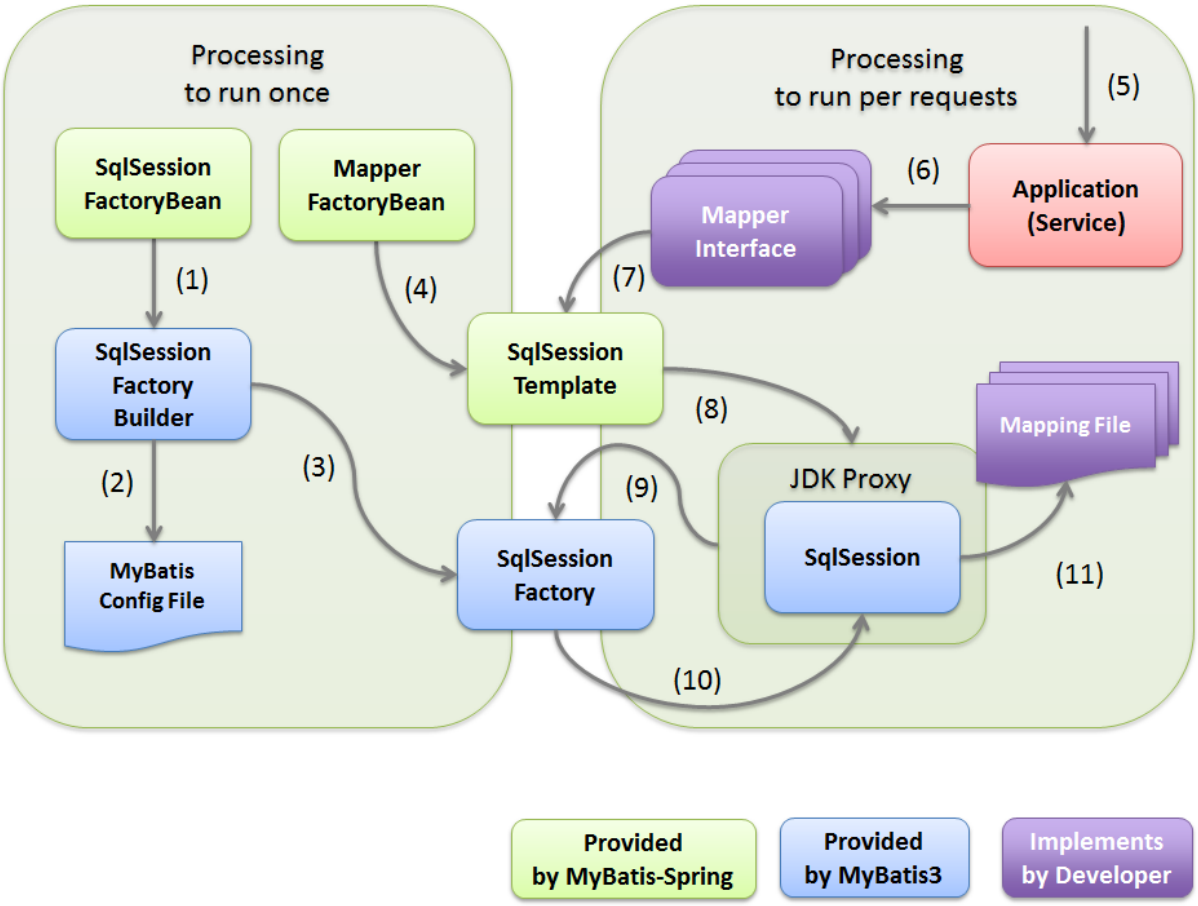


Figure.6.4 **Picture - Relationship of MyBatis-Spring components**

Sr. No.	Description
1.	`SqlSessionFactoryBean` requests building `SqlSessionFactory` for `SqlSessionFactoryBuilder`.
2.	`SqlSessionFactoryBuilder` reads MyBatis configuration file for generating `SqlSessionFactory`.
3.	`SqlSessionFactoryBuilder` generates `SqlSessionFactory` based on the definition of MyBatis configuration file. `SqlSessionFactory` thus generated is stored by the Spring DI container.
4.	`MapperFactoryBean` generates a thread safe `SqlSession` (`SqlSessionTemplate`) and a thread safe Mapper object (Proxy object of Mapper interface). Mapper object thus generated is stored by Spring DI container and DI is applied for Service class etc. Mapper object provides a thread safe implementation by using thread safe `SqlSession` (`SqlSessionTemplate`).

Processes that are performed for each request from client are executed by the following flow.

Sr. No.	Description
5.	Client requests a process for the application.
6.	Application (Service) calls the method of Mapper object (Proxy object that implements Mapper interface) injected by DI container. Refer to "*Mapper interface mechanism*" for Mapper interface mechanism.
7.	Mapper object calls `SqlSession` (`SqlSessionTemplate`) method corresponding to the called method.
8.	`SqlSession` (`SqlSessionTemplate`) calls the proxy enabled and thread safe `SqlSession` method.
9.	Proxy enabled and thread safe `SqlSession` uses MyBatis3 standard `SqlSession` assigned to the transaction. When `SqlSession` assigned to the transaction does not exist, `SqlSessionFactory` method is called to fetch `SqlSession` of standard MyBatis3.
10.	`SqlSessionFactory` returns MyBatis3 standard `SqlSession` . Since the returned MyBatis3 standard `SqlSession` is assigned to the transaction, if it is within the same transaction, same `SqlSession` is used without creating a new one.
11.	MyBatis3 standard `SqlSession` fetches SQL to be executed from mapping file and executes the SQL.

Tip: Transaction control

Although it is not explained in the flow, the commit and rollback of transaction is performed by Spring transaction control function.

Refer to "*Regarding transaction management*" for how to control transaction using Spring transaction control function.

6.2.2 How to use

Actual configuration and implementation methods for accessing database using MyBatis3 are explained below.

The explanation hereafter can be broadly classified as below.

Sr. No.	Classification	Description
1.	Overall application settings	Settings for using MyBatis3 in an application and for changing MyBatis3 operations, are explained below. The contents explained here **are required when application architecture performs settings at the time of project start-up.**Therefore, application developers need not be aware of these contents separately. Following sections correspond to this classification. • *pom.xml settings* • *Settings for integration of MyBatis3 and Spring* • *MyBatis3 settings* When a project is generated from a blank project <https://github.com/terasolunaorg/terasoluna-gfw-web-multi-blank#multi-blank-project-with-mybatis3>‘_ for MyBatis3, a major part of the configuration explained above is already configured. Hence, the application architect determines the project characteristics and adds or changes the configuration as required.
2.	How to implement data access process	How to implement basic data access process using MyBatis3 is explained. The contents explained here **are necessary for the application developers at the time of implementation.** Following sections correspond to this classification. • *Implementation of database access process* • *How to map a JavaBean in Search results* • *Search process for Entity* • *Entity registration process* • *Update process of Entity* • *Delete process for Entity* • *Implementing dynamic SQL* • *Escape during LIKE search* • *SQL Injection countermeasures*

pom.xml settings

When MyBatis3 is used in the infrastructure layer, dependency relation with terasoluna-gfw-mybatis3-dependencies is added to `pom.xml`.

In case of multi project configuration, it is added to `pom.xml` (`projectName-domain/pom.xml`) of domain project.

When a project is generated from a blank project <https://github.com/terasolunaorg/terasoluna-gfw-web-multi-blank#multi-blank-project-with-mybatis3>`_ for MyBatis3, dependency relation with terasoluna-gfw-mybatis3-dependencies is already configured.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
        http://maven.apache.org/maven-v4_0_0.xsd">

    <modelVersion>4.0.0</modelVersion>
    <artifactId>projectName-domain</artifactId>
    <packaging>jar</packaging>

    <parent>
        <groupId>com.example</groupId>
        <artifactId>mybatis3-example-app</artifactId>
        <version>1.0.0-SNAPSHOT</version>
        <relativePath>../pom.xml</relativePath>
    </parent>

    <dependencies>

        <!-- omitted -->

        <!-- (1) -->
        <dependency>
            <groupId>org.terasoluna.gfw</groupId>
            <artifactId>terasoluna-gfw-mybatis3-dependencies</artifactId>
            <type>pom</type>
        </dependency>

        <!-- omitted -->

    </dependencies>

    <!-- omitted -->

</project>
```

Sr. No.	Description
1.	Add terasoluna-gfw-mybatis3 to dependencies. Dependency relation with MyBatis3 and MyBatis-Spring is defined in terasoluna-gfw-mybatis3.

Note: In the above setting example, since it is assumed that the dependent library version is managed by the parent project terasoluna-gfw-parent, specifying the version in pom.xml is not necessary.

Warning: Configuration while using in Java SE 7 environment

terasoluna-gfw-mybatis3-dependencies configures dependency relation considering Java SE 8 as a prerequisite. Java SE 8 dependency library should be excluded as below while using in Java SE 7 environment. For java SE 8 dependency library, refer "*OSS Versions* " of architectural overview.

```
<dependency>
    <groupId>org.terasoluna.gfw</groupId>
    <artifactId>terasoluna-gfw-mybatis3-dependencies</artifactId>
    <exclusions>
        <exclusion>
            <groupId>org.mybatis</groupId>
            <artifactId>mybatis-typehandlers-jsr310</artifactId>
        </exclusion>
    </exclusions>
</dependency>
```

Settings for integration of MyBatis3 and Spring

Datasource settings

When MyBatis3 and Spring are integrated, the datasource managed by Spring DI container should be used.

When a project is generated from a blank project <https://github.com/terasolunaorg/terasoluna-gfw-web-multi-blank#multi-blank-project-with-mybatis3>‘_ for MyBatis3, datasource of Apache Commons DBCP is already configured. Hence the settings should be changed in accordance with project requirements.

Refer to "*Datasource settings* " for the details on how to configure datasource.

Transaction control settings

When MyBatis3 and Spring are integrated, `PlatformTransactionManager` managed by Spring DI container should be used for transaction control.

When using local transaction, `DataSourceTransactionManager` that performs transaction control by calling JDBC API, is used.

When a project is generated from a blank project <https://github.com/terasolunaorg/terasoluna-gfw-web-multi-blank#multi-blank-project-with-mybatis3>‘_ for MyBatis3, `DataSourceTransactionManager` is already configured.

Configuration example is as given below.

- `projectName-env/src/main/resources/META-INF/spring/projectName-env.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:jee="http://www.springframework.org/schema/jee"
    xmlns:jdbc="http://www.springframework.org/schema/jdbc"
    xsi:schemaLocation="http://www.springframework.org/schema/jdbc
        http://www.springframework.org/schema/jdbc/spring-jdbc.xsd
        http://www.springframework.org/schema/jee
        http://www.springframework.org/schema/jee/spring-jee.xsd
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd">

    <!-- omitted -->

    <!-- (1) -->
    <bean id="transactionManager"
        class="org.springframework.jdbc.datasource.
↪DataSourceTransactionManager">
        <!-- (2) -->
        <property name="dataSource" ref="dataSource" />
        <!-- (3) -->
        <property name="rollbackOnCommitFailure" value="true" />
    </bean>

    <!-- omitted -->

</beans>
```

Sr. No.	Description
1.	Specify `org.springframework.jdbc.datasource.DataSourceTransactionManager` as `PlatformTransactionManager`.
2.	Specify configured datasource bean in `dataSource` property. When SQL is executed in the transaction, connection is fetched from datasource specified here.
3.	Rollback process is called when an error occurs during commit. By adding this setting, risk of "Unintentional commit which occurs when a connection with undefined operation returns to connection pool (commit while reusing a connection, implicit commit at the time of closing a connection etc)" can be reduced. However, since an error is likely to occur at the time of rollback, it should be noted that a risk of occurrence of unintentional commit is still a possibility.

Note: bean ID of PlatformTransactionManager

It is recommended to specify `transactionManager` in id attribute.

If a value other than `transactionManager` is specified, the same value must be specified in transaction-manager attribute of `<tx:annotation-driven>` tag.

When a transaction manager provided by application server is used, use `org.springframework.transaction.jta.JtaTransactionManager` that performs transaction control by calling JTA API.

Configuration example is as given below.

- `projectName-env/src/main/resources/META-INF/spring/projectName-env.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:jee="http://www.springframework.org/schema/jee"
    xmlns:jdbc="http://www.springframework.org/schema/jdbc"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xsi:schemaLocation="http://www.springframework.org/schema/jdbc
        http://www.springframework.org/schema/jdbc/spring-jdbc.xsd
        http://www.springframework.org/schema/jee
        http://www.springframework.org/schema/jee/spring-jee.xsd
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/tx
```

```
        http://www.springframework.org/schema/tx/spring-tx.xsd">

    <!-- omitted -->

    <!-- (1) -->
    <tx:jta-transaction-manager />

    <!-- omitted -->

</beans>
```

Sr. No.	Description
1.	If `<tx:jta-transaction-manager />` is specified, bean definition of optimum `JtaTransactionManager` is performed for the application server.

MyBatis-Spring settings

When MyBatis3 and Spring are integrated, it is necessary to carry out following

- Generation of `SqlSessionFactory` that customizes the processes necessary for integrating MyBatis3 and Spring
- Generation of thread safe Mapper object (Proxy object of Mapped interface)

by using MyBatis-Spring components.

When a project is generated from a blank project <https://github.com/terasolunaorg/terasoluna-gfw-web-multi-blank#multi-blank-project-with-mybatis3>‘_ of MyBatis3, the settings for integration of MyBatis3 and Spring are already configured.

Configuration example is as given below.

- `projectName-domain/src/main/resources/META-INF/spring/projectName-infra.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:mybatis="http://mybatis.org/schema/mybatis-spring"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://mybatis.org/schema/mybatis-spring
        http://mybatis.org/schema/mybatis-spring.xsd">

    <import resource="classpath:/META-INF/spring/projectName-env.xml" />
```

```
    <!-- (1) -->
    <bean id="sqlSessionFactory"
        class="org.mybatis.spring.SqlSessionFactoryBean">
        <!-- (2) -->
        <property name="dataSource" ref="dataSource" />
        <!-- (3) -->
        <property name="configLocation"
            value="classpath:/META-INF/mybatis/mybatis-config.xml" />
    </bean>

    <!-- (4) -->
    <mybatis:scan base-package="com.example.domain.repository" />

</beans>
```

Sr. No.	Description
1.	Define the bean for `SqlSessionFactoryBean` as the component for generating `SqlSessionFactory`.
2.	Specify a bean of configured datasource in `dataSource` property. When SQL is executed in MyBatis3 process, the connection is fetched from the datasource specified here.
3.	Specify MyBatis configuration file path in `configLocation` property. The file specified is read when generating `SqlSessionFactory`.
4.	Define `<mybatis:scan>` for scanning Mapper interface and specify the base package that stores the Mapper interface in `base-package` attribute. The Mapper interface stored under specified package is scanned and a thread safe Mapper object (Proxy object of Mapper interface) is automatically generated. **[Package decided for each project should be the specified package]**

Note: How to configure MyBatis3

When `SqlSessionFactoryBean` is used, MyBatis3 configuration can be specified directly in the bean property rather than MyBatis configuration file. However, in this guideline, it is recommended to specify MyBatis3 settings in the MyBatis standard configuration file.

MyBatis3 settings

A mechanism to customize MyBatis3 operations is provided in MyBatis3.

MyBatis3 operations can be customized by adding a configuration value in MyBatis configuration file.

Only those configuration fields that are not dependent on application characteristics are explained here.
For other configuration fields, refer to "MyBatis 3 REFERENCE DOCUMENTATION(Configuration XML) " and configure in accordance with application characteristics.
Default configuration can be maintained, however it should be changed when required in accordance with application characteristics.

Note: Storage location for MyBatis configuration file

In this guideline, it is recommended to store MyBatis configuration file in `projectName-domain/src/main/resources/META-INF/mybatis/mybatis-config.xml`.

When a project is generated from a blank project <https://github.com/terasolunaorg/terasoluna-gfw-web-multi-blank#multi-blank-project-with-mybatis3>'_ of MyBatis3, the file described above is already stored.

`fetchSize` settings

When a query that returns a large amount of data is to be described, an appropriate `fetchSize` must be specified for JDBC driver.
`fetchSize` is a parameter that specifies data record count that can be fetched in a single communication between JDBC driver and database.

Since default value of JDBC driver is used if `fetchSize` is not specified, following issues are likely to appear due to JDBC driver that is being used.

- "Performance degradation" when default value of JDBC driver is small
- "Out of memory" when default value of JDBC driver is large or has no restriction

MyBatis3 can specify `fetchSize` by using 2 methods shown below to control the occurrences of these issues.

- Specifying "default `fetchSize`" applicable for all queries
- Specifying "`fetchSize`for query unit" applicable to a specific query

Note: "Regarding default fetchSize"

"Default `fetchSize`"can be used in MyBatis3.3.0 and subsequent versions.

How to specify "default `fetchSize`" is shown below.

- `projectName-domain/src/main/resources/META-INF/mybatis/mybatis-config.xml`

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE configuration PUBLIC "-//mybatis.org/DTD Config 3.0//EN"
    "http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>

    <settings>
        <!-- (1) -->
        <setting name="defaultFetchSize" value="100" />
    </settings>

</configuration>
```

Sr. No.	Description
1.	Specify data record count fetched in a single communication, in `defaultFetchSize`.

Note: How to specify "fetchsize of query unit"

When `fetchSize` is to be specified in query unit, a value must be specified in `fetchSize` attribute of XML element (`<select>` element) to describe SQL for search.

Note that, when a query for returning large volume of data is to be described, usage of "*Implementation of ResultHandler*" must also be explored.

SQL execution mode settings

MyBatis3 provides following three modes to execute SQL.

The mode to be used should be determined based on characteristics and constraints of each mode, and performance requirements.

Refer to "*Using SQL execution mode*" for how to configure an execution mode.

Sr. No.	Mode	Description
1.	SIMPLE	Creates a new ‘‘java.sql.PreparedStatement‘‘ for each SQL execution. It is a default behavior for MyBatis wherein a blank project <https://github.com/terasolunaorg/terasoluna-gfw-web-multi-blank#multi-blank-project-with-mybatis3>‘_ also operates in a `SIMPLE` mode.
2.	REUSE	Caches and reuses `PreparedStatement`. If `REUSE` mode is used when same SQL is to be executed for multiple times in the same transaction, enhanced performance can be expected as compared to `SIMPLE` mode. This is because it analyses SQL and reduces number of executions for the process that generates `PreparedStatement`.
3.	BATCH	Performs batch execution for Update SQL. (Executes SQL by using `java.sql.Statement#executeBatch()`) If `BATCH` mode is used to execute a large number of Update SQLs in succession, in the same transaction, improved performance can be expected as compared to `SIMPLE` mode or `REUSE` mode. This is because it reduces • Number of executions for the process that generates `PreparedStatement` by analyzing the SQL • Number of communications with the server However, when `BATCH` mode is used, MyBatis behavior operates in `SIMPLE` mode or in a mode different from `SIMPLE` mode. Refer to "*Precautions when using batch mode Repository*" for basic differences and precautions.

TypeAlias settings

When TypeAlias is used, an alias (short name) can be assigned for the Java class specified in mapping file.

When TypeAlias is not used, it is necessary to specify the fully qualified class name (FQCN) of Java class in `type` attribute, `parameterType` attribute and `resultType` attribute specified in a mapping file As a result, decrease in description efficiency and increase in typographical errors of mapping file are the areas of concern.

In this guideline, it is recommended to use TypeAlias to improve description efficiency, reduce typographical errors and improve readability.

When project is generated from a blank project <https://github.com/terasolunaorg/terasoluna-gfw-web-multi-blank#multi-blank-project-with-mybatis3>‘_ for MyBatis3, the class stored under the package (`${projectPackage}.domain.model`) that stores Entity is considered as a target for TypeAlias. Settings should be added as and when required.

How to configure a TypeAlias is given below.

- `projectName-domain/src/main/resources/META-INF/mybatis/mybatis-config.xml`

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE configuration
  PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
  "http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>
    <typeAliases>
        <!-- (1) -->
        <package name="com.example.domain.model" />
    </typeAliases>
</configuration>
```

Sr. No.	Description
1.	In `name` attribute of `package` element, specify the package name in which the class that sets alias is stored. The part from which package is removed acts as an alias for the class that is stored under the specified package. In the above example, the alias for `com.example.domain.model.Account` class is `Account` . **[Package decided for each project should be the specified package]**

Tip: ** How to configure Type Alias in class unit**

A method to configure TypeAlias in class unit and a method to clearly specify an alias are provided in Type Alias settings. Refer to “*TypeAlias settings*” of Appendix for details.

The description example of mapping file when using TypeAlias is as below.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
    "http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<mapper namespace="com.example.domain.repository.account.AccountRepository">

    <resultMap id="accountResultMap"
```

```
        type="Account">
        <!-- omitted -->
    </resultMap>

    <select id="findOne"
        parameterType="string"
        resultMap="accountResultMap">
        <!-- omitted -->
    </select>

    <select id="findByCriteria"
        parameterType="AccountSearchCriteria"
        resultMap="accountResultMap">
        <!-- omitted -->
    </select>

</mapper>
```

Tip: MyBatis3 standard Alias name

An alias name is already configured for general Java classes like primitive type and primitive wrapper type.

Refer to "MyBatis 3 REFERENCE DOCUMENTATION(Configuration XML-typeAliases-) " for the alias names which are already configured.

Mapping settings of NULL value and JDBC type

An error may occur when setting the column value to null for a database that is being used (JDBC driver).

This issue can be resolved by the JDBC driver by configuring `null` value and specifying a recognizable JDBC type.

When setting `null` value, if an error accompanied by following stack traces occurs, mapping of `null` value and JDBC type becomes necessary.

By default, a generic JDBC type called `OTHER` is specified in MyBatis3. However, an error may occur in JDBC driver due to `OTHER`.

```
java.sql.SQLException: Invalid column type: 1111
    at oracle.jdbc.driver.OracleStatement.getInternalType(OracleStatement.
↪java:3916) ~[ojdbc6-11.2.0.2.0.jar:11.2.0.2.0]
    at oracle.jdbc.driver.OraclePreparedStatement.
↪setNullCritical(OraclePreparedStatement.java:4541) ~[ojdbc6-11.2.0.2.0.jar:
↪11.2.0.2.0]
    at oracle.jdbc.driver.OraclePreparedStatement.
↪setNull(OraclePreparedStatement.java:4523) ~[ojdbc6-11.2.0.2.0.jar:11.2.0.2.
↪0]
    ...
```

Note: Operations when using Oracle

When Oracle is used as a database and if default settings is used as is, it has been confirmed that errors occur. Although behavior may change depending on the version, it should be described as 'change in the settings may be required when Oracle is used'.

The version wherein error is confirmed to occur in Oracle 11g R1. The error can be resolved by changing the settings wherein `NULL` type of JDBC type is mapped.

How to change the default behavior of MyBatis3 is given below.

- `projectName-domain/src/main/resources/META-INF/mybatis/mybatis-config.xml`

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE configuration PUBLIC "-//mybatis.org/DTD Config 3.0//EN"
    "http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>

    <settings>
        <!-- (1) -->
        <setting name="jdbcTypeForNull" value="NULL" />
    </settings>

</configuration>
```

Sr. No.	Description
1.	Specify JDBC type in jdbcTypeForNull. In the above example, `NULL`type is specified as JDBC type of `null` value.

Tip: How to resolve at item level

As an alternate method of resolving the error, an appropriate JDBC type supporting the Java type is set in the inline parameters of property wherein `null` value may be set.

However, when JDBC type is individually set in inline parameter, description content of mapping file and occurrence of specified mistakes may increase. Hence, in this guideline, it is recommended to resolve the errors in the overall configuration. If errors are not resolved even after changing the overall configuration, individual setting can be applied only for the property wherein an error has occurred.

TypeHandler settings

`TypeHandler` is used when mapping Java class and JDBC type.

Basically, it is used when

- A Java class object is set as a bind parameter of `java.sql.PreparedStatement` while executing an SQL.
- A value is fetched from `java.sql.ResultSet` that is obtained as SQL execution result.

A `TypeHandler` is provided by MyBatis3 for general Java classes like primitive type and primitive wrapper type class. Specific settings are not required.

Note: Regarding implementation for BLOB and CLOB

`TypeHandler` added by MyBatis 3.4 implements conversion of BLOB and `java.io.InputStream` , and CLOB and `java.io.Reader` by using API added by JDBC 4.0 (Java 1.6). In case of a JDBC driver with JDBC 4.0 support, it is not necessary to implement a new `TypeHandler` since type handler for conversion of BLOB ⇔ `InputStream` and CLOB ⇔ `Reader` is enabled by default.

Configuration while using JSR-310 Date and Time API

When a class which represents date and time offered by JSR-310 Date and Time API in MyBatis3 is used, `TypeHandler` offered by a library different from that of MyBatis (`mybatis-typehandlers-jsr310`) is used. Note that, `mybatis-typehandlers-jsr310` can be used by default in the blank project for MyBatis3 .

Further, since `TypeHandler` is auto-detected in MyBatis3.4 used by TERASOLUNA Server Framework for Java (5.3.x), it is not necessary to add `TypeHandler` to configuration file of MyBatis.

Tip: Refer to "MyBatis 3 REFERENCE DOCUMENTATION(Configuration XML-typeHandlers-) " for a `TypeHandler` provided by MyBatis3.

Tip: Enum type mapping

Enum type is mapped with a constant identifier (string) of Enum in the default behavior of MyBatis3.

In case of Enum type shown below, it is mapped with strings like `"WAITING_FOR_ACTIVE"` , `"ACTIVE"` , `"EXPIRED"` , `"LOCKED"` and stored in the table.

```
package com.example.domain.model;

public enum AccountStatus {
    WAITING_FOR_ACTIVE, ACTIVE, EXPIRED, LOCKED
}
```

In MyBatis, Enum type can be mapped with the numeric value (order in which the constants are defined). For how to map Enum type with a numeric value, refer to "MyBatis 3 REFERENCE DOCUMENTATION(Configuration XML-Handling Enums-) ".

Creating a `TypeHandler` is required while mapping a Joda-Time class and JDBC type not supported by MyBatis3.

Basically, creating a `TypeHandler` is required while mapping `org.joda.time.DateTime` type of "*Date Operations (Joda Time)*" recommended by this guideline and `TIMESTAMP` type of JDBC type.

For creation of `TypeHandler` which maps Joda-Time class and JDBC type, refer "*Implementation of TypeHandler*".

How to apply a `TypeHandler` thus created in MyBatis is explained below.

- `projectName-domain/src/main/resources/META-INF/mybatis/mybatis-config.xml`

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE configuration PUBLIC "-//mybatis.org/DTD Config 3.0//EN"
    "http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>

    <typeHandlers>
        <!-- (1) -->
        <package name="com.example.infra.mybatis.typehandler" />
    </typeHandlers>

</configuration>
```

Sr. No.	Description
1.	Configure `TypeHandler` in MyBatis configuration file. Specify a package name wherein the created `TypeHandler` is stored, in the name attribute of `package` element. The `TypeHandler` stored under specified package is automatically detected by MyBatis.

Tip: In the above example, although `TypeHandler` stored under specified package is automatically detected by MyBatis, it can also be configured in class unit.

`typeHandler` element is used when setting `TypeHandler` in class unit.

- `projectName-domain/src/main/resources/META-INF/mybatis/mybatis-config.xml`

```
<typeHandlers>
    <typeHandler handler="xxx.yyy.zzz.CustomTypeHandler" />
    <package name="com.example.infra.mybatis.typehandler" />
</typeHandlers>
```

Further, when a bean stored by DI container is to be used in `TypeHandler` , `TypeHandler` can be specified in bean definition file.

- `projectName-domain/src/main/resources/META-INF/spring/projectName-infra.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:tx="http://www.springframework.org/schema/tx" xmlns:mybatis=
↪"http://mybatis.org/schema/mybatis-spring"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd
    http://www.springframework.org/schema/tx
```

```
    http://www.springframework.org/schema/tx/spring-tx.xsd
    http://mybatis.org/schema/mybatis-spring
    http://mybatis.org/schema/mybatis-spring.xsd">

    <bean id="sqlSessionFactory" class="org.mybatis.spring.
↪SqlSessionFactoryBean">
        <property name="dataSource" ref="oracleDataSource" />
        <property name="configLocation"
            value="classpath:/META-INF/mybatis/mybatis-config.xml" />
        <property name="typeHandlers">
            <list>
                <bean class="xxx.yyy.zzz.CustomTypeHandler" />
            </list>
        </property>
    </bean>

</beans>
```

The mapping of Java class wherein `TypeHandler` is applied and JDBC type is specified as below.

- Specify as an attribute value of `typeHandler` element in MyBatis configuration file
- Specify in `@org.apache.ibatis.type.MappedTypes` annotation and `@org.apache.ibatis.type.MappedJdbcTypes` annotation
- Specify by inheriting a base class (`org.apache.ibatis.type.BaseTypeHandler`) of `TypeHandler` provided by MyBatis3

For details, refer to "MyBatis 3 REFERENCE DOCUMENTATION(Configuration XML-typeHandlers-)".

Tip: Although each of the above example is a configuration method to be applied to overall application, an individual `TypeHandler` can also be specified for each field. It is used while overwriting a `TypeHandler` that is applicable for overall application.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://
↪mybatis.org/dtd/mybatis-3-mapper.dtd" >
<mapper namespace="com.example.domain.repository.image.ImageRepository">
    <resultMap id="resultMapImage" type="Image">
        <id property="id" column="id" />
```

```
        <!-- (2) -->
        <result property="imageData" column="image_data" typeHandler=
↪"XxxBlobInputStreamTypeHandler" />
        <result property="createdAt" column="created_at"  />
    </resultMap>
    <select id="findOne" parameterType="string" resultMap="resultMapImage">
        SELECT
            id
            ,image_data
            ,created_at
        FROM
            t_image
        WHERE
            id = #{id}
    </select>
    <insert id="create" parameterType="Image">
        INSERT INTO
            t_image
        (
            id
            ,image_data
            ,created_at
        )
        VALUES
        (
            #{id}
            /* (3) */
            ,#{imageData,typeHandler=XxxBlobInputStreamTypeHandler}
            ,#{createdAt}
        )
    </insert>
</mapper>
```

Sr. No.	Description
2.	Specify a `TypeHandler` that is applicable to `typeHandler` attribute of `id` or `result` element while fetching the value from search result (`ResultSet`).
3.	Specify a `TypeHandler` that is applicable to `typeHandler` attribute of inline parameters while configuring a value in the `PreparedStatement`.

It is recommended to set TypeAlias in `TypeHandler` class when TypeHandler is to be individually specified for each field. Refer to "*TypeAlias settings*" for how to configure TypeAlias.

Implementation of database access process

A basic implementation method for accessing a database by using MyBatis3 function is explained below.

Creating Repository interface

A Repository interface is created for each Entity.

```
package com.example.domain.repository.todo;

// (1)
public interface TodoRepository {
}
```

Sr. No.	Description
1.	Create a Repository interface as an interface for Java. In the above example, a Repository interface is created for an Entity called `Todo`.

Creating Mapping file

A mapping file is created for Repository interface.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
    "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<!-- (1)  -->
<mapper namespace="com.example.domain.repository.todo.TodoRepository">
</mapper>
```

Sr. No.	Description
1.	Specify a fully qualified class name (FQCN) of Repository interface in `namespace` attribute of `mapper` element.

Note: Destination to store a mapping file

The mapping file can be stored at either of the locations given below.

- A directory that conforms to the determined rules to enable MyBatis3 to automatically read the mapping file
- An arbitrary directory

In this guideline, it is recommended to use a mechanism wherein mapping file is stored in the directory conforming to the rules determined by MyBatis3 thus enabling automatic reading of file.

It is necessary to store the mapping file on the class path at a level same as the package hierarchy of Repository interface to enable automatic reading of mapping file.

In particular, a mapping file (`TodoRepository.xml`) for Repository interface called `com.example.domain.repository.todo.TodoRepository`should be stored in `projectName-domain/src/main/resources/com/example/domain/repository/todo` directory.

CRUD process implementation

How to implement a CRUD process and considerations when implementing SQL are explained here.

How to implement following processes for the basic CRUD operation is explained.

- *How to map a JavaBean in Search results*
- *Search process for Entity*
- *Entity registration process*
- *Update process of Entity*
- *Delete process for Entity*
- *Implementing dynamic SQL*

Note: It is important to note that the searched Entity is cached in the area called as local cache while implementing CRUD process by using MyBatis3.

Default behavior of local cache provided by MyBatis3 is as given below.

- Local cache is managed in a transaction unit.
- Entity is cached for each "statement ID + pattern of built SQL + parameter value bound to the built SQL + page position (fetch range)".

In other words, when all the search APIs provided by MyBatis3 are called by the same parameter in a process within the same transaction, the instance of cached Entity is returned without executing the SQL from 2nd time onwards.

Here, it should be noted that **Entity returned by MyBatis API and Entity managed by local cache consist of the same instance**.

Tip: Local cache can also be changed so as to be managed in statement unit. When the local cache is to be managed in statement unit, MyBatis executes SQL each time and fetches the latest Entity.

The considerations while implementing SQL are explained below.

- *Escape during LIKE search*
- *SQL Injection countermeasures*

Before explaining the basic implementation, the components to be registered are explained below.

Sr. No.	Component	Description
1.	Entity	A JavaBean class that retains business process data handled by the application. Refer to "*Implementation of Entity*" for details of Entity.
2.	Repository interface	An interface that defines the method to perform CRUD operation of Entity. Refer to "*Implementation of Repository*" for details of Repository.
3.	Service class	A class for executing business process logic. Refer to "*Implementation of Service*" for details of Service.

Note: In this guideline, Mapper interface of MyBatis3 is called as Repository interface in order to standardize the architecture terminology

The explanation hereafter is given presuming that the user has read "*Implementation of Entity*" "*Implementation of Repository*" and "*Implementation of Service*".

How to map a JavaBean in Search results

How to map a JavaBean in the search results is explained before explaining the search process of Entity.

Two methods of automatic and manual, are provided in MyBatis3 to map JavaBean (Entity) in the search results (`ResultSet`). Since both the methods have distinct features, **a mapping method to be used should be determined by considering the project features and features of SQL to be executed by the application.**

Note: Mapping method to be used

The guideline provides two proposals such as

- Automatic mapping is used for simple mapping (mapping to a single object) whereas manual mapping is used in case of advanced mapping (mapping to related objects) is necessary.
- A uniform manual mapping is used

It is not mandatory to use any one of the two methods proposed above and they can be considered as one of the alternatives.

Architect should clearly identify the criteria for selecting manual mapping and automatic mapping for the programmers and look for a uniform mapping method for the entire application.

The respective features and examples for automatic mapping and manual mapping are explained below.

Automatic mapping for search results

In MyBatis3, a mechanism which automatically performs the mapping by matching column name and property name is provided as a method to map search result (`ResultSet`) column and JavaBean property.

Note: Features of automatic mapping

When automatic mapping is used, only SQL to be executed is described in the mapping file thus reducing the description content of mapping file.

By reducing the description, simple mistakes and modification locations while changing a column name or a property name can be reduced as well.

However, automatic mapping can only be used for single object. Manual mapping is required when mapping for the nested related objects.

Tip: Column names

Column name mentioned here does not signify the physical column name of a table but refers to the column which contains the search result (`ResultSet`) fetched by executing the SQL. Therefore, by using AS clause, matching a physical column name and a JavaBean property name is comparatively easier.

How to map search results in JavaBean using automatic mapping is shown below.

- `projectName-domain/src/main/resources/com/example/domain/repository/todo/TodoReposi`

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
    "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.example.domain.repository.todo.TodoRepository">

    <select id="findOne" parameterType="string" resultType="Todo">
        SELECT
            todo_id AS "todoId", /* (1) */
            todo_title AS "todoTitle",
            finished, /* (2) */
            created_at AS "createdAt",
            version
        FROM
            t_todo
        WHERE
            todo_id = #{todoId}
    </select>

</mapper>
```

Sr. No.	Description
1.	When physical column name of table and JavaBean property name are different, automatic mapping can be applied by using AS clause for matching.
2.	When physical column name of table and JavaBean property name match, there is no need to specify the AS clause.

- JavaBean

```
package com.example.domain.model;

import java.io.Serializable;
import java.util.Date;

public class Todo implements Serializable {

    private static final long serialVersionUID = 1L;

    private String todoId;

    private String todoTitle;

    private boolean finished;

    private Date createdAt;

    private long version;

    public String getTodoId() {
        return todoId;
    }

    public void setTodoId(String todoId) {
        this.todoId = todoId;
    }

    public String getTodoTitle() {
        return todoTitle;
    }

    public void setTodoTitle(String todoTitle) {
        this.todoTitle = todoTitle;
    }

    public boolean isFinished() {
        return finished;
    }

    public void setFinished(boolean finished) {
        this.finished = finished;
    }

    public Date getCreatedAt() {
        return createdAt;
    }

    public void setCreatedAt(Date createdAt) {
        this.createdAt = createdAt;
    }
```

```
    public long getVersion() {
        return version;
    }

    public void setVersion(long version) {
        this.version = version;
    }

}
```

Tip: How to map a column name separated by an underscore and a property name in camel case format

In the above example, the difference between a column name separated by an underscore and a property name in camel case format is resolved by using AS clause. However, it can be implemented by changing MyBatis3 configuration if only the difference between a column name separated by an underscore and a property name in camel case format is to be resolved.

When the physical column name of a table is separated by an underscore, automatic mapping can be performed in JavaBean property in the camel case format by adding following settings to MyBatis configuration file (`mybatis-config.xml`).

- `projectName-domain/src/main/resources/META-INF/mybatis/mybatis-config.xml`

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE configuration
  PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
  "http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>

    <settings>
        <!-- (3) -->
        <setting name="mapUnderscoreToCamelCase" value="true" />
    </settings>

</configuration>
```

Sr. No.	Description
3.	Add the settings to set `mapUnderscoreToCamelCase` to *true*. When it is set to *true*, the column name separated by an underscore is automatically converted to camel case format. As a typical example, when column name is `"todo_id"`, it is converted to `"todoId"` and mapping is performed.

- `projectName-domain/src/main/resources/com/example/domain/repository/todo/TodoReposi`

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
    "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.example.domain.repository.todo.TodoRepository">

    <select id="findOne" parameterType="string" resultType="Todo">
        SELECT
            todo_id, /* (4) */
            todo_title,
            finished,
            created_at,
            version
        FROM
            t_todo
        WHERE
            todo_id = #{todoId}
    </select>

</mapper>
```

Sr. No.	Description
4.	A simple SQL can be fetched since AS clause is not required to resolve the difference between the column name separated by an underscore and the property name in the camel case format.

Manual mapping of search results

MyBatis3 provides a mechanism to manually map search result (`ResultSet`) column and JavaBean property by defining their association in the mapping file.

Note: Features of manual mapping

When manual mapping is used, the association between search result (`ResultSet`) column and JavaBean property is defined for each item one by one in the mapping file. Therefore, mapping with extremely high flexibility and complexity can be achieved.

Manual mapping is a method to effectively map the search results (`ResultSet`) column and JavaBean property for the cases given below.

- When data model (JavaBean) that handles the application and physical table layout do not match
- When JavaBean has a nested structure (separate JavaBean is nested)

Also, manual mapping can be mapped efficiently compared with automatic mapping. If prevail efficiency of processing, it is desirable to use manual mapping instead of automatic mapping.

How to map search results in JavaBean using manual mapping is given below.

Since the idea is to explain how to use manual mapping, a simple example wherein automatic mapping can also be performed, is used for the explanation.

For a hands-on implementation example, refer to

- "MyBatis 3 REFERENCE DOCUMENTATION(Mapper XML Files-Advanced Result Maps-) "
- "*How to fetch a related Entity by a single SQL*"
- "*How to fetch a related Entity using a nested SQL*"
- `projectName-domain/src/main/resources/com/example/domain/repository/todo/TodoReposi`

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
    "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.example.domain.repository.todo.TodoRepository">

    <!-- (1) -->
    <resultMap id="todoResultMap" type="Todo">
        <!-- (2) -->
        <id column="todo_id" property="todoId" />
        <!-- (3) -->
        <result column="todo_title" property="todoTitle" />
        <result column="finished" property="finished" />
        <result column="created_at" property="createdAt" />
        <result column="version" property="version" />
    </resultMap>

    <!-- (4) -->
    <select id="findOne" parameterType="string" resultMap="todoResultMap">
```

```
        SELECT
            todo_id,
            todo_title,
            finished,
            created_at,
            version
        FROM
            t_todo
        WHERE
            todo_id = #{todoId}
    </select>

</mapper>
```

Sr. No.	Description
1.	Define the mapping of search results (ResultSet) and JavaBean, in <resultMap> element. Specify the ID to identify mapping in id attribute and the JavaBean class name (or alias) to be mapped, in type attribute. Refer to "MyBatis 3 REFERENCE DOCUMENTATION(Mapper XML Files-resultMap-) " for details of <resultMap> element, .
2.	Map search results (ResultSet) ID (PK) column and JavaBean property. Specify mapping of ID (PK) by using <id> element. Specify search result (ResultSet) column name in column attribute and JavaBean property name in property attribute. Refer to "MyBatis 3 REFERENCE DOCUMENTATION(Mapper XML Files-id & result-) " for details of <id> element.
3.	Map a column other than ID (PK) column of search results (ResultSet) and JavaBean property. Specify mapping for column other than ID (PK) using <result> element. Specify search result (ResultSet) column name in column attribute and JavaBean property name in property attribute. Refer to "MyBatis 3 REFERENCE DOCUMENTATION(Mapper XML Files-id & result-) " for details of <result> element.
4.	Specify mapping definition ID to be applied, in resultMap attribute of <select> element.

Note: How to use id element and result element

<id> element and <result> element can both be used for mapping search results (ResultSet) column and JavaBean property. However, it is recommended to use <id> element for mapping of ID (PK) column.

This is because, when <id> element is used for the mapping of ID (PK) column, the performance of mapping process for related objects and the cache control process of objects provided by MyBatis3 can

show overall improvement.

Search process for Entity

How to implement a search process of Entity for different purposes, is explained below.

Read "*How to map a JavaBean in Search results*" before reading how to implement the search process for Entity.

The explanation below is the example wherein a setting is enabled to automatically map column name separated by an underscore in property name with camel case.

- `projectName-domain/src/main/resources/META-INF/mybatis/mybatis-config.xml`

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE configuration
  PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
  "http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>

    <settings>
        <setting name="mapUnderscoreToCamelCase" value="true" />
    </settings>

</configuration>
```

Fetching a single key Entity

Implementation example wherein, a single Entity is fetched by specifying PK rather than configuring PK in a single column, is given below.

- Define method in Repository interface.

```
package com.example.domain.repository.todo;

import com.example.domain.model.Todo;

public interface TodoRepository {

```

```
    // (1)
    Todo findOne(String todoId);

}
```

Sr. No.	Description
1.	In the above example, `findOne` method is defined as the method to fetch a single Todo object matching with `todoId` (PK) specified in the argument.

- Define SQL in the mapping file.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
    "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.example.domain.repository.todo.TodoRepository">

    <!-- (2) -->
    <select id="findOne" parameterType="string" resultType="Todo">
        /* (3) */
        SELECT
            todo_id,
            todo_title,
            finished,
            created_at,
            version
        FROM
            t_todo
        /* (4) */
        WHERE
            todo_id = #{todoId}
    </select>

</mapper>
```

Sr. No.	Attribute	Description
2.	-	Implements SQL in `select` element with search result 0 to 1 record. In the above example, the SQL fetching a record that matches with ID (PK) is implemented. For details of `select` element, refer to "MyBatis3 REFERENCE DOCUMENTATION (Mapper XML Files-select-)".
	id	Specifies method name of the method defined in Repository interface.
	parameterType	Specifies parameter fully qualified class name (or alias name).
	resultType	Specifies the fully qualified class name (or alias) of JavaBean that maps the search results (`ResultSet`). When manual mapping is used, specify mapping definition to be applied, by using `resultMap` attribute in place of `resultType` attribute. Refer to "*Manual mapping of search results*" for manual mapping.
3.	-	Specify the column to be fetched. In the above example, automatic mapping is used as the method to map search results (`ResultSet`) to JavaBean. Refer to "*Automatic mapping for search results*" for automatic mapping.
4.	-	Specify search conditions in WHERE clause. Specify the value to be bound in search condition as the bind value of `#{variableName}` format. In the above example, `#{todoId}` acts as the bind variable. When argument type of Repository interface is of simple type like `String`, any name can be specified as the bind variable name, however when the argument type is JavaBean, JavaBean property name must be specified in the bind variable name.

Note: Simple type bind variable name

In case of a simple type like `String`, there is no restriction for the bind variable name, however, it is recommended to use the value same as the argument name of the method.

- Apply DI to Repository in Service class and call the interface method of Repository.

```
package com.example.domain.service.todo;

import javax.inject.Inject;

import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
```

```
import com.example.domain.model.Todo;
import com.example.domain.repository.todo.TodoRepository;

@Transactional
@Service
public class TodoServiceImpl implements TodoService {

    // (5)
    @Inject
    TodoRepository todoRepository;

    @Transactional(readOnly = true)
    @Override
    public Todo getTodo(String todoId) {
        // (6)
        Todo todo = todoRepository.findOne(todoId);
        if (todo == null) { // (7)
            throw new ResourceNotFoundException(ResultMessages.error().add(
                    "e.ex.td.5001", todoId));
        }
        return todo;
    }

}
```

Sr. No.	Description
5.	Apply DI to Repository interface in Service class.
6.	Call Repository interface method and fetch 1 Entity.
7.	Since `null` is returned when the search result shows 0 records, if required, implement the process when Entity cannot be fetched. In the above example, when Entity cannot be fetched, “resource not detected” error is generated.

Fetching Entity of composite key

The implementation example of fetching a single Entity by specifying PK rather than configuring PK in multiple columns, is given below.

Basic settings are same as while configuring PK in a single column, however the way to specify a method

argument for Repository interface is different.

- Defining the method in Repository interface.

```
package com.example.domain.repository.order;

import org.apache.ibatis.annotations.Param;

import com.example.domain.model.OrderHistory;

public interface OrderHistoryRepository {

    // (1)
    OrderHistory findOne(@Param("orderId") String orderId,
            @Param("historyId") int historyId);

}
```

Sr. No.	Description
1.	Define an argument corresponding to the column that configures PK, in the method. In the above example, `orderId` and `historyId` are defined in the argument as PK for the table that manages change history of orders received.

Tip: Bind variable name while specifying multiple method arguments

When multiple method arguments of Repository interface are specified, it is recommended to specify `@org.apache.ibatis.annotations.Param` annotation in the argument. "Bind variable name" specified while selecting the value from mapping file is specified in the `value` attribute of `@Param` annotation.

As shown in the above example, the value specified in the argument can be bound in SQL by specifying `#{orderId}` and `#{historyId}` from mapping file.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
    "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.example.domain.repository.order.
↪OrderHistoryRepository">

    <select id="findOne" resultType="OrderHistory">
        SELECT
            order_id,
            history_id,
            order_name,
            operation_type,
            created_at"
        FROM
            t_order_history
```

```
        WHERE
            order_id = #{orderId}
        AND
            history_id = #{historyId}
    </select>

</mapper>
```

Although it is not mandatory to specify `@Param` annotation, if it is not specified, a mechanical bind variable name needs to be specified as given below. The bind variable name when `@Param` annotation is not specified is formed as, " "param" + declared position of the argument(start from 1)", and thus can hamper maintainability and readability of the source code.

```
<!-- omitted -->

WHERE
    order_id = #{param1}
AND
    history_id = #{param2}

<!-- omitted -->
```

Entity search

Implementation example is given below wherein a SQL with search results 0 to N records is executed and multiple records of Entity are fetched.

> **Warning:** If the search results data is in a large quantity, using "*Implementation of ResultHandler*" should be considered.

- Defining the method for fetching multiple records of Entity.

```
package com.example.domain.repository.todo;
```

```
import java.util.List;

import com.example.domain.model.Todo;

public interface TodoRepository {

    // (1)
    List<Todo> findAllByCriteria(TodoCriteria criteria);

}
```

Sr. No.	Description
1.	In the above example, `findAllByCriteria` is defined as the method to fetch multiple records of Todo object in a list format that matches with the JavaBean (`TodoCriteria`) retaining the search conditions.

Tip: In the above example, the return value of method is specified as `java.util.List` however, search results can also be received as `java.util.Map`.

When the results are received in `Map`,

- PK value is stored in `key` of `Map`
- Entity object is stored in `value` of `Map`.

When search results are received by `Map`, `java.util.HashMap` instance is returned. Hence, it should be noted that the alignment sequence of `Map`is not guaranteed.

Implementation example is given below.

```
package com.example.domain.repository.todo;

import java.util.Map;

import com.example.domain.model.Todo;
import org.apache.ibatis.annotations.MapKey;

public interface TodoRepository {

    @MapKey("todoId")
    Map<String, Todo> findAllByCriteria(TodoCriteria criteria);

}
```

When search results are received by `Map`, `@org.apache.ibatis.annotations.MapKey` annotation is specified in the method. Property name that is handled as `key` of `Map` is specified in the `value` attribute of annotation. In the above example, PK of Todo object (`todoId`) is specified.

- Create JavaBean that retains the search conditions.

```
package com.example.domain.repository.todo;

import java.io.Serializable;
import java.util.Date;

public class TodoCriteria implements Serializable {

    private static final long serialVersionUID = 1L;

    private String title;

    private Date createdAt;

    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }

    public Date getCreatedAt() {
        return createdAt;
    }

    public void setCreatedAt(Date createdAt) {
        this.createdAt = createdAt;
    }

}
```

Note: Creating a JavaBean for retaining search conditions

Although it is not mandatory to create a JavaBean for retaining search conditions, it is recommended to create one, to clearly identify the role of the stored value. However, implementation can also be performed without creating a JavaBean.

The decision standards of the cases for which JavaBean is created and those for which it is not created should be clearly stated to the programmers by the Architect, so that an overall uniform application can be created.

Implementation example when JavaBean is not created is given below.

```
package com.example.domain.repository.todo;

import java.util.List;

import com.example.domain.model.Todo;

public interface TodoRepository {

    List<Todo> findAllByCriteria(@Param("title") String title,
            @Param("createdAt") Date createdAt);

}
```

When JavaBean is not created, the search conditions are declared one by one in an argument and "bind variable name" is specified in `value` attribute of `@Param` annotation. Multiple search conditions can be passed to SQL by defining the method described above.

- Define SQL in the mapping file.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
    "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.example.domain.repository.todo.TodoRepository">

    <!-- (2) -->
    <select id="findAllByCriteria" parameterType="TodoCriteria" resultType=
↪"Todo">
        <![CDATA[
        SELECT
            todo_id,
            todo_title,
            finished,
            created_at,
            version
        FROM
            t_todo
        WHERE
            todo_title LIKE #{title} || '%' ESCAPE '~'
        AND
            created_at < #{createdAt}
        /* (3) */
        ORDER BY
            todo_id
```

```
        ]]>
    </select>

</mapper>
```

Sr. No.	Description
2.	Implement the SQL with search results 0 to N records in `select` element. In the above example, Todo records that match with the conditions specified in `todo_title` and `created_at` are fetched.
3.	Specify sort condition. When multiple records are to be fetched, sort condition is specified. Particularly, sort condition must be specified in the SQL that fetches the record displayed on the screen.

Tip: How to use CDATA section

When a XML character (`"<"` or `">"` etc.) that needs to be escaped in SQL is specified, the readability of SQL can be maintained by using CDATA section. When CDATA section is not used, entity reference characters such as `"<"`, `">"` need to be specified, and may lead to reduced SQL readability.

In the above example, CDATA section is specified since `"<"` is used as the condition for `created_at`.

Fetching Entity records

The implementation example of fetching Entity records matching with search conditions is given below.

- Defining the method for fetching Entity records matching with search conditions.

```
package com.example.domain.repository.todo;

public interface TodoRepository {

    // (1)
    long countByFinished(boolean finished);

}
```

Sr. No.	Description
1.	Specify numeric type (`int` or `long` etc.) for the return value of method used to fetch records. In the above example, `long` is specified.

- Define SQL in the mapping file.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
    "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.example.domain.repository.todo.TodoRepository">

    <!-- (2) -->
    <select id="countByFinished" parameterType="_boolean" resultType="_long">
        SELECT
            COUNT(*)
        FROM
            t_todo
        WHERE
            finished = #{finished}
    </select>

</mapper>
```

Sr. No.	Description
2.	Execute the SQL that fetches the records. Type of return value is specified in `resultType` attribute. In the above example, primitive type alias name for specifying `long` is specified.

Tip: Primitive type alias name

"_" (underscore) should be specified at the beginning of the primitive type alias name. When "_"(underscore) is not specified, it is handled as primitive wrapper type (`java.lang.Long` etc.) alias.

Pagination search of Entity (MyBatis3 standard method)

Implementation example to search an Entity by using MyBatis3 functionality for specifying the fetching scope, is given below.

`org.apache.ibatis.session.RowBounds` class is provided in MyBatis as the class to specify the fetch range. and in SQL, it is not necessary to describe the conditions for fetch range.

Warning: Precautions when large number of data records match the search conditions

Standard MyBatis method is to move the cursor and skip the data which is outside the fetch range of search results (`ResultSet`). Hence, in proportion to the data records that match with search conditions, issues like memory exhaustion or performance degradation of cursor movement are more likely to occur.

According to JDBC result set type, the cursor movement processing supports following 2 types. Default behavior is dependent on the default result set type of JDBC driver.

- When result set type is `FORWARD_ONLY`, `ResultSet#next()` is repeatedly called and data outside the fetching range is skipped.
- When result set type is `SCROLL_SENSITIVE` or `SCROLL_INSENSITIVE`, `ResultSet#absolute(int)` is called and data outside the scope of fetching range is skipped.

Performance degradation can be restricted to a minimum by using `ResultSet#absolute(int)` however, it is dependent on the implementation of JDBC driver. If process same as `ResultSet#next()` is performed internally, it is not possible to prevent memory exhaustion or performance deterioration.

When there is a possibility of large number of data records matching the search conditions, SQL refine method should be adopted instead of pagination search which is a MyBatis3 standard method.

- Defining the method for performing Entity pagination search.

```
ackage com.example.domain.repository.todo;

import java.util.List;

import org.apache.ibatis.session.RowBounds;

import com.example.domain.model.Todo;

public interface TodoRepository {

    // (1)
    long countByCriteria(TodoCriteria criteria);

    // (2)
    List<Todo> findPageByCriteria(TodoCriteria criteria,
```

```
        RowBounds rowBounds);

}
```

Sr. No.	Description
1.	Define the method that fetches total records of Entity matching with search conditions.
2.	Define the method that extracts those Entities that fall in the fetching range from the Entities matching with search conditions. `RowBounds` that retains the information of fetch range (offset and limit) is specified as the argument of defined method.

- Define SQL in the mapping file.

  Since MyBatis3 performs the process to extract records of corresponding range from the search results, it is not necessary to filter the records within the fetch range using SQL.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
    "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.example.domain.repository.todo.TodoRepository">

    <select id="countByCriteria" parameterType="TodoCriteria" resultType="_
↪long">
        <![CDATA[
        SELECT
            COUNT(*)
        FROM
            t_todo
        WHERE
            todo_title LIKE #{title} || '%' ESCAPE '~'
        AND
            created_at < #{createdAt}
        ]]>
    </select>

    <select id="findPageByCriteria" parameterType="TodoCriteria" resultType=
↪"Todo">
        <![CDATA[
        SELECT
            todo_id,
            todo_title,
            finished,
            created_at,
```

```
            version
        FROM
            t_todo
        WHERE
            todo_title LIKE #{title} || '%' ESCAPE '~'
        AND
            created_at < #{createdAt}
        ORDER BY
            todo_id
        ]]>
    </select>

</mapper>
```

Note: Standardization of WHERE clause

When pagination search is performed, it is recommended to standardize the WHERE clause specified in "SQL that fetches total number of records for Entities matching with search condition" and "SQL that fetches the list of the Entities matching with search conditions", using include function of MyBatis3.

A standardized WHERE clause of above SQL is defined as below. Refer to "*Sharing SQL statement*" for details.

```
<sql id="findPageByCriteriaWherePhrase">
    <![CDATA[
    WHERE
        todo_title LIKE #{title} || '%' ESCAPE '~'
    AND
        created_at < #{createdAt}
    ]]>
</sql>

<select id="countByCriteria" parameterType="TodoCriteria" resultType="_long
↪">
    SELECT
        COUNT(*)
    FROM
        t_todo
    <include refid="findPageByCriteriaWherePhrase"/>
</select>

<select id="findPageByCriteria" parameterType="TodoCriteria" resultType=
↪"Todo">
    SELECT
        todo_id,
        todo_title,
        finished,
        created_at,
```

```
        version
    FROM
        t_todo
    <include refid="findPageByCriteriaWherePhrase"/>
    ORDER BY
        todo_id
</select>
```

Note: How to explicitly specify a result set type

Result set type is specified in `resultType` attribute when it is to be specified explicitly. When the default result set type of JDBC driver is `FORWARD_ONLY`, it is recommended to specify `SCROLL_INSENSITIVE`.

```
<select id="findPageByCriteria" parameterType="TodoCriteria" resultType=
↪"Todo"
    resultSetType="SCROLL_INSENSITIVE">
    <!-- omitted -->
</select>
```

- Implementing pagination search process in Service class.

```
// omitted

@Transactional
@Service
public class TodoServiceImpl implements TodoService {

    @Inject
    TodoRepository todoRepository;

    // omitted

    @Transactional(readOnly = true)
    @Override
    public Page<Todo> searchTodos(TodoCriteria criteria, Pageable pageable) {
        // (3)
        long total = todoRepository.countByCriteria(criteria);
        List<Todo> todos;
        if (0 < total) {
            // (4)
```

```
            RowBounds rowBounds = new RowBounds(pageable.getOffset(),
                pageable.getPageSize());
            // (5)
            todos = todoRepository.findPageByCriteria(criteria, rowBounds);
        } else {
            // (6)
            todos = Collections.emptyList();
        }
        // (7)
        return new PageImpl<>(todos, pageable, total);
    }

    // omitted

}
```

Sr. No.	Description
3.	First, fetch the total Entity records matching with search condition.
4.	Generate `RowBounds` object that specifies fetch range of pagination search when Entities matching with search conditions exist. Specify "skip record" in the first argument (`offset`) and "maximum fetch records" in the second argument (`limit`) of `RowBounds`. For the values to be specified as argument, it is advisable to specify the values fetched by calling `getOffset` method and `getPageSize` method of `Pageable` object provided by Spring Data Commons. Basically, the fetch range is • Records 1st to 20th when `0` is specified in offset and `20` is specified in limit • Records 21st to 40th when `20`is specified in offset and `20` is specified in limit
5.	Call Repository method and fetch Entities in the fetch range that match with search conditions.
6.	When the Entities that match with search conditions do not exist, set empty list in the search results.
7.	Create and return page information (`org.springframework.data.domain.PageImpl`).

Pagination search for Entity (SQL refinement method)

Implementation example to search an Entity by using range search mechanism provided by database, is given below.

Since SQL refinement method uses range search mechanism provided by database, Entity of fetch range can be fetched efficiently as compared to standard method of MyBatis3.

Note: It is recommended to adopt the SQL refining method when a large volume of data matching with search condition exists.

- Defining the method for performing Entity pagination search.

```
package com.example.domain.repository.todo;

import java.util.List;

import org.apache.ibatis.annotations.Param;
import org.springframework.data.domain.Pageable;

import com.example.domain.model.Todo;

public interface TodoRepository {

    // (1)
    long countByCriteria(
            @Param("criteria") TodoCriteria criteria);

    // (2)
    List<Todo> findPageByCriteria(
            @Param("criteria") TodoCriteria criteria,
            @Param("pageable") Pageable pageable);
}
```

Sr. No.	Description
1.	Define a method that fetches total Entity records matching with search conditions.
2.	Define a method to extract entities that can be fetched from the Entities matching with search conditions. `org.springframework.data.domain.Pageable` that retains the information within the fetch range (offset and limit) is specified as an argument for defined method.

Note: Reason why the argument specifies @Param annotation for a single method

In the above example, the argument specifies `@Param` annotation for a single method (`countByCriteria`). This is to standardize WHERE clause and the SQL executed when `findPageByCriteria` method is called.

By specifying bind variable name in the argument using `@Param`annotation, nested structure of bind variable name specified in SQL is combined.

A typical SQL implementation example is given below.

- Define SQL in the mapping file.

  Fetch range records are refined by SQL.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
    "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.example.domain.repository.todo.TodoRepository">

    <sql id="findPageByCriteriaWherePhrase">
        <![CDATA[
        /* (3) */
        WHERE
            todo_title LIKE #{criteria.title} || '%' ESCAPE '~'
        AND
            created_at < #{criteria.createdAt}
        ]]>
    </sql>

    <select id="countByCriteria" resultType="_long">
        SELECT
            COUNT(*)
```

```
        FROM
            t_todo
        <include refid="findPageByCriteriaWherePhrase" />
    </select>

    <select id="findPageByCriteria" resultType="Todo">
        SELECT
            todo_id,
            todo_title,
            finished,
            created_at,
            version
        FROM
            t_todo
        <include refid="findPageByCriteriaWherePhrase" />
        ORDER BY
            todo_id
        LIMIT
            #{pageable.pageSize} /* (4) */
        OFFSET
            #{pageable.offset}  /* (4) */
    </select>

</mapper>
```

Sr. No.	Description
3.	`@Param("criteria")` is specified in the arguments of `countByCriteria` and `findPageByCriteria` methods, Hence, the bind variable name specified in SQL is in `criteria.field name` format.
4.	Extract only necessary records by using the fetch range mechanism provided by database. "Skip record" is stored in `offset` of `Pageable` object whereas "maximum fetch records" is stored in `pageSize`. Above example is the implementation example with H2 Database.

- Implement a pagination search process in the Service class.

```
// omitted

@Transactional
@Service
public class TodoServiceImpl implements TodoService {

    @Inject
```

```
    TodoRepository todoRepository;

    // omitted

    @Transactional(readOnly = true)
    @Override
    public Page<Todo> searchTodos(TodoCriteria criteria,
            Pageable pageable) {
        long total = todoRepository.countByCriteria(criteria);
        List<Todo> todos;
        if (0 < total) {
            // (5)
            todos = todoRepository.findPageByCriteria(criteria,
                    pageable);
        } else {
            todos = Collections.emptyList();
        }
        return new PageImpl<>(todos, pageable, total);
    }

    // omitted

}
```

Sr. No.	Description
5.	Call Repository method and fetch Entity within the fetch range matching with search condition. `Pageable` object received by the argument can be passed as it is when calling Repository method.

Entity registration process

how to register an Entity for different purposes is explained with implementation example.

Registering a single Entity record

Implementation example for registering a single Entity record is given below.

- Defining the method in Repository interface.

```
package com.example.domain.repository.todo;

import com.example.domain.model.Todo;
```

```
public interface TodoRepository {

    // (1)
    void create(Todo todo);

}
```

Sr. No.	Description
1.	In the above example, `create` method is defined as the method for registering a single Todo object specified in the argument.

Note: Return value of the method that registers Entity

Return value for the method that registers Entity can be `void`.

However, when SQL that inserts selected results is executed, `boolean` or numeric value type (`int` or `long`) should be set as the return value based on application requirements.

- When `boolean` is specified as return value, `false` is returned when 0 records are registered and `true` is returned when 1 or more records are registered.
- When numeric value type is specified as return value, number of registered records is returned.

- Define SQL in the mapping file.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
    "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.example.domain.repository.todo.TodoRepository">

    <!-- (2) -->
    <insert id="create" parameterType="Todo">
        INSERT INTO
            t_todo
        (
            todo_id,
            todo_title,
            finished,
            created_at,
            version
        )
        /* (3) */
        VALUES
```

```
        (
            #{todoId},
            #{todoTitle},
            #{finished},
            #{createdAt},
            #{version}
        )
    </insert>

</mapper>
```

Sr. No.	Description
2.	Implement the INSERT SQL in the insert element. Specify name of the method defined in Repository interface, in `id` attribute. For details of `insert` element, refer to "MyBatis3 REFERENCE DOCUMENTATION (Mapper XML Files-insert, update and delete-)".
3.	Specify configuration value at the time of record registration in VALUE clause. The value to be bound in VALUE clause is specified as the bind variable of #{variableName} format. In the above example, since JavaBean (`Todo`) is specified as an argument of Repository interface, JavaBean property name is specified in the bind variable name.

- Apply DI to Repository in Service class and call Repository interface method.

```
package com.example.domain.service.todo;

import java.util.UUID;

import javax.inject.Inject;

import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import org.terasoluna.gfw.common.date.jodatime.JodaTimeDateFactory;

import com.example.domain.model.Todo;
import com.example.domain.repository.todo.TodoRepository;

@Transactional
@Service
public class TodoServiceImpl implements TodoService {

    // (4)
    @Inject
    TodoRepository todoRepository;
```

```
    @Inject
    JodaTimeDateFactory dateFactory;

    @Override
    public Todo create(Todo todo) {
        // (5)
        todo.setTodoId(UUID.randomUUID().toString());
        todo.setCreatedAt(dateFactory.newDate());
        todo.setFinished(false);
        todo.setVersion(1);
        // (6)
        todoRepository.create(todo);
        // (7)
        return todo;
    }

}
```

Sr. No.	Description
4.	Apply DI to Repository interface in Service class.
5.	Set the value for Entity object passed in the argument based on the application requirements. In the above example, • "UUID" as an ID • "System date and time" as registration date and time" • "`false` : Incomplete" in the completion flag • "`1`" in the version are set.
6.	Call Repository interface method and register the single Entity record.
7.	Return registered Entity. When registration value is set in the Service class process, it is recommended to return the registered Entity object as return value.

Generating key

An implementation example is given in "*Registering a single Entity record*" for generating key (ID) in Service class.

However, MyBatis3 provides a mechanism for generating key in mapping file.

Note: Cases wherein key generation functionality of MyBatis3 is used

When a database function (function or ID column etc.) is used for generating key, it is recommended to use the mechanism of MyBatis3 key generation functionality.

There are two kinds of methods for generating key.

- A method wherein the result obtained by calling the function etc. provided by database, is handled as the key
- A method wherein the result obtained by calling ID column provided by database (IDENTITY type, AUTO_INCREMENT type etc.) + `Statement#getGeneratedKeys()` added by JDBC3.0 is handled as the key.

Method wherein result obtained by calling the function etc. provided by database is handled as a key, is explained first. In the example given below, H2 Database is used as the database.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
    "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.example.domain.repository.todo.TodoRepository">

    <insert id="create" parameterType="Todo">
        <!-- (1) -->
        <selectKey keyProperty="todoId" resultType="string" order="BEFORE">
            /* (2) */
            SELECT RANDOM_UUID()
        </selectKey>
        INSERT INTO
            t_todo
        (
            todo_id,
            todo_title,
            finished,
            created_at,
            version
        )
```

```
        VALUES
        (
            #{todoId},
            #{todoTitle},
            #{finished},
            #{createdAt},
            #{version}
        )
    </insert>

</mapper>
```

Sr. No.	Attribute	Description
1.	-	Implements the SQL to generate key in `selectKey` element. In the above example, UUID is fetched by using the function provided by database. For details of `selectKey`, refer to "MyBatis3 REFERENCE DOCUMENTATION (Mapper XML Files-insert, update and delete-)".
	keyProperty	Specifies the property name of Entity that stores fetched key value. In the above example, key that is generated in `todoId` property of Entity, is set.
	resultType	Specifies the type of key value to be fetched by executing the SQL.
	order	Specifies the timing when the SQL for key generation (`BEFORE`or `AFTER`) is executed. • When `BEFORE` is specified, INSERT statement is executed after the results obtained by executing SQL specified in `selectKey` element are reflected in Entity. • When `AFTER` is specified, SQL specified in `selectKey` element is executed after executing INSERT statement and fetched value is reflected in Entity.
2.	-	Implement the SQL for generating key. In the above example, the function that generates UUID of H2 Database is called and key is generated. Implementation wherein value fetched from sequence object is formatted in a string can be cited as the typical example of key generation.

Next, the method wherein result obtained by calling ID column provided by database + `Statement#getGeneratedKeys()` added by JDBC3.0ratedKeys() is handled as a key, is explained. In the example given below, H2 Database is used as the database.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
    "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.example.domain.repository.audit.AuditLogRepository">

    <!-- (3) -->
    <insert id="create" parameterType="Todo" useGeneratedKeys="true"
↪keyProperty="logId">
        INSERT INTO
            t_audit_log
        (
            level,
            message,
            created_at,
        )
        VALUES
        (
            #{level},
            #{message},
            #{createdAt},
        )
    </insert>

</mapper>
```

Sr. No.	Attribute	Description
3.	useGeneratedKeys	When `true` is specified, function that fetches key by calling ID column + `Statement#getGeneratedKeys()` can be used. For details of `useGeneratedKeys`, refer to "MyBatis3 REFERENCE DOCUMENTATION (Mapper XML Files-insert, update and delete-)".
	keyProperty	Specify property name of the Entity that stores the key value which is automatically incremented in the database. In the above example, key value fetched by `Statement#getGeneratedKeys()` in `logId` property of Entity, is set after executing INSERT statement.

Batch registration of Entity

Implementation example for registering Entity in a batch is shown below.

The methods to perform batch registration of Entity are as given below.

- Execute INSERT statement that registers multiple records at the same time.

- There is a method to use the JDBC batch update functionality

Refer to "*Using batch mode*" for details on how to use the JDBC batch update functionality.

How to execute the INSERT statement that registers multiple records at the same time, is explained below. In the example below, H2 Database is used as the database.

- Defining the method in Repository interface.

```
package com.example.domain.repository.todo;

import java.util.List;

import com.example.domain.model.Todo;

public interface TodoRepository {

    // (1)
    void createAll(List<Todo> todos);

}
```

Sr. No.	Description
1.	In the above example, `createAll` method is defined as the method to perform batch registration for a list of Todo objects specified in the argument.

- Define the SQL in mapping file.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
    "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.example.domain.repository.todo.TodoRepository">

    <insert id="createAll" parameterType="list">
        INSERT INTO
            t_todo
        (
            todo_id,
            todo_title,
            finished,
            created_at,
            version
        )
        /* (2) */
        VALUES
        /* (3) */
```

```
        <foreach collection="list" item="todo" separator=",">
        (
            #{todo.todoId},
            #{todo.todoTitle},
            #{todo.finished},
            #{todo.createdAt},
            #{todo.version}
        )
        </foreach>
    </insert>

</mapper>
```

Sr. No.	Attribute	Description
2.	-	Specifies the configuration value at the time of registering records in VALUE clause.
3.	-	Repeats the process for the list of Todo objects passed as argument, by using `foreach` element. For details of `foreach` details, refer to "MyBatis3 REFERENCE DOCUMENTATION (Dynamic SQL-foreach-)".
	collection	Specifies the collection for processing. In the example given above, the process is repeated for the list of Repository method arguments. When `@Param` is not specified in the argument of Repository method, `"list"` is specified. When `@Param` is specified, the value specified in `value` attribute of `@Param` is specified.
	item	Specifies the local variable name that retains one element from the list. JavaBean property can be accessed from the SQL in `foreach` element, in #{Local variable name.Property name} format.
	separator	Specifies the string to separate elements in the list. In the above example, by specifying `","`, the VALUE clause for each element is separated with `","`.

Note: ** Precautions when using SQL that registers multiple records at the same time**

When SQL that registers multiple records concurrently is executed, "*Generating key*" described earlier cannot be used.

- Following SQL is generated and executed.

```
INSERT INTO
    t_todo
(
    todo_id,
    todo_title,
    finished,
    created_at,
    version
)
VALUES
(
    '99243507-1b02-45b6-bfb6-d9b89f044e2d',
    'todo title 1',
    false,
    '09/17/2014 23:59:59.999',
    1
)
,
(
    '66b096f1-791f-412f-9a0a-ee4a3a9186c2',
    'todo title 2',
    0,
    '09/17/2014 23:59:59.999',
    1
)
```

Tip: The support status and syntax for the SQL that performs batch registration differ depending on database and version. The links for reference pages of major databases are given below.

- Oracle 12c
- DB2 11.1
- PostgreSQL 9.6
- MySQL 5.7

Update process of Entity

Entity update method for different purposes is explained with implementation example.

Updating a single Entity

Implementation example for updating a single Entity is given below.

Note: Hereafter, an implementation example is explained wherein optimistic locking is performed by using version column. However, process related to optimistic locking need not be performed when optimistic locking is not required.

Refer to "*Exclusive Control* for details of exclusive control.

- Defining the method in Repository interface.

```
package com.example.domain.repository.todo;

import com.example.domain.model.Todo;

public interface TodoRepository {

    // (1)
    boolean update(Todo todo);

}
```

Sr. No.	Description
1.	In the above example, `update` method is defined as the method to update single Todo object specified in the argument.

Note: Return value of the method that updates a single Entity

The return value of the method that updates single Entity can be `boolean`.

However, when multiple records are obtained as update result and it is necessary to handle it as data mismatch error, numeric value type (`int` or `long`) needs to be specified as return value and it needs to be checked that a single update record exists. When main key is used as the update condition, return value can be set as `boolean` since multiple records are not obtained as update result.

- When `boolean` is specified as return value, `false` is returned when update records are 0 and `true`is returned when update records are 1 or more.
- When numeric value is specified as return value, number of update records is returned.

- Defining SQL in mapping file.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
    "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.example.domain.repository.todo.TodoRepository">

    <!-- (2) -->
    <update id="update" parameterType="Todo">
        UPDATE
            t_todo
        SET
            todo_title = #{todoTitle},
            finished = #{finished},
            version = version + 1
        WHERE
            todo_id = #{todoId}
        AND
            version = #{version}
    </update>

</mapper>
```

Sr. No.	Description
2.	Implement the UPDATE SQL in `update` element. Specify the method name defined in Repository interface, in `id` attribute. For details of `update` element, refer to "MyBatis3 REFERENCE DOCUMENTATION (Mapper XML Files-insert, update and delete-)". The value to be bound in SET clause and WHERE clause is specified as the bind variable with #{variableName} format. In the above example, since a JavaBean (`Todo`) is specified as argument of Repository interface, JavaBean property name is specified in the bind variable name.

- Apply DI to Repository in Service class and call Repository interface method.

```
package com.example.domain.service.todo;

import javax.inject.Inject;

import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
```

```
import com.example.domain.model.Todo;
import com.example.domain.repository.todo.TodoRepository;

@Transactional
@Service
public class TodoServiceImpl implements TodoService {

    // (3)
    @Inject
    TodoRepository todoRepository;

    @Override
    public Todo update(Todo todo) {

        // (4)
        Todo currentTodo = todoRepository.findOne(todo.getTodoId());
        if (currentTodo == null || currentTodo.getVersion() != todo.
↪getVersion()) {
            throw new ObjectOptimisticLockingFailureException(Todo.class, todo
                    .getTodoId());
        }

        // (5)
        currentTodo.setTodoTitle(todo.getTodoTitle());
        currentTodo.setFinished(todo.isFinished());

        // (6)
        boolean updated = todoRepository.update(currentTodo);
        // (7)
        if (!updated) {
            throw new ObjectOptimisticLockingFailureException(Todo.class,
                    currentTodo.getTodoId());
        }
        currentTodo.setVersion(todo.getVersion() + 1);

        return currentTodo;
    }

}
```

Sr. No.	Description
3.	Apply DI to Repository interface in Service class.
4.	Fetch the Entity to be updated from database. In the above example, when Entity is updated (records are deleted or version is updated), optimistic locking exception (`org.springframework.orm.ObjectOptimisticLockingFailureException`) provided by Spring Framework is generated.
5.	Reflect update details for the Entity to be updated. In the above example, "Title" and "Complete flag" are reflected. When there are few update items, the process can be performed as per the implementation example given above. However, when update items are more in number, it is recommended to use "*Bean Mapping (Dozer)*".
6.	Call the Repository interface method and update single Entity record.
7.	Determine update results of Entity. In the above example, when Entity is not updated (records are deleted or version is updated), optimistic locking exception (`org.springframework.orm.ObjectOptimisticLockingFailureException`) provided by Spring Framework is generated.

Tip: In the above example, when update process is successful,

```
currentTodo.setVersion(todo.getVersion() + 1);
```

is obtained.

It is a process to combine the version updated in database and the version that stores an Entity.

If database status and Entity status are not matched when referring a version in the call source (Controller or JSP etc.) process, data mismatch may occur and application is not executed as anticipated.

Batch update of Entity

Implementation example wherein Entity is updated in batch is given below.

The methods to update Entity in batch are as below.

- Execute UPDATE statement that updates multiple records simultaneously
- Use JDBC batch update functionality

Refer to "*Using batch mode*" for the details on how to use the JDBC batch update functionality.

How to execute UPDATE statement that concurrently updates multiple records is explained here.

- Defining the method in Repository interface.

```
package com.example.domain.repository.todo;

import com.example.domain.model.Todo;
import org.apache.ibatis.annotations.Param;

import java.util.List;

public interface TodoRepository {

    // (1)
    int updateFinishedByTodIds(@Param("finished") boolean finished,
                               @Param("todoIds") List<String> todoIds);

}
```

Sr. No.	Description
1.	In the above example, `updateFinishedByTodIds` method is defined as the method to update `finished` column of the records corresponding to list of IDs specified in the argument.

Note: Return value for the method that updates Entity in batch

Return value of the method that updates Entity in batch should preferably be of numeric type (`int` or `long`). When it is set as numeric type return value, number of updated records can be fetched.

- Defining the SQL in mapping file.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
    "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.example.domain.repository.todo.TodoRepository">

    <update id="updateFinishedByTodIds">
        UPDATE
            t_todo
        SET
```

```
            finished = #{finished},
            /* (2) */
            version = version + 1
        WHERE
            /* (3) */
            <foreach item="todoId" collection="todoIds"
                     open="todo_id IN (" separator="," close=")">
                #{todoId}
            </foreach>
    </update>

</mapper>
```

Sr. No.	Attribute	Description
2.	-	Updates version column when an optimistic locking is applied using a version column. If the version is not updated, optimistic locking control does not operate properly. Refer to "*Exclusive Control*" for details of exclusive control.
3.	-	Specifies the update conditions for updating multiple records in WHERE clause.
	-	Repeats process for the list of IDs passed by argument, using `foreach` element. In the above example, IN clause is generated from the list of IDs passed by argument. Refer "MyBatis3 REFERENCE DOCUMENTATION (Dynamic SQL-foreach-)" for details of `foreach`.
	collection	Specify collection for a process. In the above example, the process is repeated for a list of IDs (`todoIds`) of Repository method arguments.
	item	Specify local variable name that retains 1 element in the list.
	separator	Specify the string for separating elements in the list. In the above example, `","`, which is the separator character of IN clause, is specified.

Delete process for Entity

Deleting a single Entity

Implementation example for deleting a single Entity is given below.

Note: Explanation hereafter shows an implementation example wherein an optimistic locking is performed by using version column. However, it is not necessary to perform the processes related to optimistic locking

when optimistic locking is not required.

Refer to "*Exclusive Control*" for details of exclusive control.

- Defining method in Repository interface.

```
package com.example.domain.repository.todo;

import com.example.domain.model.Todo;

public interface TodoRepository {

    // (1)
    boolean delete(Todo todo);

}
```

Sr. No.	Description
1.	In the above example, `delete` method is defined as the method to delete single Todo object specified in the argument.

Note: ** Return value for the method that deletes a single Entity**

The return value of the method that deletes a single Entity can be `boolean`.

However, when the return value is to be handled as a data mismatch error due to multiple deletion results, it is necessary to set numeric value type (`int` or `long`) as the return value and to check whether a single deletion record exists. When main key is used as the delete condition, return value can be set to `boolean` since, multiple deleted records are not obtained.

- When `boolean` is specified as return value, `false`is returned when deleted records are 0 and `true` is returned when deleted records are 1 or more.
- When numeric value type is specified as a return value, the number of deleted records is returned.

- Defining a SQL in mapping file.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
    "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.example.domain.repository.todo.TodoRepository">

    <!-- (2) -->
    <delete id="delete" parameterType="Todo">
        DELETE FROM
            t_todo
        WHERE
            todo_id = #{todoId}
        AND
            version = #{version}
    </delete>

</mapper>
```

Sr. No.	Description
2.	Implement DELETE SQL in `delete` element. Specify method name for the method defined in Repository interface, in `id` attribute. Refer to "MyBatis3 REFERENCE DOCUMENTATION (Mapper XML Files-insert, update and delete-)" for details of `delete` element. The value to be bound in WHERE clause is specified as a bind variable of #{variableName} format. In the above example, JavaBean property name is specified in the bind variable name since a JavaBean (`Todo`) is specified as argument of Repository interface.

- Apply DI to the Repository in Service class and call the method for Repository interface.

```
package com.example.domain.service.todo;

import javax.inject.Inject;

import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import com.example.domain.model.Todo;
import com.example.domain.repository.todo.TodoRepository;

@Transactional
@Service
public class TodoServiceImpl implements TodoService {

    // (3)
```

```
    @Inject
    TodoRepository todoRepository;

    @Override
    public Todo delete(String todoId, long version) {

        // (4)
        Todo currentTodo = todoRepository.findOne(todoId);
        if (currentTodo == null || currentTodo.getVersion() != version) {
            throw new ObjectOptimisticLockingFailureException(Todo.class,␣
↪todoId);
        }

        // (5)
        boolean deleted = todoRepository.delete(currentTodo);
        // (6)
        if (!deleted) {
            throw new ObjectOptimisticLockingFailureException(Todo.class,
                    currentTodo.getTodoId());
        }

        return currentTodo;
    }

}
```

Sr. No.	Description
3.	Apply DI to Repository interface in Service class.
4.	Fetch the Entity to be deleted from database. In the above example, when Entity is updated (records are deleted or version is updated), optimistic locking exception (`org.springframework.orm.ObjectOptimisticLockingFailureException`) provided by Spring Framework is generated.
5.	Call Repository interface method and delete a single Entity.
6.	Determine the deletion result of Entity. In the above example, when Entity is not deleted (records are deleted or version is updated) optimistic locking exception (`org.springframework.orm.ObjectOptimisticLockingFailureException`) provided by Spring Framework is generated.

Batch deletion of Entity

Implementation example wherein Entity is deleted in batch is given below.

The methods to delete Entity in batch are as given below.

- Execute a DELETE statement that concurrently deletes multiple records
- Use JDBC batch update functionality

Refer to "*Using batch mode*" for how to use JDBC batch update functionality.

The method to execute DELETE statement that concurrently deletes multiple records, is explained below.

- Defining method in Repository interface.

```
package com.example.domain.repository.todo;

public interface TodoRepository {

    // (1)
    int deleteOlderFinishedTodo(Date criteriaDate);

}
```

Sr. No.	Description
1.	In the above example, `deleteOlderFinishedTodo` method is defined as the method to delete finished records that were created prior to the standard date.

Note: Return value for the method that deletes Entity in batch

Return value for the method that deletes Entity in batch can be numeric value type (`int`or `long`). When the return value is of numeric type, number of deleted records can be fetched.

- Defining SQL in mapping file.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
```

```
    "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.example.domain.repository.todo.TodoRepository">

    <delete id="deleteOlderFinishedTodo" parameterType="date">
        <![CDATA[
        DELETE FROM
            t_todo
        /* (2) */
        WHERE
            finished = TRUE
        AND
            created_at  < #{criteriaDate}
        ]]>
    </delete>

</mapper>
```

Sr. No.	Description
3.	Specify delete conditions for updating multiple records in WHERE clause. In the above example, • Finished (`finished` is `TRUE`) • Created before standard date (`created_at` before standard date) are specified as delete conditions.

Implementing dynamic SQL

An implementation example wherein dynamic SQL is built, is given below.

MyBatis3 provides a mechanism through which dynamic SQL is built by using OGNL base expression (Expression language) and XML elements for building dynamic SQL.

Refer to "MyBatis3 REFERENCE DOCUMENTATION (Dynamic SQL) <http://mybatis.github.io/mybatis-3/dynamic-sql.html>'_" for details of dynamic SQL.

MyBatis3 provides following XML elements for building a dynamic SQL.

Sr. No.	Element name	Description
1.	`if`	Element that builds SQL only when it matches with the condition.
2.	`choose`	Element that builds SQL by selecting one of the options from multiple options, that matches with the condition.
3.	`where`	Element that assigns or removes prefix and suffix for the built WHERE clause.
4.	`set`	Element that assigns or removes prefix or suffix for the built SET clause.
5.	`foreach`	Element that repeats a process for a collection or an array
6.	`bind`	Element that stores the results of OGNL expression in the variable. Variable stored by using `bind` variable can be referred in SQL.

Tip: Although it is not given in the list, `trim` element is provided as the XML element for building dynamic SQL.

`trim` element is a more generalized XML element as compared to `where` element and `set` element.

In most of the cases, `where` element and `set` element can meet the requirements. Hence, description of `trim` element is omitted in this guideline. Refer to "MyBatis3 REFERENCE DOCUMENTATION (Dynamic SQL-trim, where, set-)" when it is necessary to use `trim` element.

Implementation of if element

`if` element is the XML element that builds SQL only when it matches with specified conditions.

```
<select id="findAllByCriteria" parameterType="TodoCriteria" resultType="Todo">
    SELECT
        todo_id,
        todo_title,
        finished,
        created_at,
        version
    FROM
        t_todo
```

```
    WHERE
        todo_title LIKE #{todoTitle} || '%' ESCAPE '~'
    <!-- (1) -->
    <if test="finished != null">
        AND
            finished = #{finished}
    </if>
    ORDER BY
        todo_id
</select>
```

Sr. No.	Description
1.	Specify the condition in `test` attribute of `if` element. In the above example, when `finished` is specified as the search condition, conditions for `finished` column are added to SQL.

SQL (WHERE clause) generated by dynamic SQL described above consists of 2 patterns.

```
-- (1) finished == null
...
WHERE
    todo_title LIKE ? || '%' ESCAPE '~'
ORDER BY
    todo_id
```

```
-- (2) finished != null
...
WHERE
    todo_title LIKE ? || '%' ESCAPE '~'
AND
    finished = ?
ORDER BY
    todo_id
```

Implementation of choose element

`choose` element is the XML element for building SQL by selecting one option that matches the condition from a set of conditions.

```
<select id="findAllByCriteria" parameterType="TodoCriteria" resultType="Todo">
    SELECT
        todo_id,
        todo_title,
```

```
        finished,
        created_at,
        version
    FROM
        t_todo
    WHERE
        todo_title LIKE #{todoTitle} || '%' ESCAPE '~'
    <!-- (1) -->
    <choose>
        <!-- (2) -->
        <when test="createdAt != null">
            AND
                created_at <![CDATA[ > ]]> #{createdAt}
        </when>
        <!-- (3) -->
        <otherwise>
            AND
                created_at <![CDATA[ > ]]> CURRENT_DATE
        </otherwise>
    </choose>
    ORDER BY
        todo_id
</select>
```

Sr. No.	Description
1.	Specify the condition to build SQL by specifying `when` element and `otherwise` element in `choose` element.
2.	Specify the condition in `test` attribute of `when` element. In the above example, when `createdAt` is specified as a search condition, a condition wherein, values of `create_at`column extract the records after the specified date, is added to SQL.
3.	Specify all the SQLs in ‘‘otherwise‘‘ element that are built when the conditions do not match with `when` element . In the above example, the condition wherein `create_at` column values extract the records after the current date (records that are created on that day) is added to SQL.

SQL (WHERE clause) that is generated by dynamic SQL described above consists of 2 patterns.

```
-- (1) createdAt!=null
...
WHERE
    todo_title LIKE ? || '%' ESCAPE '~'
AND
    created_at   >   ?
ORDER BY
    todo_id
```

```
-- (2) createdAt==null
...
WHERE
    todo_title LIKE ? || '%' ESCAPE '~'
AND
    created_at > CURRENT_DATE
ORDER BY
    todo_id
```

Implementation of where element

`where` element is the XML element for dynamically generating WHERE clause.

When `where` element is used,

- Assigning WHERE clause
- Removal of AND clause and OR clause

are performed. Hence, WHERE clause can be built easily.

```
<select id="findAllByCriteria2" parameterType="TodoCriteria" resultType="Todo
↪">
    SELECT
        todo_id,
        todo_title,
        finished,
        created_at,
        version
    FROM
        t_todo
    <!-- (1) -->
    <where>
        <!-- (2) -->
        <if test="finished != null">
            AND
                finished = #{finished}
        </if>
        <!-- (3) -->
        <if test="createdAt != null">
            AND
                created_at <![CDATA[ > ]]> #{createdAt}
        </if>
    </where>
```

```
    ORDER BY
        todo_id
</select>
```

Sr. No.	Description
1.	Implement the dynamic SQL for building WHERE clause in `where` element. According to the SQL built in `where` element, processes such as assigning WHERE clause, removing AND clause and OR clause etc. can be performed.
2.	Build dynamic SQL. In the above example, when `finished` is specified as a search condition, the condition for `finished` column is added to SQL.
3.	Build dynamic SQL. In the above example, when `createdAt` is specified as a search condition, the condition for `created_at` column is added to SQL.

The SQL (WHERE clause) that is generated by dynamic SQL described above consists of 4 patterns as given below.

```
-- (1) finished != null && createdAt != null
...
FROM
    t_todo
WHERE
    finished = ?
AND
    created_at  >  ?
ORDER BY
    todo_id
```

```
-- (2) finished != null && createdAt == null
...
FROM
    t_todo
WHERE
    finished = ?
ORDER BY
    todo_id
```

```
-- (3) finished == null && createdAt != null
...
FROM
    t_todo
WHERE
    created_at  >  ?
ORDER BY
    todo_id
```

```
-- (4) finished == null && createdAt == null
...
FROM
    t_todo
ORDER BY
    todo_id
```

Implementation example for set element

`set` element is the XML element for automatically generating SET clause.

When `set` element is used,

- Assigning SET clause
- Removal of comma at the end

are performed. Hence SET clause can be easily built.

```
<update id="update" parameterType="Todo">
    UPDATE
        t_todo
    <!-- (1)  -->
    <set>
        version = version + 1,
        <!-- (2) -->
        <if test="todoTitle != null">
            todo_title = #{todoTitle}
        </if>
    </set>
    WHERE
        todo_id = #{todoId}
</update>
```

Sr. No.	Description
1.	Implement the dynamic SQL to build SET clause in `set` element. Assigning SET clause and removal of comma at the end is performed according to the SQL that is built in `set` element.
2.	Build a dynamic SQL. In the above example, when `todoTitle` is specified as an update item, `todo_title` column is added to SQL as an update column.

SQL generated by dynamic SQL described above consists of 2 patterns as below.

```
-- (1) todoTitle != null
UPDATE
    t_todo
SET
    version = version + 1,
    todo_title = ?
WHERE
    todo_id = ?
```

```
-- (2) todoTitle == null
UPDATE
    t_todo
SET
   version = version + 1
WHERE
    todo_id = ?
```

Implementation example of foreach element

`foreach` element is the XML element for repeating a process for a collection or an array.

```
<select id="findAllByCreatedAtList" parameterType="list" resultType="Todo">
    SELECT
        todo_id,
        todo_title,
        finished,
        created_at,
        version
    FROM
        t_todo
    <where>
        <!-- (1) -->
        <if test="list != null">
            <!-- (2) -->
            <foreach collection="list" item="date" separator="OR">
            <![CDATA[
                (created_at >= #{date} AND created_at < DATEADD('DAY', 1, #
↪{date}))
            ]]>
            </foreach>
        </if>
    </where>
    ORDER BY
```

```
        todo_id
</select>
```

Sr. No.	Attribute	Description
1.	-	Performs `null` check for the collection or array for which process is repeated. Null check is not necessary when `null` value is not obtained.
2.	-	Repeat the process for the collection or array and build the dynamic SQL, by using `foreach` element. In the above example, WHERE clause is built for searching the record wherein date of record creation matches with any of the specified dates (date list).
	collection	Specify the collection or array for which a process is repeated, in `collection` attribute. In the above example, collection specified in the Repository method argument is specified.
	item	Specify the local variable name that retains one element in the list, in `item` attribute. In the above example, since the date list is specified in `collection` attribute, variable name called `date` is specified.
	separator	Specify the separator string between elements in `separator` attribute. In the above example, WHERE clause of OR condition is built.

Tip: `foreach` element consists of following attributes although they are not used in the above example.

Sr. No.	Attribute	Description
1.	open	Specify the string that is set before processing the elements that are at the beginning of the collection.
2.	close	Specify the string that is set after processing the elements at the end of the collection.
3.	index	Specify the variable name that stores the loop number.

There are only a few cases that use `index` attribute, however `open` attribute and `close` attribute are used to generate IN clause etc. dynamically.

How to use `foreach`element while creating an IN clause is explained.

```
<foreach collection="list" item="statusCode"
        open="AND order_status IN ("
        separator=","
        close=")">
```

```
    #{statusCode}
</foreach>
```

Following SQL is built.

```
-- list=['accepted','checking']
...
AND order_status IN (?,?)
```

SQL (WHERE clause) generated by dynamic SQL described above consists of following 3 patterns.

```
-- (1) list=null or statusCodes=[]
...
FROM
    t_todo
ORDER BY
    todo_id
```

```
-- (2) list=['2014-01-01']
...
FROM
    t_todo
WHERE
    (created_at >= ? AND created_at < DATEADD('DAY', 1, ?))
ORDER BY
    todo_id
```

```
-- (3) list=['2014-01-01','2014-01-02']
...
FROM
    t_todo
WHERE
    (created_at >= ? AND created_at < DATEADD('DAY', 1, ?))
OR
    (created_at >= ? AND created_at < DATEADD('DAY', 1, ?))
ORDER BY
    todo_id
```

Implementation example for bind element

`bind` element is the XML element for storing OGNL expression result in the variable.

```
<select id="findAllByCriteria" parameterType="TodoCriteria" resultType="Todo">
    <!-- (1) -->
    <bind name="escapedTodoTitle"
          value="@org.terasoluna.gfw.common.query.
↪QueryEscapeUtils@toLikeCondition(todoTitle)" />
    SELECT
        todo_id,
        todo_title,
        finished,
        created_at,
        version
    FROM
        t_todo
    WHERE
        /* (2) */
        todo_title LIKE #{escapedTodoTitle} || '%' ESCAPE '~'
    ORDER BY
        todo_id
</select>
```

Sr. No.	Attribute	Description
1.	-	Store results of OGNL expression in the variable, using `bind` element. In the above example, the results obtained by calling the method using OGNL expression, are stored in the variable.
	name	Specify variable name in `name` attribute. The variable specified here can be used as SQL bind variable.
	value	Specify OGNL expression in `value` attribute. Results obtained by executing OGNL expression are stored in the variable specified by `name` attribute. In the above example, the results obtained by calling method (`QueryEscapeUtils#toLikeCondition(String)`) provided by common library are stored in the variable `escapedTodoTitle`.
2.	-	Specify the variable created by using `bind` element as the bind variable. In the above example, the variable created by using `bind` element (`escapedTodoTitle`) is specified as the bind variable.

Tip: In the above example, although the variable created by using `bind` variable is specified as the bind variable, it can also be used as substitution variable.

Refer to "*SQL Injection countermeasures*" for bind variable and substitution variable.

Escape during LIKE search

When performing LIKE search, the value to be used as search condition should be escaped for LIKE search.

The escape process for LIKE search can be implemented by using the method of `org.terasoluna.gfw.common.query.QueryEscapeUtils` class provided by common library.

Refer to "*Escaping during LIKE search*" for specifications of the escape process provided by common library.

```
<select id="findAllByCriteria" parameterType="TodoCriteria" resultType="Todo">
    <!-- (1) -->
    <bind name="todoTitleContainingCondition"
          value="@org.terasoluna.gfw.common.query.
↪QueryEscapeUtils@toContainingCondition(todoTitle)" />
    SELECT
        todo_id,
        todo_title,
        finished,
        created_at,
        version
    FROM
        t_todo
    WHERE
        /* (2) (3) */
        todo_title LIKE #{todoTitleContainingCondition} ESCAPE '~'
    ORDER BY
        todo_id
</select>
```

Sr. No.	Description
1.	Call the Escape process method for LIKE search provided by common library, by using `bind` element (OGNL expression). In the above example, escape process is performed for partial match and is stored in `todoTitleContainingCondition` variable. `QueryEscapeUtils@toContainingCondition(String)` is the method that assigns "`%`" before and after the escaped string.
2.	Specify the string that performs escape process for partial match, as bind variable of LIKE clause.
3.	Specify escape character in ESCAPE clause. Since `"~"` is used as an escape character in the Escape process provided by common library, `'~'` is specified in ESCAPE clause.

Tip: In the above example, a method that performs the Escape process for partial match is called. However, methods that perform the following processes are also provided.

- Escape for starting-with match (`QueryEscapeUtils@toStartingWithCondition(String)`)
- Escape for ends-with match (`QueryEscapeUtils@toEndingWithCondition(String)`)
- Escape only (`QueryEscapeUtils@toLikeCondition(String)`)

Refer to "*Escaping during LIKE search*" for details.

Note: In the above example, the method that performs Escape process in the mapping file is called, however Escape process can also be called as a Service process before calling the Repository method.

As role of component, it is appropriate that Escape process is performed in mapping file. Hence, in this guideline, it is recommended to perform Escape process in mapping file.

SQL Injection countermeasures

It is important to take precautions when building SQL to avoid occurrence of SQL Injection.

MyBatis3 provides following two methods as the mechanism to embed values in SQL.

Sr. No.	Method	Description
1.	Embedding value by using bind variable	When this method is used, the value is embedded after building SQL by using `java.sql.PreparedStatement` . Hence, **the value can be safely embedded.** **When the value entered by user is to be embedded in SQL, as a rule, bind variable should be used.**
2.	Embedding value by using substitution variable	When this method is used, the value is substituted as a string while building SQL. Hence **Safe embedding of value cannot be guaranteed.**

Warning: When the value entered by the user is embedded by using substitution variable, it should be noted that the risk of SQL Injection is high.

When the value entered by the user needs to be embedded by using a substitution variable, the input check must be performed in order to ensure that SQL Injection has not occurred.

Basically, **it is strongly recommended not to use the value entered by the user as it is.**

How to embed using a bind variable

How to use a bind variable is shown below.

```
<insert id="create" parameterType="Todo">
    INSERT INTO
        t_todo
    (
        todo_id,
        todo_title,
        finished,
        created_at,
        version
    )
    VALUES
    (
        /* (1) */
        #{todoId},
        #{todoTitle},
        #{finished},
        #{createdAt},
        #{version}
    )
</insert>
```

Sr. No.	Description
1.	Enclose the property name of the property that stores bind value using `#{` and `}` and specify it as bind variable.

Tip: A number of attributes can be specified in the bind variable.

The attributes that can be specified are as below.

- javaType
- jdbcType
- typeHandler
- numericScale
- mode
- resultMap
- jdbcTypeName

Basically, MyBatis simply selects an appropriate behavior just by specifying the property name. The attributes described above can be specified when MyBatis does not select an appropriate behavior.

Refer to "MyBatis3 REFERENCE DOCUMENTATION(Mapper XML Files-Parameters-) " for how to use attributes.

How to embed using a substitution variable

How to use a substitution variable is described below.

- Defining the method in Repository interface.

```
public interface TodoRepository {
    List<Todo> findAllByCriteria(@Param("criteria") TodoCriteria criteria,
                                 @Param("direction") String direction);
}
```

- Implementing SQL in mapping file.

```
<select id="findAllByCriteria" parameterType="TodoCriteria" resultType="Todo">
    <bind name="todoTitleContainingCondition"
```

```
        value="@org.terasoluna.gfw.common.query.
↪QueryEscapeUtils@toContainingCondition(criteria.todoTitle)" />
    SELECT
        todo_id,
        todo_title,
        finished,
        created_at,
        version
    FROM
        t_todo
    WHERE
        todo_title LIKE #{todoTitleContainingCondition} ESCAPE '~'
    ORDER BY
        /* (1) */
        todo_id ${direction}
</select>
```

Sr. No.	Description
1.	Enclose the property name of the property that stores the value to be substituted by `${` and `}` and specify as substitution variable. In the above example, `${direction}` part is substituted by `"DESC"` or `"ASC"` .

Warning:

Embedding value by a substitution variable must be used only after ensuring that the value is safe for the application and by restricting its use to table name, column name and sort conditions.

For example, the pair of code value and value to be embedded in SQL is stored in `Map` as shown below.

```
Map<String, String> directionMap = new HashMap<String, String>();
directionMap.put("1", "ASC");
directionMap.put("2", "DESC");
```

The value entered should be handled as code value and is expected to be converted to a safe value inside the process executing the SQL.

```
String direction = directionMap.get(directionCode);
todoRepository.findAllByCriteria(criteria, direction);
```

In the above example, `Map` is used. However, "*Codelist* " provided by common library can also be used. If "*Codelist* " is used, the value entered can be checked. Hence, the value can be safely embedded.

- `projectName-domain/src/main/resources/META-INF/spring/projectName-codelist.xm`

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd">

    <bean id="CL_DIRECTION" class="org.terasoluna.gfw.common.codelist.
↪SimpleMapCodeList">
        <property name="map">
            <map>
                <entry key="1" value="ASC" />
                <entry key="2" value="DESC" />
            </map>
        </property>
    </bean>
</beans>
```

- Service class

```
@Inject
@Named("CL_DIRECTION")
CodeList directionCodeList;

// ...

public List<Todo> searchTodos(TodoCriteria criteria, String␣
↪directionCode){
    String direction = directionCodeList.asMap().get(directionCode);
    List<Todo> todos = todoRepository.findAllByCriteria(criteria,␣
↪direction);
    return todos;
}
```

6.2.3 How to extend

Sharing SQL statement

How to share a SQL statement in multiple SQLs is explained.

In MyBatis3, SQL statement (or a part of SQL statement) can be shared by using `sql` element and `include` element.

Note: How to use a shared SQL statement

When pagination search is to be performed, WHERE clause of "SQL that fetches total records of Entity matching with search conditions" and "SQL that fetches a list of Entities matching with search conditions" should be shared.

Implementation example of mapping file is as given below.

```
<!-- (1)  -->
<sql id="findPageByCriteriaWherePhrase">
    <![CDATA[
    WHERE
        todo_title LIKE #{title} || '%' ESCAPE '~'
    AND
        created_at < #{createdAt}
    ]]>
</sql>

<select id="countByCriteria" resultType="_long">
    SELECT
        COUNT(*)
    FROM
        t_todo
    <!-- (2)  -->
    <include refid="findPageByCriteriaWherePhrase"/>
</select>

<select id="findPageByCriteria" resultType="Todo">
    SELECT
        todo_id,
        todo_title,
        finished,
        created_at,
        version
    FROM
        t_todo
    <!-- (2)  -->
    <include refid="findPageByCriteriaWherePhrase"/>
    ORDER BY
        todo_id
</select>
```

Sr. No.	Description
1.	Implement SQL statement to be shared by multiple SQLs, in `sql` element. Specify an ID unique to the mapping file, in `id` attribute.
2.	Specify the INCLUDE SQL by using `include` element. Specify the INCLUDE SQL ID (value specified in `id` attribute of `sql` element), in `refid` attribute.

Implementation of TypeHandler

Creation of a unique `TypeHandler` is required when mapping with Joda-Time class not supported by MyBatis3 standards, is necessary.

This guideline explains how to implement `TypeHandler` using "*Implementing TypeHandler for Joda-Time*" as an example.

Refer to "*TypeHandler settings*" for how to apply a created `TypeHandler` in an application.

Note: Regarding implementation for BLOB and CLOB

`TypeHandler` added by MyBatis 3.4 implements conversion of BLOB and `java.io.InputStream` , and CLOB と `java.io.Reader` by using API added by JDBC 4.0 (Java 1.6). In case of a JDBC driver of JDBC 4.0 support, it is not necessary to implement a new `TypeHandler` since type handler for the conversion of BLOB ⇔ `InputStream` and CLOB ⇔ `Reader` is enabled by default.

If a JDBC driver incompatible with JDBC 4.0 is used, a `TypeHandler` should be created considering compatibility version of JDBC driver to be used.

For example, some methods added from JDBC 4.0 are not yet implemented in JDBC driver (`postgresql-9.4-1212.jar`) for PostgreSQL9.4.

Implementing the TypeHandler for CLOB

MyBatis3 provides a `TypeHandler` for mapping CLOB in `java.lang.String`. However, when the data to be handled is very large, it is necessary to map it in `java.io.Reader`.

How to implement the `TypeHandler` for mapping CLOB in `java.io.Reader`is given below.

```
package com.example.infra.mybatis.typehandler;

import org.apache.ibatis.type.BaseTypeHandler;
import org.apache.ibatis.type.JdbcType;

import java.io.Reader;
import java.sql.*;

// (1)
public class ClobReaderTypeHandler extends BaseTypeHandler<Reader> {

    // (2)
    @Override
    public void setNonNullParameter(PreparedStatement ps, int i, Reader␣
↪parameter,
                                    JdbcType jdbcType) throws SQLException {
        ps.setClob(i, parameter);
    }

    // (3)
    @Override
    public Reader getNullableResult(ResultSet rs, String columnName)
        throws SQLException {
        return toReader(rs.getClob(columnName));
    }

    // (3)
    @Override
    public Reader getNullableResult(ResultSet rs, int columnIndex)
        throws SQLException {
        return toReader(rs.getClob(columnIndex));
    }

    // (3)
    @Override
    public Reader getNullableResult(CallableStatement cs, int columnIndex)
        throws SQLException {
        return toReader(cs.getClob(columnIndex));
    }

    private Reader toReader(Clob clob) throws SQLException {
        // (4)
        if (clob == null) {
            return null;
        } else {
            return clob.getCharacterStream();
        }
    }

}
```

Sr. No.	Description
1.	Specify `BaseTypeHandler` provided by MyBatis3 in parent class. In such cases, specify `Reader` in generic type of `BaseTypeHandler`.
2.	Implement a process that sets `Reader` in `PreparedStatement`.
3.	Fetch `Reader` from `Clob` that is fetched from `ResultSet` or `CallableStatement` and return it as the return value.
4.	Since fetched `Clob` can become `null` in the column that allows `null`, `Reader` needs to be fetched only after performing `null` check. In the implementation example described above, a private method is created since same process is required for all three methods.

Implementing TypeHandler for Joda-Time

MyBatis3 does not support Joda-time classes (`org.joda.time.DateTime`, `org.joda.time.LocalDateTime`, `org.joda.time.LocalDate` etc.).
Hence, when Joda-Time class is used in the field of Entity class, it is necessary to provide a `TypeHandler` for Joda-Time.

How to implement a `TypeHandler` for mapping `org.joda.time.DateTime` and `java.sql.Timestamp` is shown below.

> **Note:** Other classes provided by Joda-Time (`LocalDateTime`, `LocalDate`, `LocalTime` etc.) can also be implemented in the same way.

```
package com.example.infra.mybatis.typehandler;

import java.sql.CallableStatement;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Timestamp;

import org.apache.ibatis.type.BaseTypeHandler;
import org.apache.ibatis.type.JdbcType;
import org.joda.time.DateTime;
```

```
// (1)
public class DateTimeTypeHandler extends BaseTypeHandler<DateTime> {

    // (2)
    @Override
    public void setNonNullParameter(PreparedStatement ps, int i,
            DateTime parameter, JdbcType jdbcType) throws SQLException {
        ps.setTimestamp(i, new Timestamp(parameter.getMillis()));
    }

    // (3)
    @Override
    public DateTime getNullableResult(ResultSet rs, String columnName)
            throws SQLException {
        return toDateTime(rs.getTimestamp(columnName));
    }

    // (3)
    @Override
    public DateTime getNullableResult(ResultSet rs, int columnIndex)
            throws SQLException {
        return toDateTime(rs.getTimestamp(columnIndex));
    }

    // (3)
    @Override
    public DateTime getNullableResult(CallableStatement cs, int columnIndex)
            throws SQLException {
        return toDateTime(cs.getTimestamp(columnIndex));
    }

    private DateTime toDateTime(Timestamp timestamp) {
        // (4)
        if (timestamp == null) {
            return null;
        } else {
            return new DateTime(timestamp.getTime());
        }
    }

}
```

Sr. No.	Description
1.	Specify the `BaseTypeHandler` provided by MyBatis3 in parent class. In such cases, specify `DateTime` in the generic type of `BaseTypeHandler`.
2.	Convert `DateTime` to `Timestamp` and implement the process configured in ‘‘PreparedStatement‘‘.
3.	Convert `Timestamp` fetched from `ResultSet` or `CallableStatement` to `DateTime` and return as a return value.
4.	Since `Timestamp`can become `null` in the column that allows `null`, it needs to be converted to `DateTime` only after performing `null`check. In the implementation example described above, a private method is created since same process is required for all three methods.

Implementation of ResultHandler

MyBatis3 provides a mechanism wherein search results are processed for each record.

When this mechanism is used and processes like,

- Value fetched by DB is processed in Java process
- Values etc. fetched by DB are aggregated as Java process

are performed, the amount of memory consumed simultaneously can be restricted to a minimum.

For example, when the process is to be implemented wherein, search results are downloaded as data in CSV format, it is advisable to process the search results per record.

Note: It is strongly recommended to use this mechanism when the quantity of search results may be very large and when it is necessary to process the search result for each record at a time in Java process.

When this mechanism of processing the search result for each record is not used, all the search result data, “Size of one data record * number of search result records”, is stored in the memory at the same time, and the data cannot be marked for GC till the process is completed for entire data.

In contrast, when a mechanism wherein the search results are processed one at a time, is used, only the “size of one data record” is stored in the memory and that one data record can be marked for GC once the process for that data record is completed.

For example, when “size of one data record” is `2KB` and “number of search results” are `10,000` records, the concurrent memory consumption is as below.

- `20MB` memory is consumed while performing the process collectively
- `2KB` memory is consumed while performing the process per record

No particular issues have been observed in case of an application operated by a single thread. However, problems may occur in case of an application like Web application that is operated by multiple threads.

If the process is performed for 100 threads at the same time, the concurrent memory consumption is as below.

- `2GB` memory is consumed while performing the process collectively
- `200KB` memory is consumed while performing the process per record

The results are as below.

- When the process is performed collectively, depending on the maximum heap size specified, system failure due to memory exhaustion and performance degradation due to frequent occurrence of full GC are more likely to occur.
- When the process is performed per record, memory exhaustion or high-cost GC process can be controlled.

Please note that the numbers used above are just the guidelines and not the actual measured values.

How to implement a process wherein the search results are downloaded as CSV data is given below.

- Defining the method in Repository interface.

```
public interface TodoRepository {

    // (1) (2)
    void collectAllByCriteria(TodoCriteria criteria, ResultHandler<Todo>␣
↪resultHandler);

}
```

Sr. No.	Description
1.	Specify `org.apache.ibatis.session.ResultHandler` as an argument of the method.
2.	Specify `void` type as the return value of the method. Precaution should be taken as, `ResultHandler` is not called when a type other than `void` is specified.

- Defining SQL in mapping file.

```
<!-- (3) -->
<select id="collectAllByCriteria" parameterType="TodoCriteria" resultType=
↪"Todo">
    SELECT
        todo_id,
        todo_title,
        finished,
        created_at,
        version
    FROM
        t_todo
    <where>
        <if test="title != null">
            <bind name="titleContainingCondition"
                  value="@org.terasoluna.gfw.common.query.
↪QueryEscapeUtils@toContainingCondition(title)" />
            todo_title LIKE #{titleContainingCondition} ESCAPE '~'
        </if>
        <if test="createdAt != null">
            <![CDATA[
            AND created_at < #{createdAt}
            ]]>
        </if>
    </where>
</select>
```

Sr. No.	Description
3.	Mapping file is implemented in the same way as the normal search process.

Warning: Specifying fetchSize attribute

When a query to return a large amount of data is to be described, an appropriate value should be set in `fetchSize` attribute. `fetchSize` is a parameter which specifies data record count to be fetched in a single communication between JDBC driver and database. Note that, "default `fetchSize`" can be specified in MyBatis configuration file, in MyBatis3.3.0 and subsequent versions.

Refer "*fetchSize settings*" for details of `fetchSize`.

- Apply DI to Repository in Service class and call Repository interface method.

```
public class TodoServiceImpl implements TodoService {

    private static final DateTimeFormatter DATE_FORMATTER =
        DateTimeFormat.forPattern("yyyy/MM/dd");

    @Inject
    TodoRepository todoRepository;

    public void downloadTodos(TodoCriteria criteria,
        final BufferedWriter downloadWriter) {

        // (4)
        ResultHandler<Todo> handler = new ResultHandler<Todo>() {
            @Override
            public void handleResult(ResultContext<? extends Todo> context) {
                Todo todo = context.getResultObject();
                StringBuilder sb = new StringBuilder();
                try {
                    sb.append(todo.getTodoId());
                    sb.append(",");
                    sb.append(todo.getTodoTitle());
                    sb.append(",");
                    sb.append(todo.isFinished());
                    sb.append(",");
                    sb.append(DATE_FORMATTER.print(todo.getCreatedAt().
↪getTime()));
                    downloadWriter.write(sb.toString());
                    downloadWriter.newLine();
                } catch (IOException e) {
                    throw new SystemException("e.xx.fw.9001", e);
                }
            }
        };

        // (5)
        todoRepository.collectAllByCriteria(criteria, handler);

    }

}
```

Sr. No.	Description
4.	Generate `ResultHandler` instance. Implement the process that is performed for each record in `handleResult` method of `ResultHandler`. In the above example, `ResultHandler` implementation class is not created and `ResultHandler` is implemented as an anonymous object. Implementation class can also be created, however when it is not required to be shared by multiple processes, it need not be created.
5.	Call the method of Repository interface. When calling the method, specify `ResultHandler` instance generated in (4), in the argument. When `ResultHandler`is used, MyBatis repeats the following processes for the number of search results. • Records are fetched from search results and are mapped to JavaBean. • `handleResult(ResultContext)` method of `ResultHandler` instance is called.

Warning: Precautions while using ResultHandler

When `ResultHandler` is used, following two points should be considered.

- MyBatis3 provides a mechanism wherein the search results are stored in local cache and global binary cache to improve the efficiency of search process. However, the data returned from the method which uses `ResultHandler` as an argument, is not cached.
- When `ResultHandler` is used for the statement that maps data of multiple rows in a single Java object by using manual mapping, an object in the incomplete state (the status of related Entity object prior to mapping) can be passed.

Tip: ResultContext method

Following method is provided in `ResultContext` which is an argument of `ResultHandler#handleResult` method.

Sr. No.	Method	Description
1.	getResultObject	A method for fetching object which mapped search results.
2.	getResultCount	A method for fetching the call count of `ResultHandler#handleResult` method.
3.	stop	A method to notify MyBatis to stop the processing for the subsequent records. It is advisable to use this method when all the subsequent records are to be deleted.

A method `isStopped` is also provided in `ResultContext` .However, its description is omitted since it is used by MyBatis.

Using SQL execution mode

Following three types of modes are provided in MyBatis3 to execute SQL. Default is `SIMPLE`.

Here,

- How to use execution mode
- Precautions while using Repository of batch mode

are explained.

Refer to "*SQL execution mode settings*" for the explanation of execution mode.

Using PreparedStatement reuse mode

When the execution mode is changed from `SIMPLE` to ''REUSE'' mode, the handling of `PreparedStatement` in MyBatis changes. However, there is no change in behavior (use method) of MyBatis.

How to change the execution mode from default (`SIMPLE`) to `REUSE` is shown below.

- Settings are added to `projectName-domain/src/main/resources/META-INF/mybatis/mybatis-config`

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE configuration
        PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
```

```
        "http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>
    <settings>
        <!-- (1) -->
        <setting name="defaultExecutorType" value="REUSE"/>
    </settings>
</configuration>
```

Sr. No.	Description
1.	Change `defaultExecutorType` to `REUSE`. When the settings given above are performed, default behavior changes to Prepared-Statement reuse mode.

Using batch mode

When all the Update system methods of Mapper interface are to be called in a batch mode, execution mode can be changed to `BATCH` mode using the method same as "*Using PreparedStatement reuse mode*"

However, since batch mode has a number of constraints, in actual application development, it is presumed to be used in combination with `SIMPLE` or `REUSE` mode.

For example,

- Batch mode is used in the process wherein the highest priority is to meet performance requirements associated with update of large amount of data.
- `SIMPLE` or `REUSE` mode is used for a process wherein it is necessary to determine the update results to maintain data consistency such as in optimistic locking control etc.

> **Warning:** ** Precautions while using the execution mode in combination**
>
> When multiple execution modes are to be used in the application, **it should be noted that the execution mode cannot be changed within the same transaction.**
>
> For example, if multiple execution modes are used within the same transaction, MyBatis detects inconsistency and throws an error.
>
> This signifies that the processes below cannot be performed within the same transaction
>
> - Calling XxxRepository method in `BATCH` mode
> - Calling YyyRepository method in `REUSE` mode.

> Service or Repository acts as the transaction boundary for the application that is created based on these guidelines.
>
> Hence, **when multiple execution modes are to be used in the application, it is necessary to identify the execution mode while designing a Service or a Repository.**
>
> Transaction can be separated by specifying `@Transactional(propagation = Propagation.REQUIRES_NEW)` as method annotation for Service or Repository. Refer to "*Regarding transaction management*" for details of transaction control.

Hereafter,

- How to configure for using multiple execution modes in combination
- How to implement an application

are explained.

Settings for creating an individual batch mode Repository

When a batch mode Repository is to be created for a specific Repository, a Bean can be defined for the Repository by using `org.mybatis.spring.mapper.MapperFactoryBean` provided by MyBatis-Spring.

A Bean is registered for

- A repository of `REUSE` mode as a Repository for normal use
- A Repository of `BATCH` mode for a specific Repository

in the configuration example below.

- Bean definition is added to - `projectName-domain/src/main/resources/META-INF/spring/projectNam`

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:mybatis="http://mybatis.org/schema/mybatis-spring"
```

```
       xsi:schemaLocation="
       http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans.xsd
       http://www.springframework.org/schema/context
       http://www.springframework.org/schema/context/spring-context.xsd
       http://mybatis.org/schema/mybatis-spring
       http://mybatis.org/schema/mybatis-spring.xsd">

    <bean id="sqlSessionFactory"
          class="org.mybatis.spring.SqlSessionFactoryBean">
        <property name="dataSource" ref="dataSource"/>
        <property name="configLocation"
                  value="classpath:META-INF/mybatis/mybatis-config.xml"/>
    </bean>

    <!-- (1) -->
    <bean id="sqlSessionTemplate"
          class="org.mybatis.spring.SqlSessionTemplate">
        <constructor-arg index="0" ref="sqlSessionFactory"/>
        <constructor-arg index="1" value="REUSE"/>
    </bean>

    <mybatis:scan base-package="com.example.domain.repository"
                  template-ref="sqlSessionTemplate"/> <!-- (2) -->

    <!-- (3) -->
    <bean id="batchSqlSessionTemplate"
          class="org.mybatis.spring.SqlSessionTemplate">
        <constructor-arg index="0" ref="sqlSessionFactory"/>
        <constructor-arg index="1" value="BATCH"/>
    </bean>

    <!-- (4) -->
    <bean id="todoBatchRepository"
          class="org.mybatis.spring.mapper.MapperFactoryBean">
        <!-- (5) -->
        <property name="mapperInterface"
                  value="com.example.domain.repository.todo.TodoRepository"/>
        <!-- (6) -->
        <property name="sqlSessionTemplate" ref="batchSqlSessionTemplate"/>
    </bean>

</beans>
```

Sr. No.	Description
1.	Define the Bean for `SqlSessionTemplate` to be used in the Repository for normal use.
2.	Scan the Repository for normal use and register a Bean. Specify `SqlSessionTemplate` defined in (1), in `template-ref` attribute.
3.	Define a Bean for `SqlSessionTemplate` in order to use in the Repository of batch mode.
4.	Define a Bean for the Repository in batch mode. Specify a value that does not overlap with the Bean name of Repository scanned in (2), in `id` attribute. The Bean name of Repository scanned in (2) is a value for which the interface name is changed to "lowerCamelCase". In the above example, `TodoRepository` for batch mode is registered in a Bean called `todoBatchRepository`.
5.	Specify an interface name (FQCN) for Repository that uses a batch mode, in `mapperInterface` property.
6.	Specify `SqlSessionTemplate` for batch mode that is defined in (3), in `sqlSessionTemplate` property.

Note: If a Bean is defined for `SqlSessionTemplate`, following WARN log is output when the application is terminated.

This is because `close` method is called while terminating ApplicationContext of Spring since ''java.io.Closeable'' is inherited by `SqlSession` interface.

```
21:12:35.999 [Thread-2] WARN  o.s.b.f.s.DisposableBeanAdapter - Invocation␣
↪of destroy method 'close' failed on bean with name 'sqlSessionTemplate'
java.lang.UnsupportedOperationException: Manual close is not allowed over␣
↪a Spring managed SqlSession
    at org.mybatis.spring.SqlSessionTemplate.close(SqlSessionTemplate.java:
↪310) ~[mybatis-spring-1.2.2.jar:1.2.2]
    at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method) ~[na:1.
↪8.0_20]
    at sun.reflect.NativeMethodAccessorImpl.
↪invoke(NativeMethodAccessorImpl.java:62) ~[na:1.8.0_20]
    at sun.reflect.DelegatingMethodAccessorImpl.
↪invoke(DelegatingMethodAccessorImpl.java:43) ~[na:1.8.0_20]
    at java.lang.reflect.Method.invoke(Method.java:483) ~[na:1.8.0_20]
```

If there are no specific issues related to system operation, this measure is not required since it does not affect the application behavior even after the log is output.

However, in case of any system operation issues like log monitoring etc, log output can be controlled by specifying the method (`destroy-method` attribute) that is called while terminating Spring Application-

Context.

In the example below, it is specified such that `getExecutorType` method is called. `getExecutorType` is the method which is only used for returning the execution mode specified in constructor argument and has no impact on other operations.

```
<bean id="batchSqlSessionTemplate"
      class="org.mybatis.spring.SqlSessionTemplate"
      destroy-method="getExecutorType">
    <constructor-arg index="0" ref="sqlSessionFactory"/>
    <constructor-arg index="1" value="BATCH"/>
</bean>
```

Settings for creating Batch mode Repository in batch

When a batch mode Repository is to be created in a batch, a Bean can be defined for the Repository by using the scan function (`mybatis:scan` element) provided by MyBatis-Spring.

In the configuration example below, Bean is registered for the Repositories of `REUSE` mode and `BATCH` mode, with respect to all the Repositories.

- Create `BeanNameGenerator`.

```
package com.example.domain.repository;

import org.springframework.beans.factory.config.BeanDefinition;
import org.springframework.beans.factory.support.BeanDefinitionRegistry;
import org.springframework.beans.factory.support.BeanNameGenerator;
import org.springframework.util.ClassUtils;

import java.beans.Introspector;

// (1)
public class BachRepositoryBeanNameGenerator implements BeanNameGenerator {
    // (2)
    @Override
    public String generateBeanName(BeanDefinition definition,␣
↪BeanDefinitionRegistry registry) {
        String defaultBeanName = Introspector.decapitalize(ClassUtils.
↪getShortName(definition
                .getBeanClassName()));
        return defaultBeanName.replaceAll("Repository", "BatchRepository");
```

```
    }
}
```

Sr. No.	Description
1.	Create a class that generates the Bean name to be registered in Spring ApplicationContext. This class is necessary to avoid duplication in the bean name of `REUSE` mode Repository for normal use and the Bean name of `BATCH` mode.
2.	Implement the method for generating a Bean name. In the above example, duplication of Bean name for `REUSE` mode Repository for normal use can be prevented by setting `BatchRepository` as the Bean name suffix.

- Bean definition is added to `projectName-domain/src/main/resources/META-INF/spring/projectName`

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:mybatis="http://mybatis.org/schema/mybatis-spring"
       xsi:schemaLocation="
       http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans.xsd
       http://www.springframework.org/schema/context
       http://www.springframework.org/schema/context/spring-context.xsd
       http://mybatis.org/schema/mybatis-spring
       http://mybatis.org/schema/mybatis-spring.xsd">

    <bean id="sqlSessionFactory"
          class="org.mybatis.spring.SqlSessionFactoryBean">
        <property name="dataSource" ref="dataSource"/>
        <property name="configLocation"
                  value="classpath:META-INF/mybatis/mybatis-config.xml"/>
    </bean>

    <!-- ... -->

    <bean id="batchSqlSessionTemplate"
          class="org.mybatis.spring.SqlSessionTemplate">
        <constructor-arg index="0" ref="sqlSessionFactory"/>
        <constructor-arg index="1" value="BATCH"/>
    </bean>

    <!-- (3) -->
    <mybatis:scan base-package="com.example.domain.repository"
```

```
        template-ref="batchSqlSessionTemplate"
        name-generator="com.example.domain.repository.
↪BatchRepositoryBeanNameGenerator"/>

</beans>
```

Sr. No.	Attribute	Description
3.	-	Register the Bean for batch mode Repository by using `mybatis:scan` element.
	base-package	Specify the base package that scans the Repository. Repository interface that exists under specified package is scanned and a Bean is registered in Spring ApplicationContext.
	template-ref	Specify Bean of `SqlSessionTemplate` for batch mode.
	name-generator	Specify a class for generating the Bean name of scanned Repository. Basically, specify the class name (FQCN) for the class created in (1). If class name is not specified, Bean names are duplicated. Hence, batch mode Repository is not registered in Spring ApplicationContext.

How to use the Repository of batch mode

An implementation example on how to access database by using batch mode Repository is given below.

```
@Transactional
@Service
public class TodoServiceImpl implements TodoService {

    // (1)
    @Inject
    @Named("todoBatchRepository")
    TodoRepository todoBatchRepository;

    @Override
    public void updateTodos(List<Todo> todos) {
        for (Todo todo : todos) {
            // (2)
            todoBatchRepository.update(todo);
        }
    }

}
```

Sr. No.	Description
1.	Inject the Repository of batch mode.
2.	Call the method for batch mode Repository and update Entity. In case of batch mode Repository, since the SQL is not executed in the timing when method is called. As a result, the update results returned from method need to be ignored. The SQL for updating an Entity is executed in a batch immediately before committing a transaction and the transaction is committed if there is no error.

Note: Batch execution timing

Timing of SQL batch execution is as below.

- Immediately before the transaction is committed
- Immediately before executing the query (SELECT)

Notes of sequence of calling repository method refer to "*Sequence of calling Repository method*".

Precautions when using batch mode Repository

It is important to note the following points in implementation of Service class when using batch mode Repository.

- *Determination of update results*
- *How to detect the unique constraint violation*
- *Sequence of calling Repository method*

Determination of update results

When batch mode Repository is used, the validity of update results cannot be checked.

Update results returned from Mapper interface method when a batch mode is used, are as follows.

- Fixed value(`org.apache.ibatis.executor.BatchExecutor#BATCH_UPDATE_RETURN_VALUE`) when return value is numeric (`int` or `long`)
- `false` when return value is `boolean`

This is due to the mechanism wherein SQL is not executed within the timing of calling Mapper interface method but is queued for batch execution (`java.sql.Statement#addBatch()`).

It signifies that the implementation below cannot be performed.

```
@Transactional
@Service
public class TodoServiceImpl implements TodoService {

    @Inject
    @Named("todoBatchRepository")
    TodoRepository todoBatchRepository;

    @Override
    public void updateTodos(List<Todo> todos) {
        for (Todo todo : todos) {
            boolean updateSuccess = todoBatchRepository.update(todo);
            // (1)
            if (!updateSuccess) {
                // ...
            }
        }
    }

}
```

Sr. No.	Description
1.	When the implementation is performed as described below, update results are always returned as `false` resulting in the execution of process at the time of update failure.

Based on the application requirement, it is also necessary to check the validity of update results executed in the batch. In such cases, "a method to execute SQL queued for batch execution" is provided in the Mapper interface."

In MyBatis 3.2, `flushStatements` method of `org.apache.ibatis.session.SqlSession` interface must be called directly, however, a method is supported in MyBatis 3.3.0 and subsequent versions wherein a method which assigns `@org.apache.ibatis.annotations.Flush` annotation in Mapper interface is created.

Warning: Regarding update results returned by JDBC driver while using batch mode

Although it has been described earlier that update results at the time of batch execution can be received when a method which assigns `@Flush` annotation (and `flushStatements` method of `SqlSession` interface) is used, it cannot be guaranteed that the update results returned from JDBC driver can be used as "number of processed records".

Since it depends on implementation of JDBC driver to be used, the specifications of JDBC driver to be used must be checked in advance.

How to create and call a method which assigns `@Flush` annotation is given below. |

```
public interface TodoRepository {
    // ...
    @Flush // (1)
    List<BatchResult> flush();
}
```

```
@Transactional
@Service
public class TodoServiceImpl implements TodoService {

    @Inject
    @Named("todoBatchRepository")
    TodoRepository todoBatchRepository;

    @Override
    public void updateTodos(List<Todo> todos) {

        for (Todo todo : todos) {
            todoBatchRepository.update(todo);
        }

        List<BatchResult> updateResults = todoBatchRepository.flush(); // (2)

        // Validate update results
        // ...

    }

}
```

Sr. No.	Description
1.	Create a method which assigns `@Flush` annotation (hereafter referred to as "`@Flush` method"). When it is necessary to determine update results, specify list type of `org.apache.ibatis.executor.BatchResult` as a return value. When it is not necessary to determine update results (when only a database error like a unique constraint violation is to be handled), the return value can be `void`.
2.	Call `@Flush` method within the timing in which SQL queued for batch execution is to be executed. If `@Flush` method is called, `flushStatements` method of `SqlSession` object associated with Mapper interface is called and SQL queued for batch execution is executed. When it is necessary to determine update results, validity of update results returned from `@Flush` method is checked.

How to detect the unique constraint violation

When batch mode Repository is used, it is not possible to detect database errors like unique constraint violation etc. as a Service process.

This is due to the mechanism wherein SQL is not executed within the timing of calling a Mapper interface method and is queued for the batch execution (`java.sql.Statement#addBatch()`). It signifies that the implementation given below cannot be performed.

```
@Transactional
@Service
public class TodoServiceImpl implements TodoService {

    @Inject
    @Named("todoBatchRepository")
    TodoRepository todoBatchRepository;

    @Override
    public void storeTodos(List<Todo> todos) {
        for (Todo todo : todos) {
            try {
                todoBatchRepository.create(todo);
            // (1)
            } catch (DuplicateKeyException e) {
                // ....
            }
        }
    }

}
```

Sr. No.	Description
1.	When the implementation is performed as described above, `org.springframework.dao.DuplicateKeyException` exception does not occur within that timing. Hence, the process after `DuplicateKeyException` supplement is not executed. This is because SQL batch execution is performed after termination of Service process (just before the transaction is committed).

Depending on application requirement, it is necessary to detect a unique constraint violation at the time of batch execution. In such cases, "a method to execute SQL queued for batch execution (`@Flush` method)" must be provided in Mapper interface. Refer "*Determination of update results*" described earlier for details of `@Flush` method.

Sequence of calling Repository method

Batch mode is used to improve the performance of update process. However, if the calling sequence of Repository method is incorrect, no improvement is observed in the performance.

It is important to understand the MyBatis specifications given below, in order to improve the performance using batch mode.

- When query (SELECT) is executed, SQL waiting in the queue till then is executed in batch.
- `PreparedStatement` is generated for each update process (Repository method) called in succession and SQL is queued.

It signifies that if the implementation is performed as below, advantages of using batch mode cannot be obtained.

- Example 1

```
@Transactional
@Service
public class TodoServiceImpl implements TodoService {

    @Inject
    @Named("todoBatchRepository")
    TodoRepository todoBatchRepository;

    @Override
    public void storeTodos(List<Todo> todos) {
        for (Todo todo : todos) {
            // (1)
            Todo currentTodo = todoBatchRepository.findOne(todo.getTodoId());
            if (currentTodo == null) {
                todoBatchRepository.create(todo);
            } else{
                todoBatchRepository.update(todo);
            }
        }
    }

}
```

Sr. No.	Description
1.	When the implementation is performed as described in the above example, since a query is executed at the start of iteration process, SQL is executed in a batch for each record. This is almost same as the execution in simple mode (`SIMPLE`). When the process described above is necessary, it is more effective to use Repository of PreparedStatement reuse mode (`REUSE`).

- Example 2

```
@Transactional
@Service
public class TodoServiceImpl implements TodoService {

    @Inject
    @Named("todoBatchRepository")
    TodoRepository todoBatchRepository;

    @Override
    public void storeTodos(List<Todo> todos) {
        for (Todo todo : todos) {
            // (2)
            todoBatchRepository.create(todo);
            todoBatchRepository.createHistory(todo);
        }
    }

}
```

Sr. No.	Description
2.	When the process described above is necessary, Repository method is called alternately. Hence, `PreparedStatement` is generated for each record. This is almost same as executing a process in simple mode (`SIMPLE`). When the process described above is necessary, it is more effective to use PreparedStatement reuse mode Repository (`REUSE`).

Implementation of a stored procedure

How to call a stored procedure or function registered in database from MyBatis3 is explained.

A function registered in PostgreSQL is called in the implementation example explained below.

- Register a stored procedure (function).

```
/* (1) */
CREATE FUNCTION findTodo(pTodoId CHAR)
RETURNS TABLE(
    todo_id CHAR,
    todo_title VARCHAR,
    finished BOOLEAN,
    created_at TIMESTAMP,
    version BIGINT
) AS $$ BEGIN RETURN QUERY
SELECT
    t.todo_id,
```

```
    t.todo_title,
    t.finished,
    t.created_at,
    t.version
FROM
    t_todo t
WHERE
    t.todo_id = pTodoId;
END;
$$ LANGUAGE plpgsql;
```

Sr. No.	Description
1.	It is the function used to fetch records for specified ID.

- Defining the method in Repository interface.

```
// (2)
public interface TodoRepository extends Repository {
    Todo findOne(String todoId);
}
```

Sr. No.	Description
2.	The interface should be same as the interface used while executing the SQL.

- Call a stored procedure in the mapping file.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
        "http://mybatis.org/dtd/mybatis-3-mapper.dtd" >
<mapper namespace="com.example.domain.repository.todo.TodoRepository">

    <!-- (3) -->
    <select id="findOne" parameterType="string" resultType="Todo"
            statementType="CALLABLE">
        <!-- (4) -->
        {call findTodo(#{todoId})}
    </select>
```

```
</mapper>
```

Sr. No.	Description
3.	Implement the statement to call the stored procedure. Specify `CALLABLE` in `statementType` attribute when calling the stored procedure. When `CALLABLE` is specified, the stored procedure is called by using `java.sql.CallableStatement`. Specify `resultType` attribute or `resultMap` attribute in order to map the OUT parameter in JavaBean.
4.	Call the stored procedure. When the stored procedure (faction) is to be called, specify in `{call Procedure or Function name (IN parameter...)}` format. In the above example, the procedure is called by specifying an ID in the IN parameter for a function called `findTodo`.

6.2.4 Appendix

Mapper interface mechanism

When using a Mapper interface, the developer can execute SQL only by creating a Mapper interface and mapping file.

when executing an application, implementation class of Mapper interface is generated by MyBatis3 using the Proxy function of JDK. Hence, the developer need not create an implementation class of Mapper interface.

It is not necessary to define inheritance and annotation of interface provided by MyBatis3, for the Mapper interface and it can be simply created as a Java interface.

Example of how to create a Mapper interface and mapping file, and example of how to use it in the application (Service) are given below.

Here, the focus should be on the main points related to explanation of code since the aim is to form an image of the deliverables to be created by developers.

- How to create a Mapper interface

  In this guideline, since it is presumed that that the Mapper interface of MyBatis3 is used as Repository interface, interface name is in the format, “Entity name” + `"Repository"` .

```
package com.example.domain.repository.todo;

import com.example.domain.model.Todo;

public interface TodoRepository {
    Todo findOne(String todoId);
}
```

- How to create a mapping file

  In the mapping file, FQCN (fully qualified class name) of Mapper interface is specified as namespace. Its association with SQL that is executed while calling the method defined in the Mapper interface can be formed by specifying a method name in id attribute of various statement tags (insert/update/delete/select tags).

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org/DTD Mapper 3.0//EN"
    "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.example.domain.repository.todo.TodoRepository">

    <resultMap id="todoResultMap" type="Todo">
        <result column="todo_id" property="todoId" />
        <result column="title" property="title" />
        <result column="finished" property="finished" />
    </resultMap>

    <select id="findOne" parameterType="String" resultMap="todoResultMap">
      SELECT
        todo_id,
        title,
        finished
      FROM
        t_todo
      WHERE
        todo_id = #{todoId}
    </select>

</mapper>
```

- How to use Mapper interface in an application (Service)

  When a method of Mapper interface is to be called from an application (Service), a method of Mapper object injected by Spring (DI container) is called. The application (Service) transparently executes SQL by calling Mapper object method and can obtain SQL execution results.

```
package com.example.domain.service.todo;

import com.example.domain.model.Todo;
import com.example.domain.repository.todo.TodoRepository;
```

```
public class TodoServiceImpl implements TodoService {

    @Inject
    TodoRepository todoRepository;

    public Todo getTodo(String todoId){
        Todo todo = todoRepository.findOne(todoId);
        if(todo == null){
            throw new ResourceNotFoundException(
                ResultMessages.error().add("e.ex.td.5001" ,todoId));
        }
        return todo;
    }

}
```

The process flow up to SQL execution when Mapper interface method is called, is shown below.

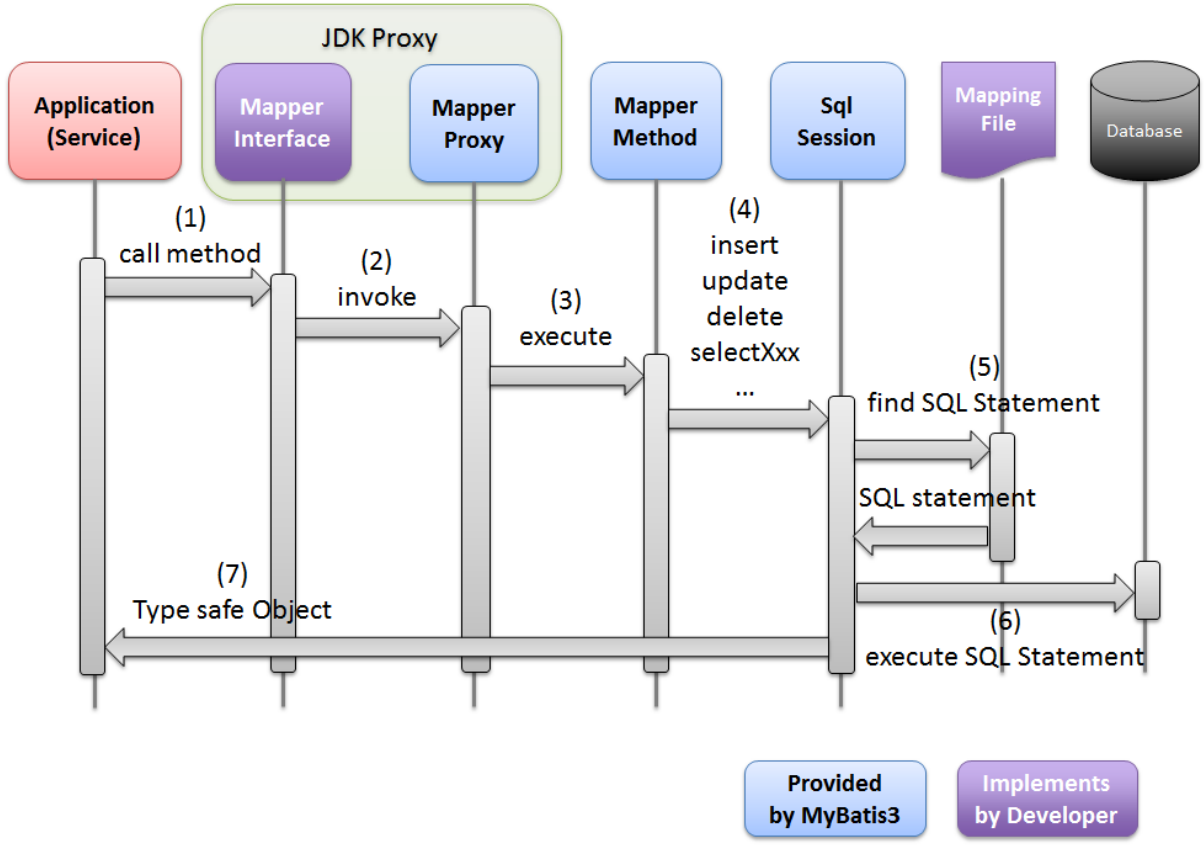


Figure.6.5 **Picture - Mapper mechanism**

Sr. No.	Description
1.	Application calls the method defined in Mapper interface. The implementation class of Mapper interface (Proxy object of Mapper interface) is generated by MyBatis3 components at application startup time.
2.	Proxy object of Mapper interface calls invoke method of `MapperProxy` . Role of `MapperProxy` is to handle the method calling of Mapper interface.
3.	`MapperProxy` generates `MapperMethod` corresponding to the called Mapper interface method and calls execute method. `MapperMethod` plays the role of calling `SqlSession` method corresponding to the called Mapper interface method.
4.	`MapperMethod` calls `SqlSession` method. When a `SqlSession` method is called, a key (hereafter referred to as “Statement ID”) is passed in order to specify the SQL statement to be executed.
5.	`SqlSession` fetches a SQL statement from a mapping file using the specified statement ID as a key.

次のページに続く

Table. 6.9 – 前のページからの続き

Sr. No.	Description
6.	`SqlSession` sets the value in bind variable specified in the SQL statement fetched by mapping file and executes SQL.
7.	Mapper interface (`SqlSession`) converts SQL execution results to JavaBean etc. and returns it to the application. When number of records or number of updated records are to be fetched, primitive type or primitive wrapper type etc. form the return values.

Tip: Statement ID

Statement ID is a key to specify the SQL statement to be executed. It is generated in accordance with the rule **"FQCN of Mapper interface + "." + name of the called Mapper interface method"**.

When SQL statement corresponding to the statement ID generated by `MapperMethod` is to be defined in the mapping file, it is necessary to specify "FQCN of Mapper interface" in the namespace of mapping file and "method name of Mapper interface" in id attribute of various statement tags.

TypeAlias settings

TypeAlias should basically be configured per package by using `package` element. However, following methods can also be used.

- A method to configure an alias name per class
- A method to overwrite the alias name which is assigned as default (a method that specifies an optional alias name)

Configuring TypeAlias per class

TypeAlias can also be configured per class.

- `projectName-domain/src/main/resources/META-INF/mybatis/mybatis-config.xml`

```
<typeAliases>
    <!-- (1) -->
    <typeAlias
        type="com.example.domain.repository.account.AccountSearchCriteria" />
    <package name="com.example.domain.model" />
</typeAliases>
```

Sr. No.	Description
1.	Specify the fully qualified class name (FQCN) of the class for which alias is to be set, in `type` attribute of `typeAlias` element. In the above example, alias name of `com.example.domain.repository.account.AccountSear` class becomes `AccountSearchCriteria` (part after removing package part). When an optional value is to be specified in the alias name, optional alias name can be specified in `alias` attribute of `typeAlias` element.

Overwriting alias name assigned as default

When alias is set by using `package` element or when alias is set by omitting `alias` attribute of `typeAlias` element, alias for TypeAlias is a part obtained after removing the package part from fully qualified class name (FQCN).

When optional alias is to be used instead of default alias, it can be specified by specifying `@org.apache.ibatis.type.Alias` annotation in the class wherein TypeAlias to is to be configured.

- Java class for configuring alias

```
package com.example.domain.model.book;

@Alias("BookAuthor") // (1)
public class Author {
    // ...
}
```

```
package com.example.domain.model.article;

@Alias("ArticleAuthor") // (1)
public class Author {
    // ...
}
```

Sr. No.	Description
1.	Specify the alias name in `value` attribute of `@Alias` annotation. In the above example, the alias name for `com.example.domain.model.book.Author` class is `BookAuthor` . When a class with same name is stored in a different package, different alias names can be configured for each class by using this method. However, in this guideline, it is recommended to design the class name such that duplication is avoided. In the above example, `BookAuthor` and `ArticleAuthor` should be considered as class names.

Tip: An alias name for TypeAlias is applied in the following priority order.

- Value specified for `alias` attribute of `typeAlias` element
- Value specified for `value` attribute of `@Alias` annotation
- Alias name assigned as default (part after removing the package name from fully qualified class name)

SQL switching by the database

MyBatis3 provides a mechanism (`org.apache.ibatis.mapping.VendorDatabaseIdProvider`) wherein vendor information of database that is connected from JDBC driver, is fetched and SQL to be used is switched.

This mechanism is effective when building an application that can support multiple databases as operating environment.

Note: In this guideline, it is recommended to manage the components and configuration file that are dependent on the environment by a sub project called [projectName]-env and create components and configuration file that are in execution environment at the time of building.

[projectName]-env is a sub-project to absorb differences in each of the following

- Development environment (local PC environment)
- Various test environments
- Commercial environment

It can also be used in the development of an application that supports multiple databases.

Basically, it is recommended to manage environment-dependent components and configuration file by using a sub-project called [projectName]-env. However, the mechanism below can also be used if only a minor SQL difference is to be absorbed.

Architects should try to achieve a uniform implementation for overall application by clearly identifying a guideline on how to implement SQL environment dependency based on the differences in the database.

- Bean is defined in `projectName-domain/src/main/resources/META-INF/spring/projectName-infra`

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:mybatis="http://mybatis.org/schema/mybatis-spring"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://mybatis.org/schema/mybatis-spring
        http://mybatis.org/schema/mybatis-spring.xsd
    ">
```

```
    <import resource="classpath:/META-INF/spring/projectName-env.xml" />

    <!-- (1) -->
    <bean id="databaseIdProvider"
          class="org.apache.ibatis.mapping.VendorDatabaseIdProvider">
        <!-- (2) -->
        <property name="properties">
            <props>
                <prop key="H2">h2</prop>
                <prop key="PostgreSQL">postgresql</prop>
            </props>
        </property>
    </bean>

    <bean id="sqlSessionFactory"
        class="org.mybatis.spring.SqlSessionFactoryBean">
        <property name="dataSource" ref="dataSource" />
        <!-- (3) -->
        <property name="databaseIdProvider" ref="databaseIdProvider"/>
        <property name="configLocation"
            value="classpath:/META-INF/mybatis/mybatis-config.xml" />
    </bean>

    <mybatis:scan base-package="com.example.domain.repository" />

</beans>
```

Sr. No.	Description
1.	Define a Bean for `VendorDatabaseIdProvider` provided by MyBatis3. `VendorDatabaseIdProvider` is a class for handling a product name (`java.sql.DatabaseMetaData#getDatabaseProductName()`) of a database that is fetched from JDBC driver, as a database ID.
2.	Specify the mapping of database product name fetched from JDBC driver and database ID in `properties` property. Refer to "MyBatis3 REFERENCE DOCUMENTATION(Configuration-databaseIdProvider-)" for specifications of mapping.
3.	Specify `DatabaseIdProvider` defined in (1) for the `databaseIdProvider` property of `SqlSessionFactoryBean` that uses database ID By specifying this, it is possible to refer the database ID from mapping file.

Note: In this guideline, it is recommended to use a method to map the product name of database and database ID, by specifying `properties` property

This is due to possible change in the product name of database fetched from JDBC driver, based on the JDBC version.

When `properties` property is used, the difference between product names of each version to be used can be managed at a single location.

- Implementing mapping file.

```
<insert id="create" parameterType="Todo">
    <!-- (1) -->
    <selectKey keyProperty="todoId" resultType="string" order="BEFORE"
               databaseId="h2">
        SELECT RANDOM_UUID()
    </selectKey>
    <selectKey keyProperty="todoId" resultType="string" order="BEFORE"
               databaseId="postgresql">
        SELECT UUID_GENERATE_V4()
    </selectKey>

    INSERT INTO
      t_todo
    (
        todo_id
        ,todo_title
        ,finished
        ,created_at
        ,version
    )
    VALUES
    (
        #{todoId}
        ,#{todoTitle}
        ,#{finished}
        ,#{createdAt}
        ,#{version}
    )
</insert>
```

Sr. No.	Description
1.	When statement elements (`select` element, `update` element, `sql` element etc.) are to be changed for each database, specify the database ID in `databaseId` attribute of each element. When `databaseId` attribute is specified, the statement element that matches with database ID is used. In the above example, ID is generated, by calling the UUID generation function specific for each database.

Tip: In the above example, `UUID_GENERATE_V4()` is called as the UUID generation function for PostgreSQL. However, this function belongs to a sub-module called uuid-ossp.

When this function is to be used, uuid-ossp module should be enabled.

Tip: Database ID can also be referred in OGNL base expression (Expression language).

It signifies that database ID can be used as a condition for dynamic SQL. How to implement is given below.

```
<select id="findAllByCreatedAtBefore" parameterType="_int" resultType="Todo
↪">
    SELECT
        todo_id,
        todo_title,
        finished,
        created_at,
        version
    FROM
        t_todo
    WHERE
        <choose>
            <!-- (2) -->
            <when test="_databaseId == 'h2'">
                <bind name="criteriaDate"
                      value="'DATEADD(\'DAY\',#{days} * -1,#{currentDate})'
↪"/>
            </when>
            <when test="_databaseId == 'postgresql'">
                <bind name="criteriaDate"
                      value="'#{currentDate}::DATE - (#{days} * INTERVAL \
↪'1 DAY\')'"/>
            </when>
        </choose>
        <![CDATA[
            created_at < ${criteriaDate}
        ]]>
</select>
```

Sr. No.	Description
2.	Database ID is stored in a specific variable called `_databaseId`, in a OGNL base expression (Expression language). In the above example, the condition for extracting records that are created prior to "System date - specified date" is specified by using database function.

How to fetch a related Entity by a single SQL

How to fetch a main Entity and a related Entity by a single SQL is explained below.

When a mechanism that fetches a main Entity and a related Entity together is to be used, it is not necessary to build an Entity (JavaBean) in the Service class and Service class can solely focus on implementation of business logic (business rules).

Further, this method can also be used to avoid the N+1.

Refer to "*How to resolve N+1*" for N+1.

Warning: It is important to note the following points when main Entity and related Entity are to be fetched together.

- In the explanation below, all the related Entities are fetched together by a single SQL. However, when it is to be used in the actual project, only the related Entities that are required in the process should be fetched. If unused related Entities are fetched together, it results in generation of unnecessary objects and as mapping process is performed, it causes efficiency deterioration. **Particularly, in the SQL that performs list search, in many cases, only the required related Entities are fetched.**
- For the related Entities which are used less frequently, the entities can be fetched independently without fetching them together.

  If the related Entities that are used less frequently are fetched simultaneously, unnecessary objects are generated and as mapping process is performed, it can cause efficiency deterioration.
- When multiple related Entities with 1:N relation are to be included, it is advisable to adopt the method wherein the main Entity and related Entity are fetched separately. When there are multiple related Entities with 1:N relation, it becomes necessary to fetch unnecessary data from DB causing deterioration inefficiency. Refer to "*How to resolve N+1*" for how to fetch main Entity and related Entity separately.

Tip: Following methods are provided when it is necessary to separately fetch the related Entities that are used less frequently.

- By calling a method (SQL) that fetches a related Entity in the process of Service class.

- By marking the related Entity as "Lazy Load" and transparently executing the SQL when Getter method is called.

When "Lazy Load" mechanism is used, it is not necessary to build an Entity (JavaBean) in Service class and Service class can solely focus on implementation of business logic (business rules).

When "Lazy Load" is used in SQL that performs list search, N+1 can occur. Hence adequate care must be taken.

Refer to "*Settings to apply Lazy Load for related Entity*" for how to use "Lazy Load".

An implementation example is explained below wherein order data handled by a shopping site is fetched together in a single SQL and mapped in main Entity and related Entity.

The implementation method explained here is just an example. MyBatis3 provides a lot of functions that are not explained in this section and can perform a much higher level of mapping.

Refer to "MyBatis3 REFERENCE DOCUMENTATION(Mapper XML Files-Result Maps-) " for details of mapping function of MyBatis3.

Table layout and data

Table used in the explanation is as given below.

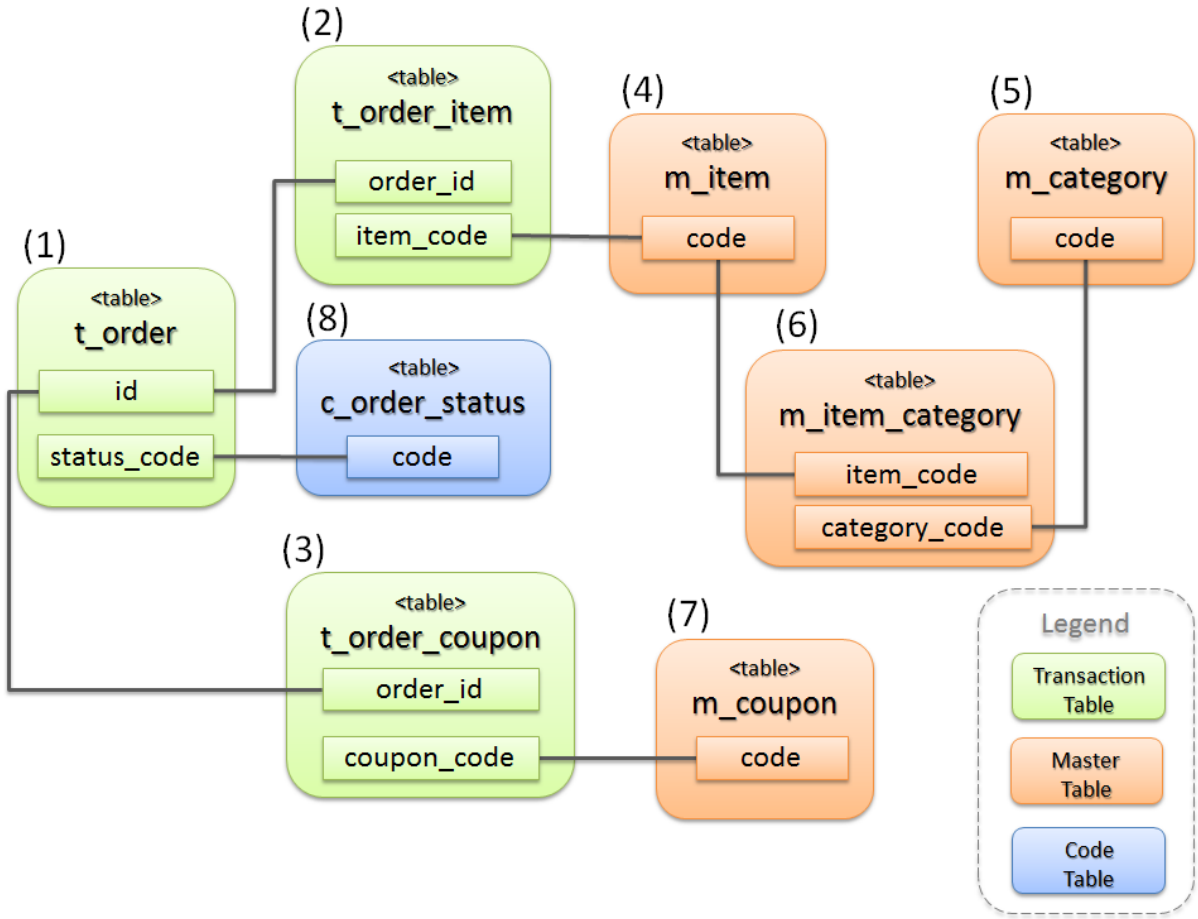


Figure.6.6 **Picture - ER diagram**

Sr. No.	Category	Table name	Description
1.	Transaction system	t_order	A table that stores order data. 1 record is stored for one order.
2.		t_order_item	A table that stores the product data purchased by one order. When multiple products are purchased in a single order, the number of product records is stored.
3.		t_order_coupon	A table that stores data of coupons that are used in a single order. When multiple coupons are used in a single order, the number of coupon records is stored. When coupons are not used, records are not stored.
4.	Master system	m_item	A master table that defines the product.
5.		m_category	A master table that defines product category.
6.		m_item_category	A master table that defines the category to which a product belongs. It stores the mapping of product and category. It is a model wherein one product can belong to multiple categories.
7.		m_coupon	A master table that defines a coupon.

SQL (DDL and DML) for creating the storage data and the table layout used in the explanation are as follows. (SQL is for H2 Database)

- DDL for creating a master table

```
CREATE TABLE m_item (
    code CHAR(10),
    name NVARCHAR(256),
    price INTEGER,
    CONSTRAINT m_item_pk PRIMARY KEY(code)
);

CREATE TABLE m_category (
    code CHAR(10),
    name NVARCHAR(256),
    CONSTRAINT m_category_pk PRIMARY KEY(code)
);

CREATE TABLE m_item_category (
    item_code CHAR(10),
    category_code CHAR(10),
    CONSTRAINT m_item_category_pk PRIMARY KEY(item_code, category_code),
    CONSTRAINT m_item_category_fk1 FOREIGN KEY(item_code) REFERENCES m_
↪item(code),
    CONSTRAINT m_item_category_fk2 FOREIGN KEY(category_code) REFERENCES m_
↪category(code)
);

CREATE TABLE m_coupon (
    code CHAR(10),
    name NVARCHAR(256),
    price INTEGER,
    CONSTRAINT m_coupon_pk PRIMARY KEY(code)
);
```

- DDL for creating a code table

```
CREATE TABLE c_order_status (
    code VARCHAR(10),
    name NVARCHAR(256),
    CONSTRAINT c_order_status_pk PRIMARY KEY(code)
);
```

- DDL for creating a transaction table

```
CREATE TABLE t_order (
    id INTEGER,
    status_code VARCHAR(10),
    CONSTRAINT t_order_pk PRIMARY KEY(id),
    CONSTRAINT t_order_fk FOREIGN KEY(status_code) REFERENCES c_order_
↪status(code)
);

CREATE TABLE t_order_item (
    order_id INTEGER,
    item_code CHAR(10),
    quantity INTEGER,
    CONSTRAINT t_order_item_pk PRIMARY KEY(order_id, item_code),
    CONSTRAINT t_order_item_fk1 FOREIGN KEY(order_id) REFERENCES t_order(id),
    CONSTRAINT t_order_item_fk2 FOREIGN KEY(item_code) REFERENCES m_item(code)
);

CREATE TABLE t_order_coupon (
    order_id INTEGER,
    coupon_code CHAR(10),
    CONSTRAINT t_order_coupon_pk PRIMARY KEY(order_id, coupon_code),
    CONSTRAINT t_order_coupon_fk1 FOREIGN KEY(order_id) REFERENCES t_
↪order(id),
    CONSTRAINT t_order_coupon_fk2 FOREIGN KEY(coupon_code) REFERENCES m_
↪coupon(code)
);
```

- DML for data input

```
-- Setup master tables
INSERT INTO m_item VALUES ('ITM0000001','Orange juice',100);
INSERT INTO m_item VALUES ('ITM0000002','NotePC',100000);

INSERT INTO m_category VALUES ('CTG0000001','Drink');
INSERT INTO m_category VALUES ('CTG0000002','PC');
INSERT INTO m_category VALUES ('CTG0000003','Hot selling');

INSERT INTO m_item_category VALUES ('ITM0000001','CTG0000001');
INSERT INTO m_item_category VALUES ('ITM0000002','CTG0000002');
INSERT INTO m_item_category VALUES ('ITM0000002','CTG0000003');

INSERT INTO m_coupon VALUES ('CPN0000001','Join coupon',3000);
INSERT INTO m_coupon VALUES ('CPN0000002','PC coupon',30000);

-- Setup code tables
INSERT  INTO  c_order_status VALUES ('accepted','Order accepted');
INSERT  INTO  c_order_status VALUES ('checking','Stock checking');
INSERT  INTO  c_order_status VALUES ('shipped','Item Shipped');

-- Setup transaction tables
```

```
INSERT INTO t_order VALUES (1,'accepted');
INSERT INTO t_order VALUES (2,'checking');

INSERT INTO t_order_item VALUES (1,'ITM0000001',1);
INSERT INTO t_order_item VALUES (1,'ITM0000002',2);
INSERT INTO t_order_item VALUES (2,'ITM0000001',3);
INSERT INTO t_order_item VALUES (2,'ITM0000002',4);

INSERT INTO t_order_coupon VALUES (1,'CPN0000001');
INSERT INTO t_order_coupon VALUES (1,'CPN0000002');

COMMIT;
```

Entity class diagram

In the implementation example, the records stored in the table above are mapped in the following Entity (JavaBean).

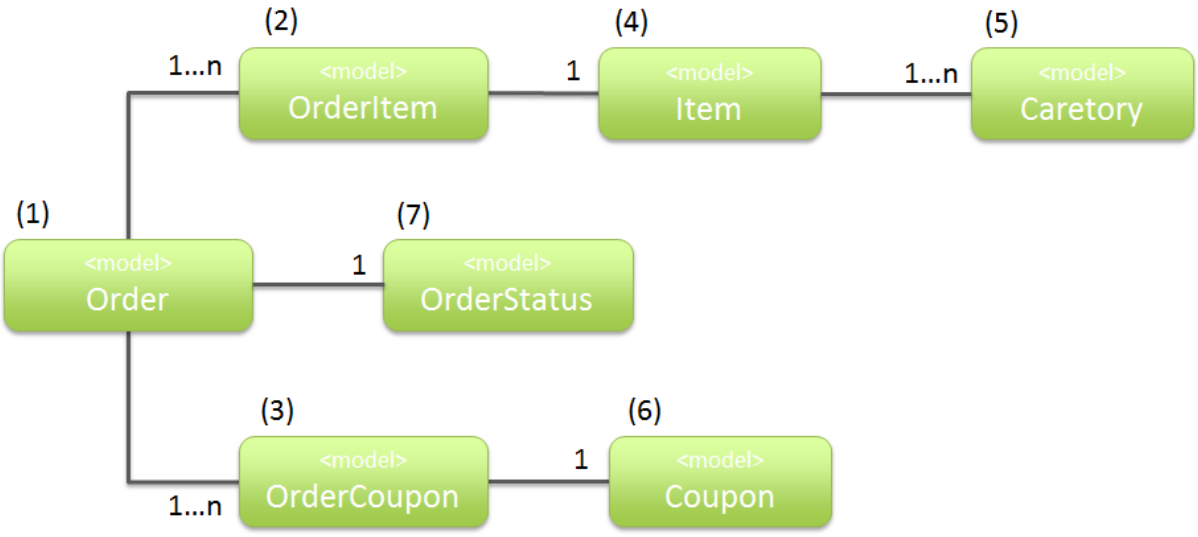


Figure.6.7 **Picture - Class(JavaBean) diagram**

Sr. No.	Class name	Description

1.	Order	JavaBean that represents 1 record of t_order table.
		A single record of `OrderStatus` and multiple records of `OrderItem` and `OrderCoupon` are stored as related Entities.
		`public class Order implements Serializable {`
		`  private static final long serialVersionUID = 1L;`
		`  private int id;`
		`  private OrderStatus orderStatus;`
		`  List<OrderItem> orderItems;`
		`  List<OrderCoupon> orderCoupons;`
		`  // ...`
		`}`

2.	OrderItem	JavaBean that represents 1 record of t_order_item table.
		`Item` is stored as related Entity.
		`public class OrderItem implements Serializable {`
		`  private static final long serialVersionUID = 1L;`
		`  private int orderId;`
		`  private Item item;`
		`  private int quantity;`
		`  // ...`
		`}`

3.	OrderCoupon	A JavaBean that represents 1 record of t_order_coupon table.
		`Coupon` is stored as related Entity.
		`public class OrderCoupon implements Serializable {`
		`  private static final long serialVersionUID = 1L;`
		`  private int orderId;`
		`  private Coupon coupon;`
		`  // ...`
		`}`

次のページに続く

Table. 6.10 – 前のページからの続き

Sr. No.	Class name	Description

4.	Item	JavaBean that represents 1 record of m_item table.
		Multiple, associated `Category` records are stored as related objects.
		Mapping with `Category` is performed by m_item_category table.
		`public class Item implements Serializable {`
		`private static final long serialVersionUID = 1L;`
		`private String code;`
		`private String name;`
		`private int price;`
		`private List<Category> categories;`
		`// ...`
		`}`

5.	Category	JavaBean that represents 1 record of m_category table.
		`public class Category implements Serializable {`
		`private static final long serialVersionUID = 1L;`
		`private String code;`
		`private String name;`
		`// ...`
		`}`

6.	Coupon	JavaBean that represents 1 record of m_coupon table.
		`public class Coupon implements Serializable {`
		`private static final long serialVersionUID = 1L;`
		`private String code;`
		`private String name;`
		`private int price;`
		`// ...`
		`}`

7.	OrderStatus	A JavaBean that represents 1 record of c_order_status table.
		`public class OrderStatus implements Serializable {`
		`private static final long serialVersionUID = 1L;`
		`private String code;`
		`private String name;`
		`// ...`
		`}`

Implementation of Repository interface

Following methods are implemented in the example.

- A method that fetches 1 record of Order object (`findOne`)
- A method that fetches an Order object of corresponding page (`findPage`)

```
package com.example.domain.repository.order;

import com.example.domain.model.Order;

import java.util.List;

public interface OrderRepository {

    Order findOne(int id);

    List<Order> findPage(@Param("pageable") Pageable pageable);

}
```

SQL implementation

When related Entities are to be fetched together by a single SQL, JOIN is performed on the tables to be fetched and all the records required for mapping are fetched.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
        "http://mybatis.org/dtd/mybatis-3-mapper.dtd" >
<mapper namespace="com.example.domain.repository.order.OrderRepository">

    <!-- (1) -->
    <sql id="selectFromJoin">
        SELECT
            /* (2) */
            o.id,
            /* (3) */
            o.status_code,
            os.name AS status_name,
```

```
            /* (4) */
            oi.quantity,
            i.code AS item_code,
            i.name AS item_name,
            i.price AS item_price,
            /* (5) */
            ct.code AS category_code,
            ct.name AS category_name,
            /* (6) */
            cp.code AS coupon_code,
            cp.name AS coupon_name,
            cp.price AS coupon_price
        FROM
            ${orderTable} o
        /* (7) */
        INNER JOIN c_order_status os ON os.code = o.status_code
        INNER JOIN t_order_item oi ON oi.order_id = o.id
        INNER JOIN m_item i ON i.code = oi.item_code
        INNER JOIN m_item_category ic ON ic.item_code = i.code
        INNER JOIN m_category ct ON ct.code = ic.category_code
        /* (8) */
        LEFT JOIN t_order_coupon oc ON oc.order_id = o.id
        LEFT JOIN m_coupon cp ON cp.code = oc.coupon_code
    </sql>

    <!-- (9) -->
    <select id="findOne" parameterType="_int" resultMap="orderResultMap">
        <bind name="orderTable" value="'t_order'" />
        <include refid="selectFromJoin"/>
        WHERE
            o.id = #{id}
        ORDER BY
            item_code ASC,
            category_code ASC,
            coupon_code ASC
    </select>

    <!-- (10) -->
    <select id="findPage" resultMap="orderResultMap">
        <bind name="orderTable" value="
            '(
              SELECT
                  *
              FROM
                  t_order
              ORDER BY
                  id DESC
              LIMIT #{pageable.pageSize}
              OFFSET #{pageable.offset}
              )'" />
```

```
        <include refid="selectFromJoin"/>
        ORDER BY
            id DESC,
            item_code ASC,
            category_code ASC,
            coupon_code ASC
    </select>

    <!-- ... -->

</mapper>
```

Sr. No.	Description
1.	Implement SELECT clause, FROM clause and JOIN clause for `findOne` method and `findPage` method. In the above example, the common location for `findOne` method and `findPage` method is standardized.
2.	Fetch the data required for generating an Order object.
3.	Fetch the data required for generating an OrderStatus object. Care must be taken to avoid duplicating the column name to be fetched. In the above example, `name` column is duplicated. Hence, an alias name (`status_` prefix) is specified using AS clause.
4.	Fetch the data required for generating an OrderItem object and Item object. Care must be taken to avoid duplication of column name to be fetched. In the above example, `code`,`name` and `price` are duplicated. Hence, an alias name (`item_` prefix) is specified using AS clause.
5.	Fetch the data required for generating Category object. Care must be taken to avoid duplication of column name to be fetched. In the above example, `code` and `name` are duplicated. Hence, an alias name (`category_` prefix) is specified using AS clause.
6.	Fetch the data required for generating an OrderCoupon object and Coupon object. Care must be taken to avoid duplication of column name to be fetched. In the above example, `code`,`name` and `price`are duplicated. Hence, alias name (`coupon_` prefix) is specified using AS clause.
7.	Perform JOIN for the tables that store data required for generating related objects.
8.	Perform Outer JOIN for the table wherein records might not be stored. When coupon is not used, it is necessary to perform Outer JOIN since records are not stored in t_order_coupon. This is also applicable for t_coupon to be joined with t_order_coupon.
9.	Implement the SQL for `findOne` method. Specify alignment sequence of the Entity with 1:N relation, in ORDER BY clause. In the above example, sorting is done in the ascending order of PK.
10.	Implement SQL for `findPage` method. Specify alignment sequence of Entity that has a 1:N relation with Order, in ORDER BY clause. In the above example, Order is sorted in descending sequence of PK (new sequence) and related Entity is sorted in the ascending order of PK.

Tip: When it is necessary to perform a pagination search while fetching related Entities with 1:N relation together by a single SQL, `RowBounds` provided by MyBatis3 cannot be used.

Following can be considered as alternative methods.

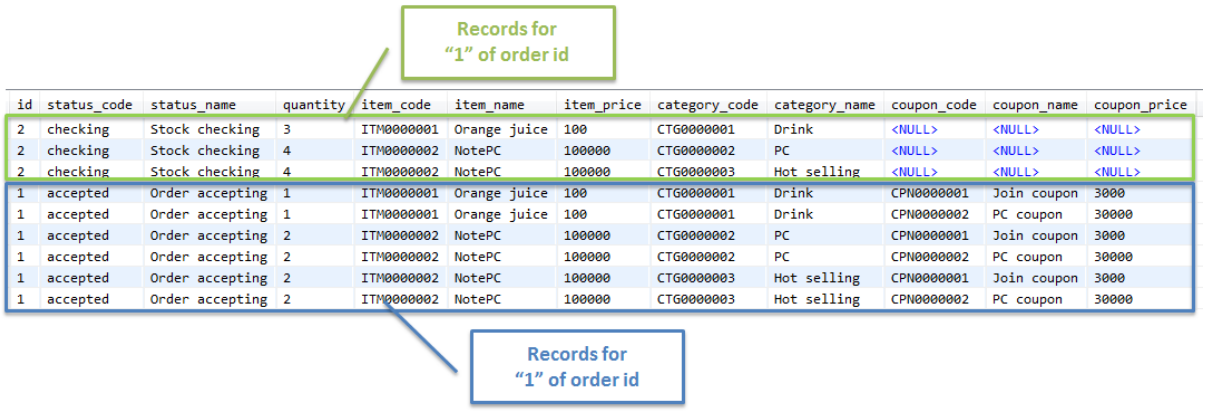
- At first, a method is called which searches only main Entity and related Entities are then fetched by calling a separate method.
- A virtual table that stores only main Entity within the page range is created by SQL and all the records necessary for mapping are fetched by performing JOIN on the records of virtual table (`findPage` shown in the above example is implemented using this pattern)

If SQL (findPage) above is executed, following records are fetched. The order records are 2. However, a total of 9 records are fetched since the records are joined with a related table that stores multiple records.

The breakdown is

- 1st to 3rd rows are records for generating `Order` object with order ID `2`
- 4th to 9th rows are records for generating `Order` object with order ID `1`

How to map search results (`ResultSet`) to JavaBean is explained below using a record wherein order ID is `1`.



id	status_code	status_name	quantity	item_code	item_name	item_price	category_code	category_name	coupon_code	coupon_name	coupon_price
2	checking	Stock checking	3	ITM0000001	Orange juice	100	CTG0000001	Drink	<NULL>	<NULL>	<NULL>
2	checking	Stock checking	4	ITM0000002	NotePC	100000	CTG0000002	PC	<NULL>	<NULL>	<NULL>
2	checking	Stock checking	4	ITM0000002	NotePC	100000	CTG0000003	Hot selling	<NULL>	<NULL>	<NULL>
1	accepted	Order accepting	1	ITM0000001	Orange juice	100	CTG0000001	Drink	CPN0000001	Join coupon	3000
1	accepted	Order accepting	1	ITM0000001	Orange juice	100	CTG0000001	Drink	CPN0000002	PC coupon	30000
1	accepted	Order accepting	2	ITM0000002	NotePC	100000	CTG0000002	PC	CPN0000001	Join coupon	3000
1	accepted	Order accepting	2	ITM0000002	NotePC	100000	CTG0000002	PC	CPN0000002	PC coupon	30000
1	accepted	Order accepting	2	ITM0000002	NotePC	100000	CTG0000003	Hot selling	CPN0000001	Join coupon	3000
1	accepted	Order accepting	2	ITM0000002	NotePC	100000	CTG0000003	Hot selling	CPN0000002	PC coupon	30000

Figure.6.8 **Picture - Result Set of findPage**

Implementation of mapping

The definition for mapping the records described above in the `Order` object and related Entity is given below.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
        "http://mybatis.org/dtd/mybatis-3-mapper.dtd" >
<mapper namespace="com.example.domain.repository.order.OrderRepository">

    <!-- ... -->

    <!-- (1) -->
    <resultMap id="orderResultMap" type="Order">
        <id property="id" column="id"/>
        <!-- (2) -->
        <result property="orderStatus.code" column="status_code" />
        <result property="orderStatus.name" column="status_name" />
        <!-- (3) -->
        <collection property="orderItems" ofType="OrderItem">
            <id property="orderId" column="id"/>
            <id property="item.code" column="item_code"/>
            <result property="quantity" column="quantity"/>
            <association property="item" resultMap="itemResultMap"/>
        </collection>
        <!-- (4) -->
        <collection property="orderCoupons" ofType="OrderCoupon"
                    notNullColumn="coupon_code">
            <id property="orderId" column="id"/>
            <!-- (5) -->
            <id property="coupon.code" column="coupon_code"/>
            <result property="coupon.name" column="coupon_name"/>
            <result property="coupon.price" column="coupon_price"/>
        </collection>
    </resultMap>

    <!-- (6) -->
    <resultMap id="itemResultMap" type="Item">
        <id property="code" column="item_code"/>
        <result property="name" column="item_name"/>
        <result property="price" column="item_price"/>
        <!-- (7) -->
        <collection property="categories" ofType="Category">
            <id property="code" column="category_code"/>
            <result property="name" column="category_name"/>
        </collection>
    </resultMap>

</mapper>
```

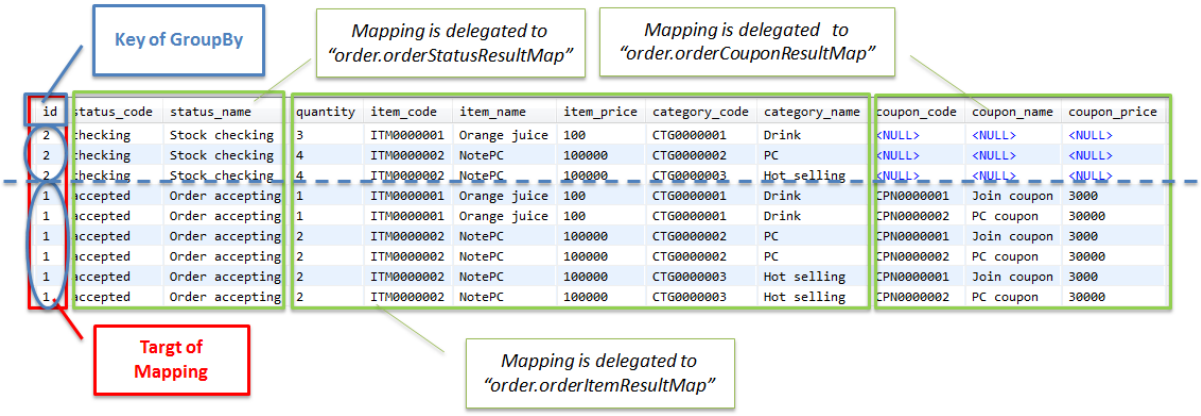
Sr. No.	Description
1.	Definition to map fetched records in `Order` object. Perform mapping of related Entities (`OrderStatus`, `OrderItem` and `OrderCoupon`).
2.	Definition to map fetched records in `OrderStatus` object.
3.	Definition to map fetched records in `OrderItem` object. Mapping to related Entities (`Item`) is delegated to another `resultMap`(6).
4.	Definition to map fetched records in `OrderCoupon` object.
5.	Definition to map fetched records in `Coupon` object.
6.	Definition to map fetched records in `Item` object.
7.	Definition to map fetched records in `Category`object.

Mapping to Order object

Perform mapping to `Order` object.

```
<!-- (1) -->
<resultMap id="orderResultMap" type="Order">
    <!-- (2) -->
    <id property="id" column="id"/>
    <result property="orderStatus.code" column="status_code" />
    <result property="orderStatus.name" column="status_name" />
    <collection property="orderItems" ofType="OrderItem">
        <id property="orderId" column="id"/>
        <id property="item.code" column="item_code"/>
        <result property="quantity" column="quantity"/>
        <association property="item" resultMap="itemResultMap"/>
    </collection>
    <collection property="orderCoupons" ofType="OrderCoupon"
                notNullColumn="coupon_code">
        <id property="orderId" column="id"/>
        <id property="coupon.code" column="coupon_code"/>
        <result property="coupon.name" column="coupon_name"/>
        <result property="coupon.price" column="coupon_price"/>
    </collection>
```

```
</resultMap>
```



id	status_code	status_name	quantity	item_code	item_name	item_price	category_code	category_name	coupon_code	coupon_name	coupon_price
2	checking	Stock checking	3	ITM0000001	Orange juice	100	CTG0000001	Drink	<NULL>	<NULL>	<NULL>
2	checking	Stock checking	4	ITM0000002	NotePC	100000	CTG0000002	PC	<NULL>	<NULL>	<NULL>
2	checking	Stock checking	4	ITM0000002	NotePC	100000	CTG0000003	Hot selling	<NULL>	<NULL>	<NULL>
1	accepted	Order accepting	1	ITM0000001	Orange juice	100	CTG0000001	Drink	CPN0000001	Join coupon	3000
1	accepted	Order accepting	1	ITM0000001	Orange juice	100	CTG0000001	Drink	CPN0000002	PC coupon	30000
1	accepted	Order accepting	2	ITM0000002	NotePC	100000	CTG0000002	PC	CPN0000001	Join coupon	3000
1	accepted	Order accepting	2	ITM0000002	NotePC	100000	CTG0000002	PC	CPN0000002	PC coupon	30000
1	accepted	Order accepting	2	ITM0000002	NotePC	100000	CTG0000003	Hot selling	CPN0000001	Join coupon	3000
1	accepted	Order accepting	2	ITM0000002	NotePC	100000	CTG0000003	Hot selling	CPN0000002	PC coupon	30000

Figure.6.9 **Picture - ResultMap for Order**

Sr. No.	Description
1.	Map search results to `Order` object. Specify the class to be mapped in `type` attribute.
2.	Value of `id` column of fetched records is set in `Order#id` property. Since `id` column is PK, mapping is performed by using `id` element. If `id` element is used, records are grouped by the specified property values. Basically, they are grouped in two groups - `id=1` and `id=2`. Two `Order` objects are generated.

Mapping to OrderStatus object

Perform mapping to `OrderStatus` object.

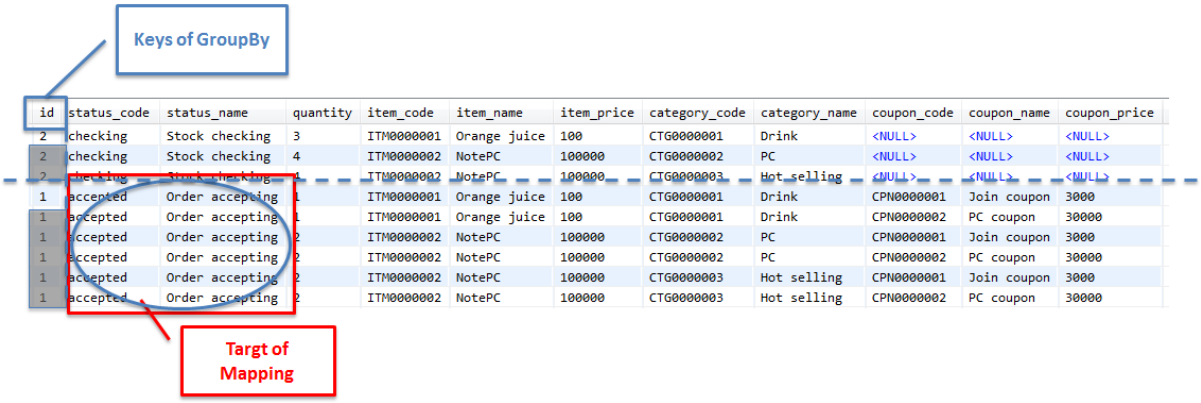
Note: For creating Entity class of "*Implementation of Entity*", "Code table is handled as a basic type of java.lang.String rather than handling as an Entity.". This is because the data stored in the code table uses a different mechanism like "*Codelist*" etc.

The purpose of this section is to explain how to map to a related Entity (JavaBean), hence the explanation on code table also being handled as an Entity is added.

In the actual project, it is recommended to create an Entity based on the policies for creating an Entity class.

```
<resultMap id="orderResultMap" type="Order">
    <id property="id" column="id"/>
```

```
    <!-- (1) -->
    <result property="orderStatus.code" column="status_code" />
    <!-- (2) -->
    <result property="orderStatus.name" column="status_name" />
    <collection property="orderItems" ofType="OrderItem">
        <id property="orderId" column="id"/>
        <id property="item.code" column="item_code"/>
        <result property="quantity" column="quantity"/>
        <association property="item" resultMap="itemResultMap"/>
    </collection>
    <collection property="orderCoupons" ofType="OrderCoupon"
                notNullColumn="coupon_code">
        <id property="orderId" column="id"/>
        <id property="coupon.code" column="coupon_code"/>
        <result property="coupon.name" column="coupon_name"/>
        <result property="coupon.price" column="coupon_price"/>
    </collection>
</resultMap>
```



id	status_code	status_name	quantity	item_code	item_name	item_price	category_code	category_name	coupon_code	coupon_name	coupon_price
2	checking	Stock checking	3	ITM0000001	Orange juice	100	CTG0000001	Drink	<NULL>	<NULL>	<NULL>
2	checking	Stock checking	4	ITM0000002	NotePC	100000	CTG0000002	PC	<NULL>	<NULL>	<NULL>
2	checking	Stock checking	1	ITM0000002	NotePC	100000	CTG0000003	Hot selling	<NULL>	<NULL>	<NULL>
1	accepted	Order accepting	1	ITM0000001	Orange juice	100	CTG0000001	Drink	CPN0000001	Join coupon	3000
1	accepted	Order accepting	1	ITM0000001	Orange juice	100	CTG0000001	Drink	CPN0000002	PC coupon	30000
1	accepted	Order accepting	2	ITM0000002	NotePC	100000	CTG0000002	PC	CPN0000001	Join coupon	3000
1	accepted	Order accepting	2	ITM0000002	NotePC	100000	CTG0000002	PC	CPN0000002	PC coupon	30000
1	accepted	Order accepting	2	ITM0000002	NotePC	100000	CTG0000003	Hot selling	CPN0000001	Join coupon	3000
1	accepted	Order accepting	2	ITM0000002	NotePC	100000	CTG0000003	Hot selling	CPN0000002	PC coupon	30000

Figure.6.10 **Picture - ResultMap for OrderStatus**

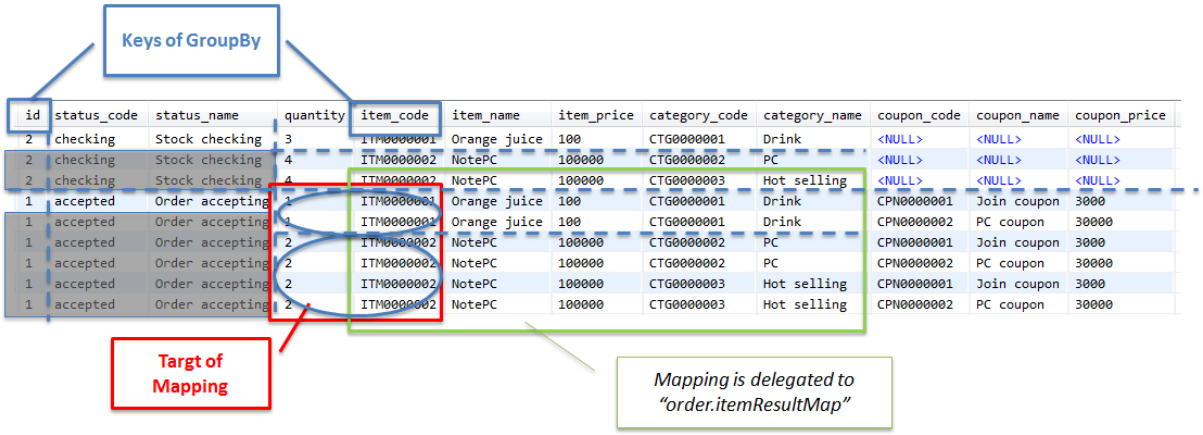
Sr. No.	Description
1.	Set the value of `status_code` column of fetched records in `OrderStatus#code` property.
2.	Set the value of `status_name` column of fetched records in `OrderStatus#name` property.

Note: The value of records that are grouped in `id` column is set in `OrderStatus` object.

Mapping to OrderItem object

Perform mapping to `OrderItem` object.

```
<resultMap id="orderResultMap" type="Order">
    <id property="id" column="id"/>
    <result property="orderStatus.code" column="status_code" />
    <result property="orderStatus.name" column="status_name" />
    <!-- (1) -->
    <collection property="orderItems" ofType="OrderItem">
        <!-- (2) -->
        <id property="orderId" column="id"/>
        <!-- (3) -->
        <id property="item.code" column="item_code"/>
        <!-- (4) -->
        <result property="quantity" column="quantity"/>
        <!-- (5) -->
        <association property="item" resultMap="itemResultMap"/>
    </collection>
    <collection property="orderCoupons" ofType="OrderCoupon"
                notNullColumn="coupon_code">
        <id property="orderId" column="id"/>
        <id property="coupon.code" column="coupon_code"/>
        <result property="coupon.name" column="coupon_name"/>
        <result property="coupon.price" column="coupon_price"/>
    </collection>
</resultMap>
```



id	status_code	status_name	quantity	item_code	item_name	item_price	category_code	category_name	coupon_code	coupon_name	coupon_price
2	checking	Stock checking	3	ITM0000001	Orange juice	100	CTG0000001	Drink	<NULL>	<NULL>	<NULL>
2	checking	Stock checking	4	ITM0000002	NotePC	100000	CTG0000002	PC	<NULL>	<NULL>	<NULL>
2	checking	Stock checking	4	ITM0000003	NotePC	100000	CTG0000003	Hot selling	<NULL>	<NULL>	<NULL>
1	accepted	Order accepting	1	ITM0000001	Orange juice	100	CTG0000001	Drink	CPN0000001	Join coupon	3000
1	accepted	Order accepting	1	ITM0000001	Orange juice	100	CTG0000001	Drink	CPN0000002	PC coupon	30000
1	accepted	Order accepting	2	ITM0000002	NotePC	100000	CTG0000002	PC	CPN0000001	Join coupon	3000
1	accepted	Order accepting	2	ITM0000002	NotePC	100000	CTG0000002	PC	CPN0000002	PC coupon	30000
1	accepted	Order accepting	2	ITM0000002	NotePC	100000	CTG0000003	Hot selling	CPN0000001	Join coupon	3000
1	accepted	Order accepting	2	ITM0000002	NotePC	100000	CTG0000003	Hot selling	CPN0000002	PC coupon	30000

Figure.6.11 **Picture - ResultMap for OrderItem**

Sr. No.	Description
1.	Map search results in `OrderItem` object and add to `Order#orderItems` property. When search results are to be mapped to the related Entities with 1:N relation, `collection` element is used. Refer to "MyBatis3 REFERENCE DOCUMENTATION(Mapper XML Files-collection-)" for details of "`collection` element.
2.	Set the value of `id` column of fetched records in `OrderItem#orderId` property. Since `id` column is PK, the mapping is performed by using `id` element.
3.	Set the value of `item_code` column of fetched records in `Item#code` property. Since `item_code`column is PK, the mapping is performed by using `id` element. If `id` element is used, the records are grouped in the specified property value. Basically, they are grouped in two groups - `Item#code=ITM0000001` and `Item#code=ITM0000002`. Two `OrderItem` objects are generated.
4.	Set the value of `quantity` column of fetched records in `OrderItem#quantity` property.
5.	Generation of `Item` object is delegated to a different `resultMap`, and the generated object is set in `OrderItem#item` property. For actual mapping, refer to "*Mapping to Item object*" . When mapping is to be performed in the related Entities with a 1:1 relation, `association` element is used. For details of `association` element, refer to "MyBatis3 REFERENCE DOCUMENTATION (Mapper XML Files-association-)" .

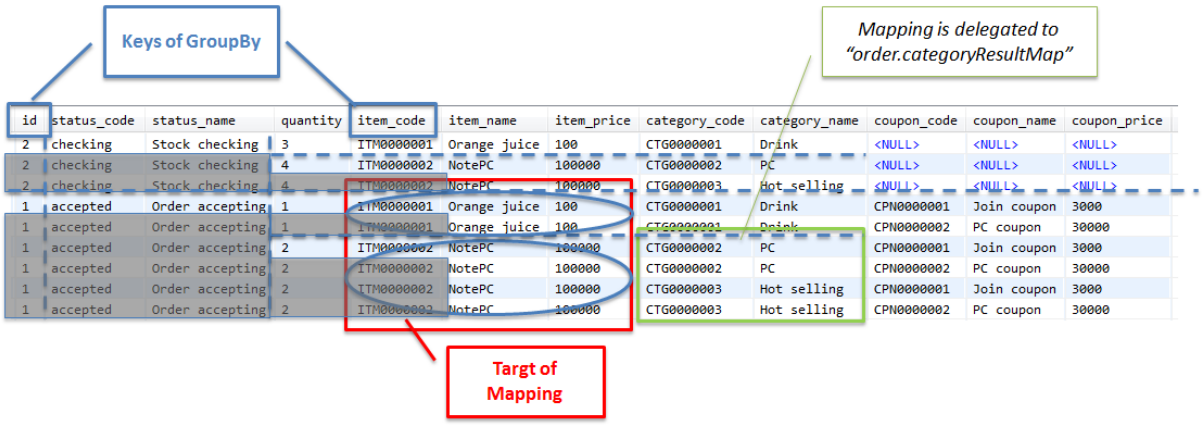
Note: In `OrderItem` object, values of records that are grouped in `id` column and `item_code` column are set.

Mapping to Item object

Perform mapping to `Item` object.

```
<!-- (1) -->
<resultMap id="itemResultMap" type="Item">
    <!-- (2) -->
    <id property="code" column="item_code"/>
    <!-- (3) -->
    <result property="name" column="item_name"/>
    <!-- (4) -->
    <result property="price" column="item_price"/>
```

```
    <collection property="categories" ofType="Category">
        <id property="code" column="category_code"/>
        <result property="name" column="category_name"/>
    </collection>
</resultMap>
```



id	status_code	status_name	quantity	item_code	item_name	item_price	category_code	category_name	coupon_code	coupon_name	coupon_price
2	checking	Stock checking	3	ITM0000001	Orange juice	100	CTG0000001	Drink	<NULL>	<NULL>	<NULL>
2	checking	Stock checking	4	ITM0000002	NotePC	100000	CTG0000002	PC	<NULL>	<NULL>	<NULL>
2	checking	Stock checking	4	ITM0000002	NotePC	100000	CTG0000003	Hot selling	<NULL>	<NULL>	<NULL>
1	accepted	Order accepting	1	ITM0000001	Orange juice	100	CTG0000001	Drink	CPN0000001	Join coupon	3000
1	accepted	Order accepting	1	ITM0000001	Orange juice	100	CTG0000001	Drink	CPN0000002	PC coupon	30000
1	accepted	Order accepting	2	ITM0000002	NotePC	100000	CTG0000002	PC	CPN0000001	Join coupon	3000
1	accepted	Order accepting	2	ITM0000002	NotePC	100000	CTG0000002	PC	CPN0000002	PC coupon	30000
1	accepted	Order accepting	2	ITM0000002	NotePC	100000	CTG0000003	Hot selling	CPN0000001	Join coupon	3000
1	accepted	Order accepting	2	ITM0000002	NotePC	100000	CTG0000003	Hot selling	CPN0000002	PC coupon	30000

Figure.6.12 **Picture - ResultMap for Item**

Sr. No.	Description
1.	Map search results to `Item` object. Specify the class to be mapped in `type` attribute.
2.	Set the value of `item_code` column of fetched records in `Item#code`. Since `item_code` column is PK, mapping is performed by using `id` element.
3.	Set the value of `item_name` column of fetched records in `Item#name`.
4.	Set the value of `item_price` column of fetched records in `Item#price`.

Note: In `Item` object, values of records that are grouped in `id` column and `item_code` column are set.

Mapping to Category object

Perform mapping to `Category` object.

```
<resultMap id="itemResultMap" type="Item">
    <id property="code" column="item_code"/>
    <result property="name" column="item_name"/>
    <result property="price" column="item_price"/>
    <!-- (1) -->
    <collection property="categories" ofType="Category">
        <!-- (2) -->
        <id property="code" column="category_code"/>
        <!-- (3) -->
        <result property="name" column="category_name"/>
    </collection>
</resultMap>
```

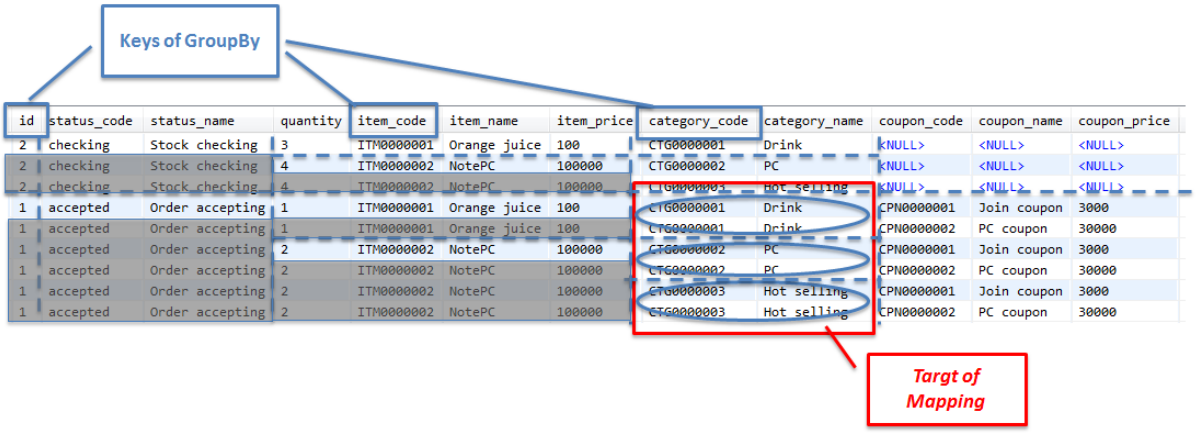


Figure.6.13 **Picture - ResultMap for Category**

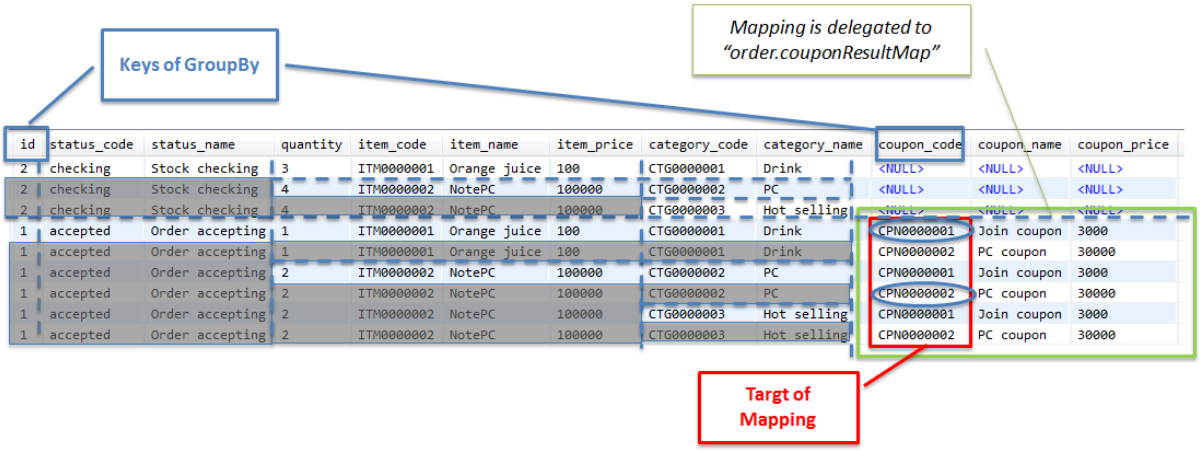
Sr. No.	Description
1.	Map the search results to `Category` object and add to `Item#categories` property. When mapping to related Entities with 1:N relation, `collection` element is used. For details of `collection` element, refer to "MyBatis3 REFERENCE DOCUMENTATION (Mapper XML Files-collection-)" .
2.	Set the value of `category_code` column of fetched records in `Category#code`. Since `category_code` column is PK, the mapping is performed by using `id` element. If `id` element is used, the records are grouped by specified property values. Basically, following objects are generated • `Category` object of `Category#code=CTG0000001` as category of `Item#code=ITM0000001` • Two `Category` objects of `Category#code=CTG0000002` and `Category#code=CTG0000003`as category of `Item#code=ITM0000002`.
3.	Set the value of `category_name` column of fetched records in `Category#name` .

Note: In `Category` object, values of records that are grouped in `id` column, `item_code` column and `category_code` column are set.

Mapping to OrderCoupon object

Perform mapping to `OrderCoupon` object.

```
<resultMap id="orderResultMap" type="Order">
    <id property="id" column="id"/>
    <result property="orderStatus.code" column="status_code" />
    <result property="orderStatus.name" column="status_name" />
    <collection property="orderItems" ofType="OrderItem">
        <id property="orderId" column="id"/>
        <id property="item.code" column="item_code"/>
        <result property="quantity" column="quantity"/>
        <association property="item" resultMap="itemResultMap"/>
    </collection>
    <!-- (1) -->
    <collection property="orderCoupons" ofType="OrderCoupon" notNullColumn=
↪"coupon_code">
        <!-- (2) -->
        <id property="orderId" column="id"/>
        <!-- (3) -->
        <id property="coupon.code" column="coupon_code"/>
        <result property="coupon.name" column="coupon_name"/>
        <result property="coupon.price" column="coupon_price"/>
    </collection>
</resultMap>
```



id	status_code	status_name	quantity	item_code	item_name	item_price	category_code	category_name	coupon_code	coupon_name	coupon_price
2	checking	Stock checking	3	ITM0000001	Orange juice	100	CTG0000001	Drink	<NULL>	<NULL>	<NULL>
2	checking	Stock checking	4	ITM0000002	NotePC	100000	CTG0000002	PC	<NULL>	<NULL>	<NULL>
2	checking	Stock checking	4	ITM0000002	NotePC	100000	CTG0000003	Hot selling	<NULL>	<NULL>	<NULL>
1	accepted	Order accepting	1	ITM0000001	Orange juice	100	CTG0000001	Drink	CPN0000001	Join coupon	3000
1	accepted	Order accepting	1	ITM0000001	Orange juice	100	CTG0000001	Drink	CPN0000002	PC coupon	30000
1	accepted	Order accepting	2	ITM0000002	NotePC	100000	CTG0000002	PC	CPN0000001	Join coupon	3000
1	accepted	Order accepting	2	ITM0000002	NotePC	100000	CTG0000002	PC	CPN0000002	PC coupon	30000
1	accepted	Order accepting	2	ITM0000002	NotePC	100000	CTG0000003	Hot selling	CPN0000001	Join coupon	3000
1	accepted	Order accepting	2	ITM0000002	NotePC	100000	CTG0000003	Hot selling	CPN0000002	PC coupon	30000

Figure.6.14 **Picture - ResultMap for OrderCoupon**

Sr. No.	Description
1.	Map the search results to `OrderCoupon` object and add to `Order#orderCoupons` property. When mapping to related Entities with 1:N relation, `collection` element is used. For details of `collection` element, refer to "MyBatis3 REFERENCE DOCUMENTATION (Mapper XML Files-collection-)" . It is important to note that `notNullColumn` attribute is specified in the above example. This setting is to avoid the generation of `OrderCoupon` object when no records exist in `t_coupon` table. In this implementation example, records of `t_coupon` table are not stored in the data with `id` column `2`, hence from the search results, it is understood that the values of `coupon_code`, `coupon_name` and `coupon_price` are `null`. It is not necessary to specify `notNullColumn` attribute if the columns that are mapped to `OrderCoupon` object are restricted to the three columns described above. However, in the implementation example, since the value of `id` column is mapped in `OrderCoupon#orderId` property, `notNullColumn` attribute should be specified. This is because MyBatis generates an object when the value which is not `null` is set in the column for mapping. As shown in the above example, if `coupon_code` column is specified in the `notNullColumn` attribute, the object is generated only when the value of `coupon_code` column is not `null`(in other words, when records exist). Multiple columns can be specified in `notNullColumn`attribute.
2.	Set the value of `id` column of fetched records in `OrderCoupon#orderId` property. Since `orderId` is PK, `id` element is used.
3.	Set the value of `coupon_code` column of fetched records in `Coupon#code`. Since `coupon_code` column is PK, the mapping is performed by using `id` element. If `id`element is used, the records are grouped by the specified property value. Basically, they are grouped in 2 groups - `Coupon#code=CPN0000001` and `Coupon#code=CPN0000002`. Then, two `OrderCoupon` objects are generated.

Mapping to Coupon object

Perform mapping to `Coupon` object.

```
<resultMap id="orderResultMap" type="Order">
    <id property="id" column="id"/>
    <result property="orderStatus.code" column="status_code" />
    <result property="orderStatus.name" column="status_name" />
    <collection property="orderItems" ofType="OrderItem">
        <id property="orderId" column="id"/>
        <id property="item.code" column="item_code"/>
        <result property="quantity" column="quantity"/>
        <association property="item" resultMap="itemResultMap"/>
    </collection>
    <collection property="orderCoupons" ofType="OrderCoupon" notNullColumn=
↪"coupon_code">
        <id property="orderId" column="id"/>
        <!-- (1) -->
        <id property="coupon.code" column="coupon_code"/>
        <!-- (2) -->
        <result property="coupon.name" column="coupon_name"/>
        <!-- (3) -->
        <result property="coupon.price" column="coupon_price"/>
    </collection>
</resultMap>
```

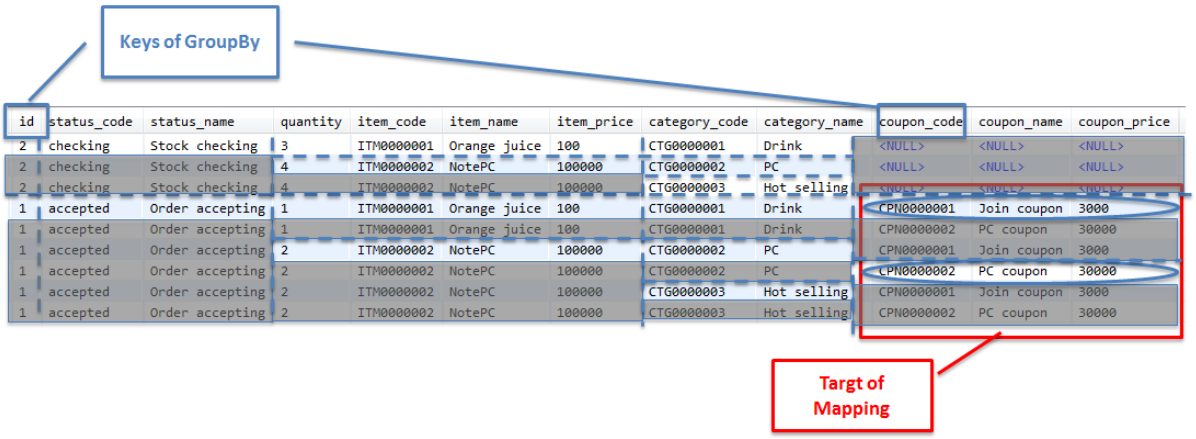


Figure.6.15 **Picture - ResultMap for Coupon**

Sr. No.	Description
1.	Set the value of `coupon_code` column of fetched records in `Coupon#code`.
2.	Set the value of `coupon_name` column of fetched records in `Coupon#name`.
3.	Set the value of `coupon_price` column of fetched records in `Coupon#price`.

Note: In `Coupon` object, values of records that are grouped in `id` column and `coupon_code` column are set.

Object diagram after mapping

The status of `Order` object and related Entities that are actually mapped is as given below.

Records and columns mapped to `Order` object are as follows:

By specifying groupBy attribute, grayed out part is merged with the non-grayed part.

Warning: It should be noted that when the records with 1:N relationship are mapped using JOINs, fetching the data of grayed out part becomes unnecessary.

If the same SQL is used in a process that does not use data from the N part, it results in fetching of unnecessary data. Therefore, two separate SQLs, one that fetches the N part and another that does not fetch the N part should be created.

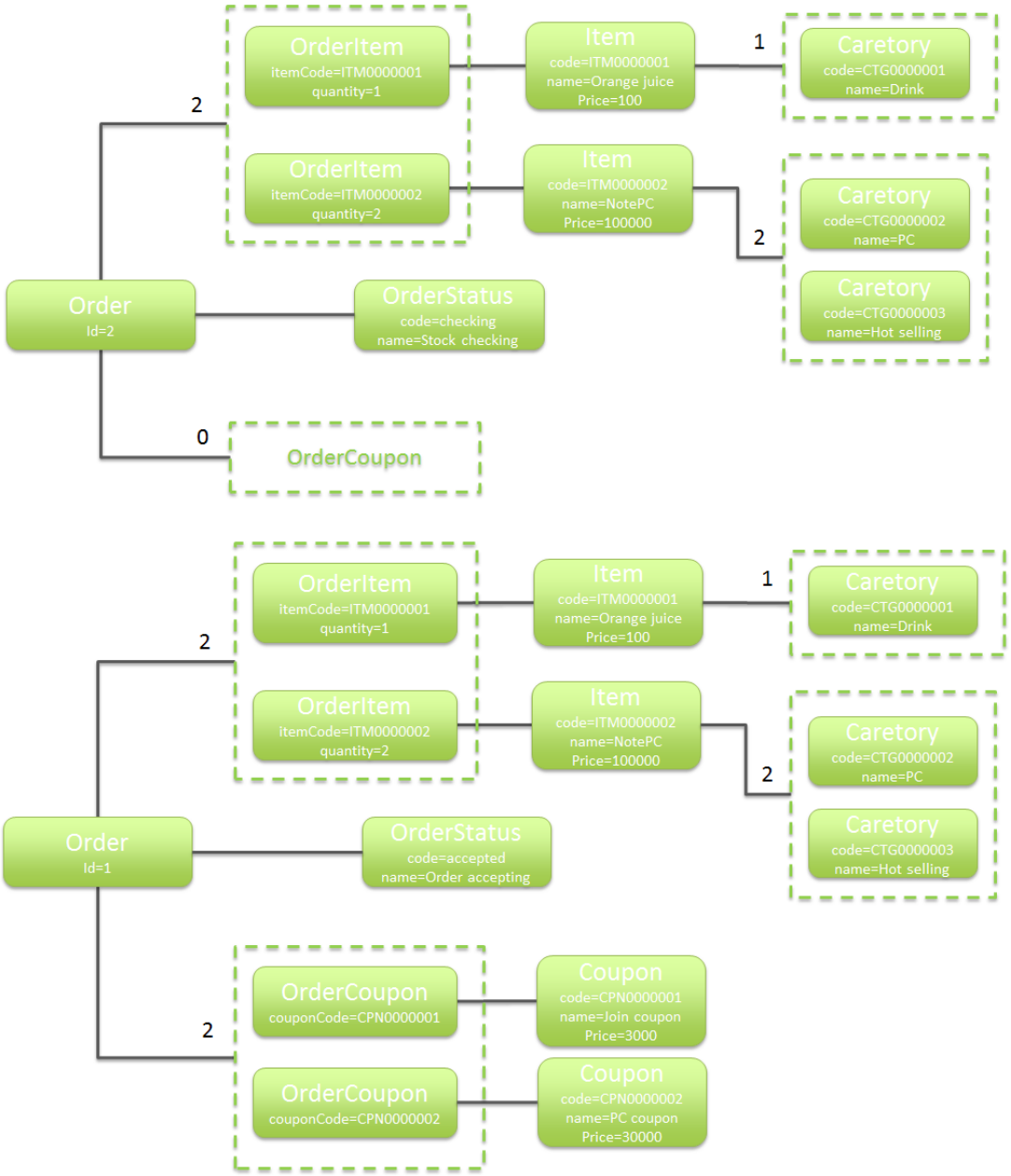


Figure.6.16 **Picture - Mapped object diagram**

id	status_code	status_name	quantity	item_code	item_name	item_price	category_code	category_name	coupon_code	coupon_name	coupon_price
2	checking	Stock checking	3	ITM0000001	Orange juice	100	CTG0000001	Drink	<NULL>	<NULL>	<NULL>
2	checking	Stock checking	4	ITM0000002	NotePC	100000	CTG0000002	PC	<NULL>	<NULL>	<NULL>
2	checking	Stock checking	4	ITM0000002	NotePC	100000	CTG0000003	Hot selling	<NULL>	<NULL>	<NULL>
1	accepted	Order accepting	1	ITM0000001	Orange juice	100	CTG0000001	Drink	CPN0000001	Join coupon	3000
1	accepted	Order accepting	1	ITM0000001	Orange juice	100	CTG0000001	Drink	CPN0000002	PC coupon	30000
1	accepted	Order accepting	2	ITM0000002	NotePC	100000	CTG0000002	PC	CPN0000001	Join coupon	3000
1	accepted	Order accepting	2	ITM0000002	NotePC	100000	CTG0000002	PC	CPN0000002	PC coupon	30000
1	accepted	Order accepting	2	ITM0000002	NotePC	100000	CTG0000003	Hot selling	CPN0000001	Join coupon	3000
1	accepted	Order accepting	2	ITM0000002	NotePC	100000	CTG0000003	Hot selling	CPN0000002	PC coupon	30000

Figure.6.17 **Picture - Valid Result Set**

How to fetch a related Entity using a nested SQL

MyBatis3 provides a method wherein a related Entity is fetched at the time of mapping by using a different SQL (nested SQL).

When a related Entity is fetched by using a nested SQL, following can be performed easily.

- Individual SQL definition
- Mapping definition of `resultMap` element

> **Warning:** It should be noted that even if the definitions are simplified, if nested SQL is used extensively, it can cause N+1.
>
> Default MyBatis3 behavior while using nested SQL is "Eager Load". It signifies that SQL is executed regardless of whether related Entities are used and the following risks are more likely to occur.
>
> - Unnecessary SQL execution and data fetching
> - N+1

Tip: MyBatis3 provides an option to change the behavior to "Lazy Load" while fetching a related Entity using a nested SQL.

Refer to "*Settings to apply Lazy Load for related Entity*" for how to use "Lazy Load".

How to fetch a related Entity using a nested SQL

Implementation example wherein a related Entity is fetched by using a nested SQL is shown below.

```
<resultMap id="itemResultMap" type="Item">
    <id property="code" column="item_code"/>
    <result property="name" column="item_name"/>
    <result property="price" column="item_price"/>
    <!-- (1) -->
    <collection property="categories" column="item_code"
        select="findAllCategoryByItemCode" />
</resultMap>

<select id="findAllCategoryByItemCode"
```

```
    parameterType="string" resultType="Category">
    SELECT
        ct.code,
        ct.name
    FROM
        m_item_category ic
    INNER JOIN m_category ct ON ct.code = ic.category_code
    WHERE
        ic.item_code = #{itemCode}
    ORDER BY
        code
</select>
```

Sr. No.	Description
1.	Specify the statement ID of an SQL to be called in `select` attribute of `association` element or `collection` element. Specify the column name that stores the parameter value to be passed to SQL, in `column` attribute. In the above example, value of `item_code` column is passed as a parameter of `findAllCategoryByItemCode`. For details of attributes that can be specified, refer to "MyBatis3 REFERENCE DOCUMENTATION(Mapper XML Files-Nested Select for Association-)" .

Note: In the above example, since `fetchType` attribute is not specified, whether to use "Lazy Load" or "Eager Load" depends on configuration of overall application.

For configuration of overall application, refer to "*MyBatis settings for using Lazy Load*".

Settings to apply Lazy Load for related Entity

Although default behavior of MyBatis3 while fetching a related Entity by using a nested SQL is "Eager Load", "Lazy Load" can also be used.

The minimum necessary settings for using "Lazy Load" and how to use "Lazy Load" are explained below.

For the configuration values that are not explained, refer to "MyBatis3 REFERENCE DOCUMENTATION(Mapper XML Files-settings-)" .

Adding a Bytecode Manipulation Library

When "Lazy Load" is used, one of the libraries given below is necessary in order to generate a Proxy object for implementing "Lazy Load".

- JAVASSIST
- CGLIB

CGLIB had been used as default until MyBatis 3.2 series. From MyBatis 3.3.0, JAVASSIST has been used as default at the MyBatis 3.3.0 or later version. In addition, it is possible to use "Lazy Load" without adding the library because the JAVASSIST is embedded in the MyBatis since MyBatis 3.3.0.

Note: When CGLIB is to be used in MyBatis 3.3.0 or later,

- CGLIB artifact should be added to `pom.xml`
- "`proxyFactory=CGLIB`" should be added to MyBatis configuration file (`projectName-domain/src/main/resources/META-INF/mybatis/mybatis-config.xml`)

Refer to "MyBatis3 PROJECT DOCUMENTATION(Project Dependencies-compile-)" for information about CGLIB artifact.

MyBatis settings for using Lazy Load

In MyBatis3, availability of "Lazy Load" can be specified in the 2 locations given below.

- Overall configuration of application (MyBatis configuration file)
- Individual configuration (mapping file)
- Overall configuration of application is specified in MyBatis configuration file (`projectName-domain/src/main/resources/META-INF/mybatis/mybatis-config.xml`).

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE configuration
        PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
        "http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>
    <settings>
        <!-- (1) -->
        <setting name="lazyLoadingEnabled" value="true"/>
```

```
    </settings>
</configuration>
```

Sr. No.	Description
1.	Specify default behavior of application as `lazyLoadingEnabled`. • `true`: "Lazy Load" • `false`: "Eager Load" (Default) When `fetchType` attribute of `association` element and `collection` element is to be specified, the specified value of `fetchType` attribute is given priority.

Warning: When `select` attribute of `association` element or `collection` element is used in "`false`: "Eager Load"" state, exercise caution as SQL is executed at the time of mapping.

It is recommended to set `lazyLoadingEnabled` to `true` unless there is a specific reason.

- Individual settings are specified in `fetchType` attribute of `association` element and `collection` element of mapping file.

```
    <resultMap id="itemResultMap" type="Item">
        <id property="code" column="item_code"/>
        <result property="name" column="item_name"/>
        <result property="price" column="item_price"/>
        <!-- (2) -->
        <collection property="categories" column="item_code"
            fetchType="lazy"
            select="findAllCategoryByItemCode" />
    </resultMap>

    <select id="findAllCategoryByItemCode"
        parameterType="string" resultType="Category">
        SELECT
            ct.code,
            ct.name
        FROM
            m_item_category ic
        INNER JOIN m_category ct ON ct.code = ic.category_code
        WHERE
            ic.item_code = #{itemCode}
        ORDER BY
            code
    </select>
```

Sr. No.	Description
2.	Specify `lazy` or `eager` in `fetchType` attribute of `association` element or `collection` element. If `fetchType` attribute is specified, overall configuration of application can be over-written.

Settings for controlling execution timing of Lazy Load

MyBatis3 provides an option to control the timings to execute "Lazy Load" (`aggressiveLazyLoading`) [*1].

The default value of this option is `false` in Mybatis 3.4.2 and subsequent versions and it is executed within the timing when a getter method of "Lazy Load" property is called.

Warning: When `aggressiveLazyLoading` is "`true`", "Lazy Load" is executed within the timing when the getter method of the object which retains "Lazy Load" property is called. Therefore, it should be noted that SQL is likely to be executed regardless of the fact that data fetching is not actually required.

Basically, there is a case wherein the following mapping is carried out and only the property for which "Lazy load" is not to be performed, is accessed. In case of "`true`", "Lazy Load" is executed even without directly accessing the property for which "Lazy Load" is to be performed.

It is recommended to keep `aggressiveLazyLoading` to "`false`" (default) unless there is a specific reason to change otherwise.

- Entity

```
public class Item implements Serializable {
    private static final long serialVersionUID = 1L;
    private String code;
    private String name;
    private int price;
    private List<Category> categories;
    // ...
}
```

- Mapping file

```
<resultMap id="itemResultMap" type="Item">
    <id property="code" column="item_code"/>
    <result property="name" column="item_name"/>
    <result property="price" column="item_price"/>
    <collection property="categories" column="item_code"
        fetchType="lazy" select="findByItemCode" />
</resultMap>
```

[*1] For how to set, refer MyBatis reference.

- Application code (Service)

```
    Item item = itemRepository.findOne(itemCode);
    // (1)
    String code = item.getCode();
    String name = item.getName();
    String price = item.getPrice();
    // ...
}
```

Sr. No.	Description
1.	In the above example, `categories` property targeted for "Lazy Load" is not accessed. However, "Lazy Load" is executed while accessing `Item#code` property. In case of "`false`" (default), "Lazy Load" is not executed in the cases described above.

6.3 Database Access (JPA)

Todo

TBD

The following topics in this chapter are currently under study.

- `persistence.xml` settings
- Implementing dynamic query using QueryDSL
- Examples of cases wherein it is desirable to change the default setting values for specifying fetch method of related-entities.
- Using multiple PersistenceUnits
- Using Native query

6.3.1 Overview

This section explains the method to access the database using JPA.

In this guideline, it is assumed that Hibernate is used as JPA provider and Spring Data JPA as JPA wrapper.

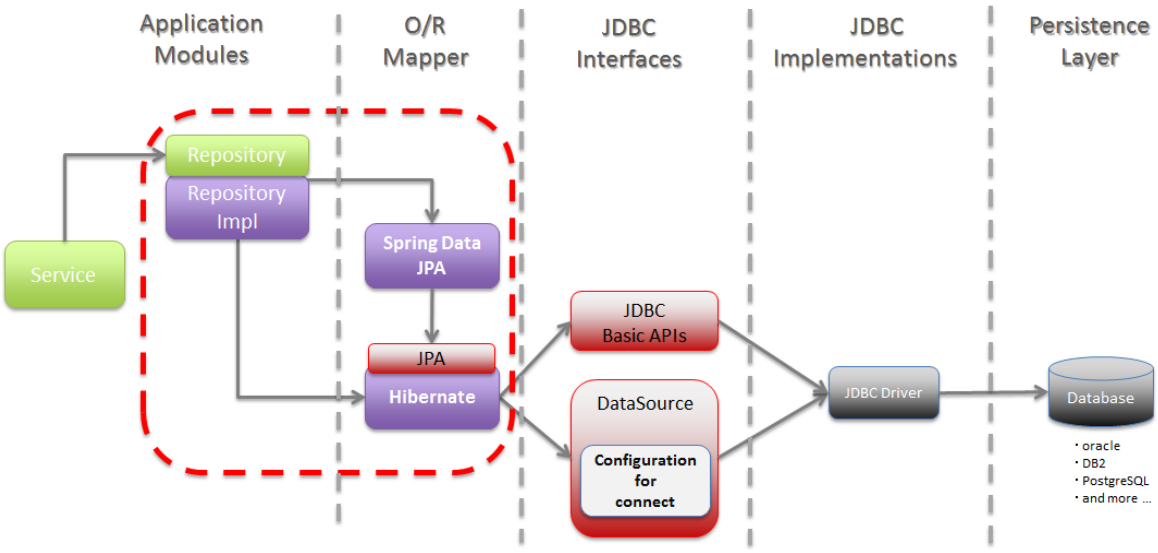


Figure.6.18 **Picture - Target of description**

Warning: The contents described in this chapter may not be applicable for JPA providers other than Hibernate such as EclipseLink etc.

About JPA

JPA (Java Persistence API) defines the following as an API of Java:

1. a way of mapping the records in a relational database, with the java objects
2. a mechanism for reflecting the operations done on the java object, to the records in a relational database.

JPA defines only specifications, it does not provide implementation.

JPA implementation is provided as a reference implementation by the vendors developing O/R Mapper such as Hibernate.

The reference implementation provided by the vendors developing O/R Mapper is called JPA Provider.

O/R Mapping of JPA

Mapping of Java objects to the relational database records at the time of using JPA is as follows:

In JPA, if the value stored in the "managed" entity is changed (by calling setter method), there is a mechanism to reflect the changes in the relational database.

This mechanism is quite similar to the client software such as Table viewer with Edit functionality.

In client software such as Table viewer, if the value of viewer is changed, it is reflected in the database. While in JPA, if the value of Java object (JavaBean) called "Entity" is changed, it is reflected in the database.

Basic JPA terminology

The basic terminology of JPA is described below.

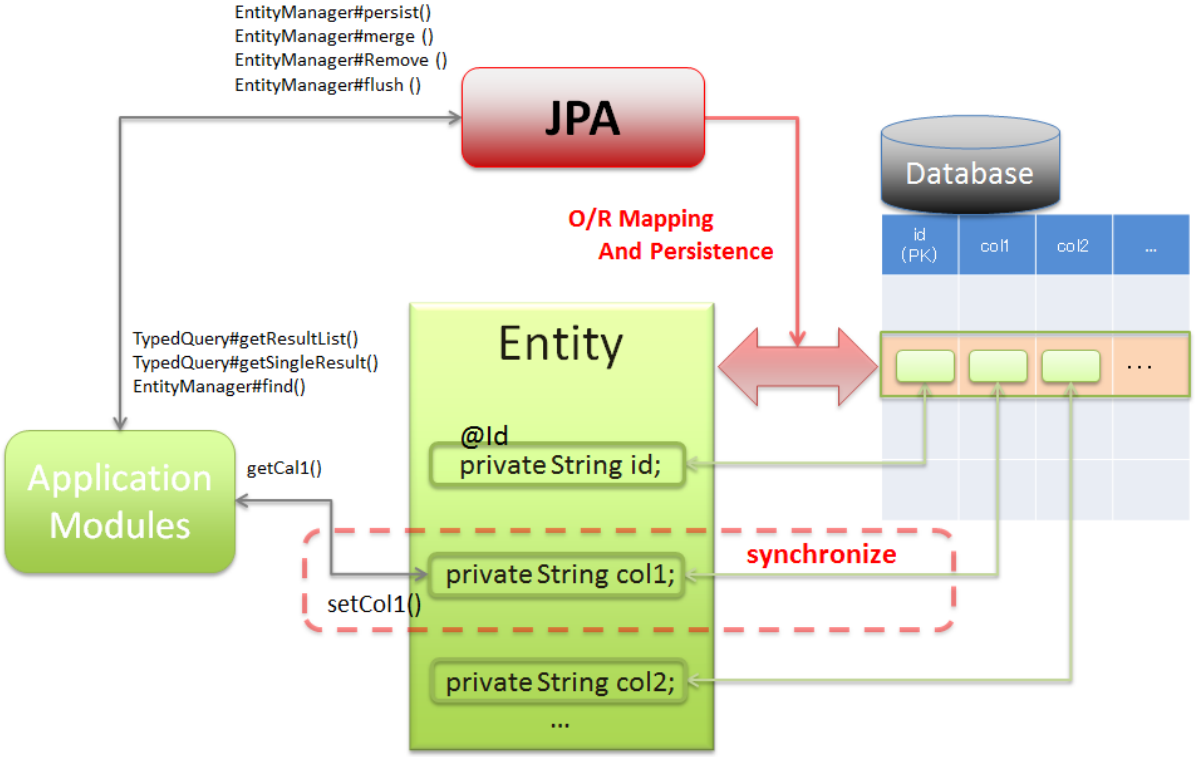


Figure.6.19 **Picture - Image of O/R Mapping**

Sr. No.	Term	Description
1.	Entity class	A Java class representing the records in the relational database The class with `@javax.persistence.Entity` annotation is an Entity class.
2.	EntityManager	An interface which provides API necessary for managing the life cycle of entity Using methods of `javax.persistence.EntityManager`, the application handles the relational database records as Java objects. When using Spring Data JPA, this interface is usually not used directly; however, if it is necessary to generate a query that cannot be expressed using the Spring Data JPA mechanism, then this interface can be used to fetch the entity.

次のページに続く

Table. 6.11 – 前のページからの続き

Sr. No.	Term	Description
3.	TypedQuery	An interface which provides API for searching an entity Using methods of `javax.persistence.TypedQuery`, the application searches for the entity matching the specified conditions other than ID. When using Spring Data JPA, this interface is usually not used directly; however, if it is necessary to generate a query that cannot be expressed using Spring Data JPA mechanism, then this interface can be used to search the entity. The method for directly operating (updating or deleting) the entity of persistence layer (DB) matching the conditions, is also provided in this interface.
4.	PersistenceContext	Area for managing entities The life cycle of an entity that is fetched or created through `EntityManager` is managed by storing that entity in this area. The entity managed in this area is referred to as “Managed Entity”. This area cannot be directly accessed from the application. Apart from “Managed” entity, the other states of entity are “New”, “Removed” and “Detached”.
5.	find method	Method for fetching “managed” entity If the entity corresponding to the ID does not exist in PersistenceContext, get (SELECT) the records stored in the relational database and create a “managed” entity.
6.	persist method	Method for setting “new” entity created in the application, to “managed” entity It is a method of `EntityManager` and all the operations performed to INSERT the records in the relational database are accumulated in PersistenceContext (and later reflected to the database at the time of transaction commit or when the flush method of EntityManager is called).

次のページに続く

Table. 6.11 – 前のページからの続き

Sr. No.	Term	Description
7.	merge method	Method for setting the “detached” entity which is not managed in PersistenceContext, to “managed” entity. It is a method of `EntityManager` and the all operations performed to UPDATE the records stored in a relational database are accumulated in PersistenceContext (and later reflected to the database at the time of transaction commit or when the flush method of EntityManager is called). However, if the record matching the ID does not exist in the relational database, then INSERT is executed instead of UPDATE.
8.	remove method	Method for setting the “managed” entity to “removed” entity It is a method of `EntityManager` and all the operations performed to DELETE the records stored in a relational database are accumulated in PersistenceContext.
9.	flush method	Method for forcibly reflecting the operations performed for the entity managed in PersistenceContext to the relational database. It is a method of `EntityManager` and the accumulated un-reflected operations are reflected in the relational database. Normally, operations are reflected to the relational database only when a transaction is committed; however, flush method is used when the operation needs to be reflected in the database before committing a transaction.

Managing life cycle of entity

The life cycle of entity is managed as follows:

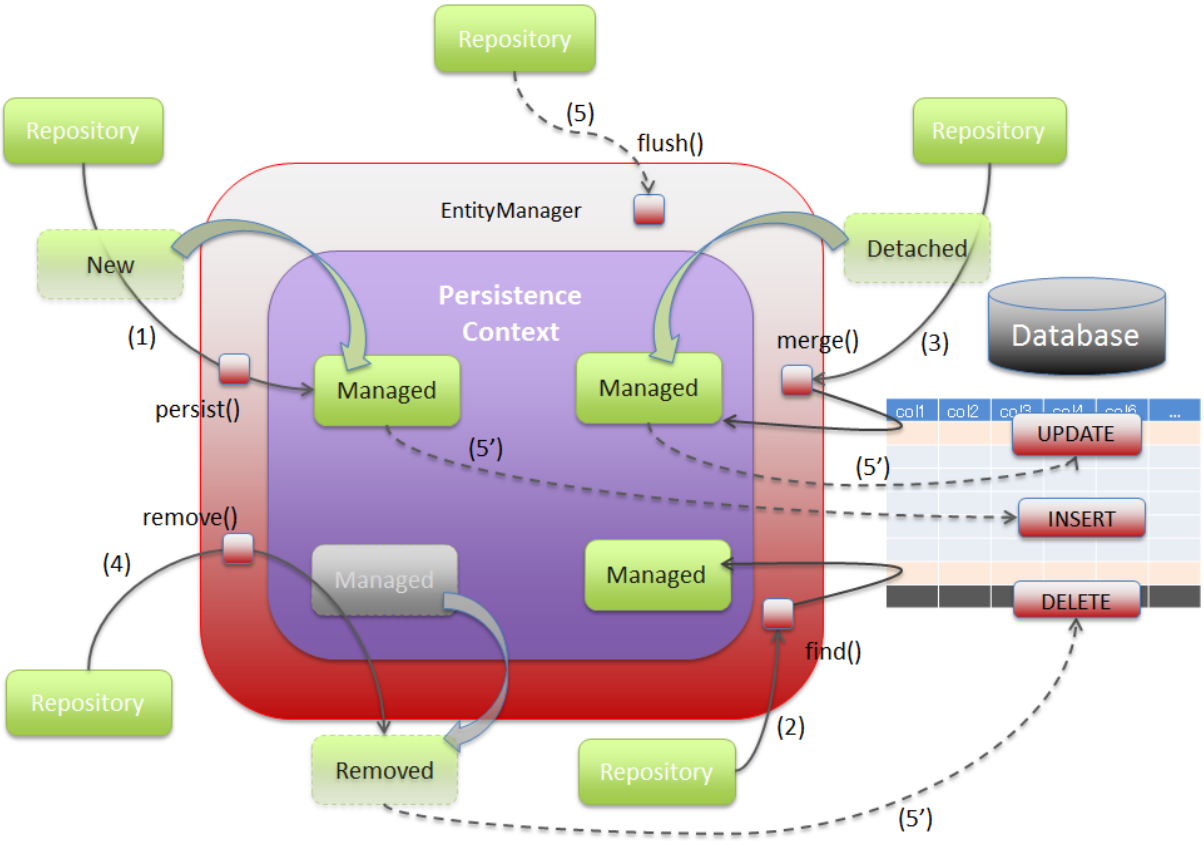


Figure.6.20 **Picture - Life cycle of entity**

Sr. No.	Description
(1)	When persist method of `EntityManager` is called, the entity ("New" entity) passed as an argument is stored in PersistenceContext as "managed" entity.
(2)	When find method of `EntityManager` is called, a "managed" entity with ID passed as an argument, is returned. If it does not exist in PersistenceContext, the records to be mapped are retrieved from the relational database by executing a query and stored as "managed" entity.

次のページに続く

Table. 6.12 – 前のページからの続き

Sr. No.	Description
(3)	When merge method of `EntityManager` is called, the state of the entity (“detached”) passed as an argument, is merged with “managed” entity. If it does not exist in PersistenceContext, the records to be mapped are retrieved from the relational database by executing a query. The state of the entity passed as an argument is merged after the “managed” entity is stored. Note that when this method is called, the entity passed as an argument does not necessarily get stored as “managed” entity unlike persist method.
(4)	When remove method of `EntityManager` is called, the “managed” entity passed as an argument becomes “removed” entity. If this method is called, it is not possible to retrieve the “removed” entity.
(5)	When flush method of `EntityManager` is called, the operations of the entity accumulated using persist, merge and remove methods are reflected in the relational database. By calling this method, the changes done for an entity are synchronized with the records of relational database. However, the changes made only for the records of relational database are not synchronized with the entity. If the entity is searched by executing a query without using find method of `EntityManager`, then prior to the search process, a process similar to flush method is executed in the internal logic of `EntityManager` and the operations of the accumulated entity are reflected in the relational database. For timing to reflect the persistence operations at the time of using Spring Data JPA, refer to *Reflection timing of persistence processing (1)* *Reflection timing of persistence processing (2)*

Note: About other life cycle management methods

The detach method, refresh method and clear method are available in `EntityManager` to manage the entity life cycle. However, when using Spring Data JPA, there is no mechanism to call these methods using the default function, hence only their roles are described below.

- detach method is used to set a "managed" entity to "detached" entity.
- refresh method is used to update the "managed" entity as per the state of relational database.
- clear method is used to delete the entity managed in PersistenceContext and the accumulated operations from the memory.

clear method can be called by setting the clearAutomatically attribute of `@Modifying` annotation of Spring Data JPA to `true`. For details, refer to *Operating the entities of Persistence Layer directly*.

Note: About operations of "new" and "detached" entities

The operations performed on "new" and "detached" entities are not reflected in the relational database unless persist method or merge method is called.

About Spring Data JPA

Spring Data JPA provides the library to create Repository using JPA.

If Spring Data JPA is used, it is possible to retrieve an entity that matches the specified conditions only by defining the
query method in the Repository interface; hence the amount of implementation for performing the entity operations can be reduced.
However, only static query which can be expressed using annotation, can be defined in query method; hence it is necessary to implement custom Repository class for the query such as dynamic query which cannot be expressed using annotation.

The basic flow at the time of accessing the database using Spring Data JPA is shown below.

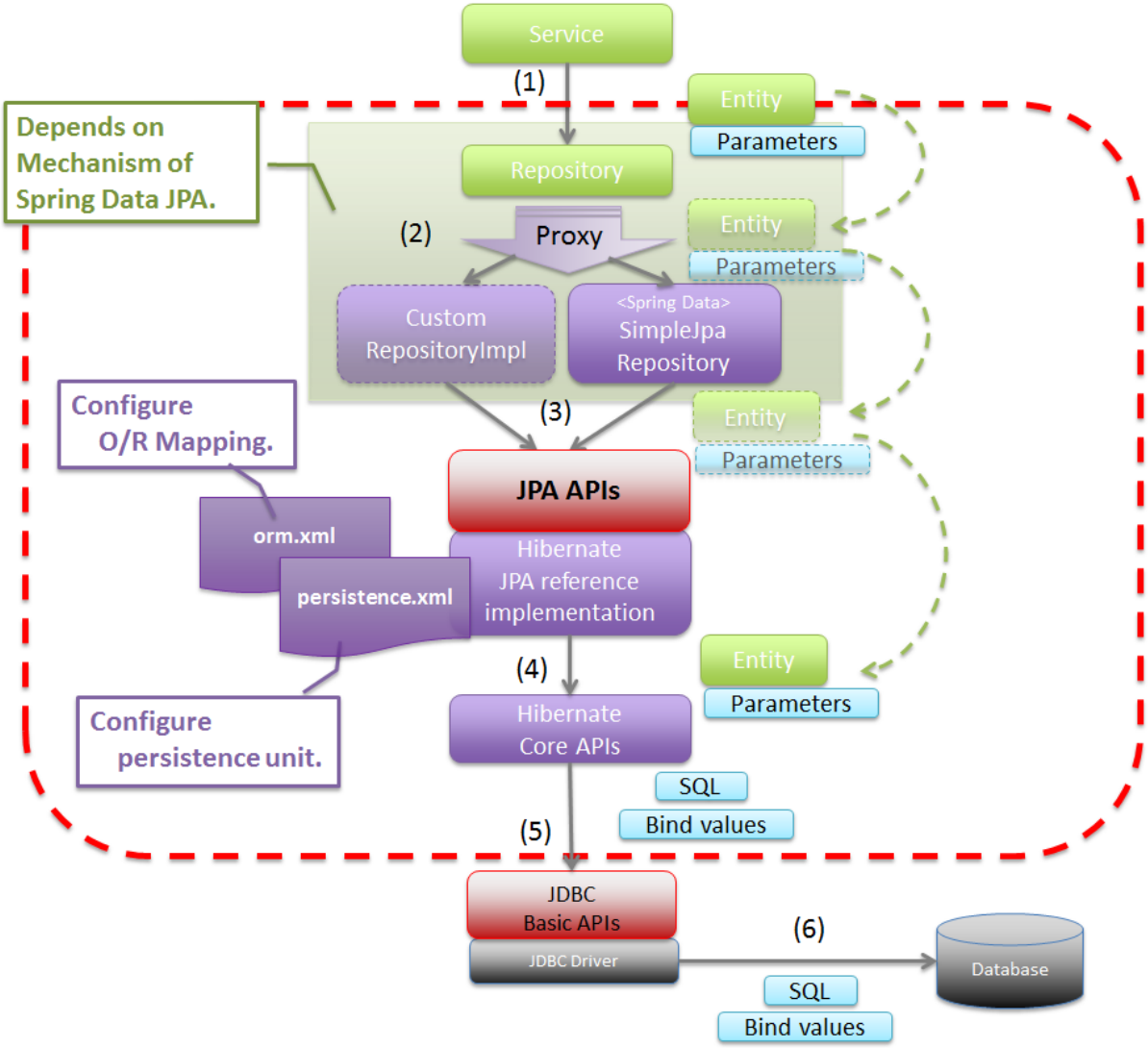


Figure.6.21 **Picture - Basic flow of Spring Data JPA**

Sr. No.	Description
(1)	Call the method of Repository interface from Service. Entity object, Entity ID etc. are passed as method calling parameters. In the above example, entity is passed, however a primitive value can also be passed.
	次のページに続く

Table. 6.13 – 前のページからの続き

Sr. No.	Description
(2)	Proxy class which dynamically implements Repository interface, delegates the process to `org.springframework.data.jpa.repository.support.SimpleJpaRepository` or custom Repository class. Parameters specified by Service are passed.
(3)	Repository implementation class calls JPA APIs. The parameters specified by Service and the parameters generated by implementation class of Repository are passed.
(4)	Hibernate JPA reference implementation calls the Hibernate core APIs. The parameters specified by implementation class of Repository and the parameters generated by Hibernate JPA reference implementation are passed.
(5)	Hibernate Core API generates SQL and bind values from the specified parameters and passes to JDBC driver. (API of java.sql.PreparedStatement is used for binding the actual values.)
(6)	JDBC driver executes SQL.

When creating the repository using Spring Data JPA, APIs of JPA need not be called directly; however, it is better to know which JPA method is being called by

methods of Repository interface of Spring Data JPA.

The JPA methods called by main methods of Repository interface of Spring Data JPA are shown below.

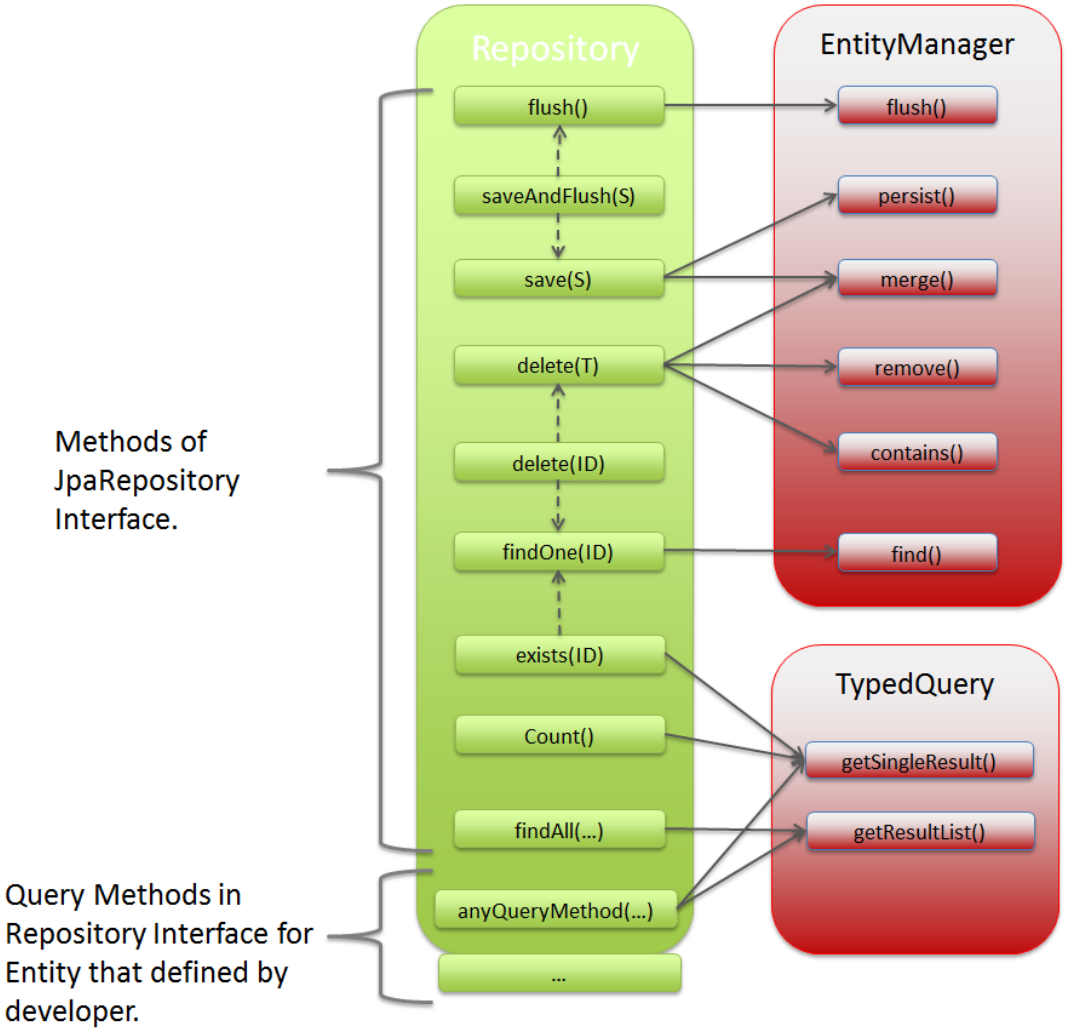


Figure.6.22 **Picture - API Mapping of Spring Data JPA and JPA**

6.3.2 How to use

pom.xml settings

When using JPA (Spring Data JPA) in infrastructure layer, add the following dependency to pom.xml

```
<!-- (1) -->
<dependency>
    <groupId>org.terasoluna.gfw</groupId>
    <artifactId>terasoluna-gfw-jpa-dependencies</artifactId>
    <type>pom</type>
</dependency>
```

Sr. No.	Description
(1)	`terasoluna-gfw-jpa-dependencies` where the libraries associated with JPA are defined should be added to dependency.

Note: In the above setting example, since it is assumed that the dependent library version is managed by the parent project terasoluna-gfw-parent, specifying the version in pom.xml is not necessary.

Application Settings

Datasource settings

Set connection information of the database to datasource.

For datasource settings, refer to *Datasource settings*.

EntityManager settings

Perform settings to use `EntityManager`.

- xxx-infra.xml

```
<!-- (1) -->
<bean id="jpaVendorAdapter"
    class="org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter">
    <!-- (2) -->
    <property name="showSql" value="false" />
    <!-- (3) -->
    <property name="database" value="POSTGRESQL" />
</bean>
```

```
<!-- (4) -->
<bean id="entityManagerFactory"
    class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean
↪">
    <!-- (5) -->
    <property name="packagesToScan" value="xxxxxx.yyyyyy.zzzzzz.domain.model"␣
↪/>
    <!-- (6) -->
    <property name="dataSource" ref="dataSource" />
    <!-- (7) -->
    <property name="jpaVendorAdapter" ref="jpaVendorAdapter" />
    <!-- (8) -->
    <property name="jpaPropertyMap">
        <util:map>
            <entry key="hibernate.hbm2ddl.auto" value="" />
            <entry key="hibernate.ejb.naming_strategy"
                value="org.hibernate.cfg.ImprovedNamingStrategy" />
            <entry key="hibernate.connection.charSet" value="UTF-8" />
            <entry key="hibernate.show_sql" value="false" />
            <entry key="hibernate.format_sql" value="false" />
            <entry key="hibernate.use_sql_comments" value="true" />
            <entry key="hibernate.jdbc.batch_size" value="30" />
            <entry key="hibernate.jdbc.fetch_size" value="100" />
        </util:map>
    </property>
</bean>
```

Sr. No.	Description
(1)	Specify the adapter class associated with JPA provider. Hibernate will be used as JPA provider; hence specify `org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter`.
(2)	Set the SQL output flag. In the example, “false: Do not output” has been specified.

次のページに続く

Table. 6.14 – 前のページからの続き

Sr. No.	Description
(3)	Set the value corresponding to RDBMS to be used. It is possible to specify the value defined in `org.springframework.orm.jpa.vendor.Database` enumerator type. In the example, "PostgreSQL" has been specified. **[The value should be changed according to the database used in the project]** If the database to be used changes with the environment, the value should be defined in properties file.
(4)	Specify FactoryBean class to create `javax.persistence.EntityManagerFactory` instance. Specify `org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean`.
(5)	Specify the package where entity classes are kept. The entity classes of the specified package can be managed using `javax.persistence.EntityManager`. **[The value should be changed to the relevant package name according to the project]**
(6)	Specify the datasource to be used for accessing persistence layer (DB).
(7)	Specify `JpaVendorAdapter` bean. Specify the bean set in (1).
(8)	Specify the settings to configure `EntityManager` of Hibernate. For details, refer to "Hibernate Reference Documentation" .

Tip: When using the Oracle database, ANSI standard SQL JOIN for combining tables, can be used by specifying the following settings in `jpaPropertyMap` mentioned in (8).

```
<bean id="entityManagerFactory"
    class="org.springframework.orm.jpa.
↪LocalContainerEntityManagerFactoryBean">
    <!-- omitted -->
    <property name="jpaPropertyMap">
        <util:map>
            <!-- omitted -->
            <entry key="hibernate.dialect"
                   value="org.hibernate.dialect.Oracle12cDialect" />  <!--␣
↪(9) -->
        </util:map>
    </property>
</bean>
```

Sr. No.	Description
(9)	Specify `org.hibernate.dialect.Oracle12cDialect` in `"hibernate.dialect"`. By specifying `Oracle12cDialect`, ANSI standard SQL JOIN clause for combining the tables can be used.

Perform the following settings when transaction manager (JTA) of the application server is to be used.

The difference with the case wherein JTA is not used, is explained below.

For other locations, same settings as the case wherein JTA is not used can be performed.

- xxx-infra.xml

```
<bean id="entityManagerFactory"
    class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean
↪">

    <!-- omitted -->

    <!-- (10) -->
    <property name="jtaDataSource" ref="dataSource" />

    <!-- omitted -->
```

```
    <property name="jpaPropertyMap">
        <util:map>

            <!-- omitted -->

            <!-- (11)  -->
            <entry key="hibernate.transaction.jta.platform"
                value="org.hibernate.service.jta.platform.internal.
↪WeblogicJtaPlatform" />

        </util:map>
    </property>
</bean>
```

Sr. No.	Description
(10)	Specify the datasource to be used for accessing persistence layer (DB). When using JTA, specify the DataSource defined in application server in "jtaDataSource" property and not in "dataSource" property. Refer to *Datasource settings* of common edition for the method to fetch DataSource defined in application server.
(11)	Add JTA platform specification in "jpaPropertyMap" property. The above example illustrates usage of Weblogic JTA. The configurable value (platform) is FQCN of org.hibernate.service.jta.platform.spi.JtaPlatform implementation class. The implementation class for main application servers is provided by Hibernate.

Note: When it is necessary to switch the transaction manager to be used as per the environment, then it is recommended that you define "entityManagerFactory" bean in xxx-env.xml instead of xxx-infra.xml.

An example wherein it is necessary to change the transaction manager as per environment can be: use of application server without JTA function such as Tomcat in case of local environment, and use of application server with JTA function such as Weblogic in case of production environment as well as various test environments.

PlatformTransactionManager settings

Perform the following settings when using local transaction.

- xxx-env.xml

```
<bean id="transactionManager"
    class="org.springframework.orm.jpa.JpaTransactionManager"> <!-- (1) -->
    <property name="entityManagerFactory" ref="entityManagerFactory" /> <!--␣
↪(2) -->
</bean>
```

Sr. No.	Description
(1)	Specify `org.springframework.orm.jpa.JpaTransactionManager`. This class controls transaction by calling APIs of JPA.
(2)	Specify Factory of `EntityManager` to be used in the transaction.

Perform the following settings when transaction manager (JTA) of the application server is to be used.

- xxx-env.xml

```
<tx:jta-transaction-manager /> <!-- (1) -->
```

Sr. No.	Description
(1)	The most appropriate `org.springframework.transaction.jta.JtaTransactionManager` is defined as bean with id as "transactionManager", in the application server on which the application has been deployed. This class controls the transaction by calling JTA APIs.

persistence.xml settings

When using `LocalContainerEntityManagerFactoryBean`, there are no mandatory settings to be performed in `persistence.xml`.

Todo

TBD

Currently, there are no mandatory settings to be performed in `persistence.xml`; however such need may arise in future.

When using EntityManagerFactory in application server of Java EE, it may be necessary to perform few settings in `persistence.xml` ; hence we are planning to provide maintenance for such settings in future.

Settings for validating Spring Data JPA

- xxx-infra.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:jpa="http://www.springframework.org/schema/data/jpa"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation=".....
    http://www.springframework.org/schema/data/jpa
    http://www.springframework.org/schema/data/jpa/spring-jpa.xsd"> <!-- (1) -
↪->

    <!-- ... -->

</beans>
```

```
<jpa:repositories base-package="xxxxxx.yyyyyy.zzzzzz.domain.repository" /> <!-
↪- (2) -->
```

Sr. No.	Description
(1)	Import schema definition for Spring Data JPA configuration and assign ("`jpa`") as namespace.
(2)	Specify base package wherein Repository interface and custom Repository class are stored. Interface inheriting `org.springframework.data.repository.Repository` and interface with `org.springframework.data.repository.RepositoryDefinition` annotation are automatically defined as a bean of Repository class of Spring Data JPA.

•Attributes of <jpa:repositories> element

entity-manager-factory-ref, transaction-manager-ref, named-queries-location, query-lookup-strategy, factory-class and repository-impl-postfix are present as attributes.

Sr. No.	Element	Description
1.	entity-manager-factory-ref	Specify Factory for generating `EntityManager` to be used in Repository. If multiple Factories of `EntityManager` are to be created, then it is necessary to specify the bean to be used.
2.	transaction-manager-ref	Specify `PlatformTransactionManager` to be used when the methods of Repository are called. The bean registered with `"transactionManager"` bean name is used by default. It needs to be specified when the bean name of `PlatformTransactionManager` to be used is not `"transactionManager"`.
3.	named-queries-location	Specify the location of Spring Data JPA properties file wherein Named Query is specified. “classpath:META-INF/jpa-named-queries.properties” is used by default.
4.	query-lookup-strategy	Specify the method to Lookup the query to be executed when query method is called. By default, it is `"CREATE_IF_NOT_FOUND"`. For details, refer to Spring Data Commons - Reference Documentation の “Query lookup strategies”. Use the default settings if there is no specific reason.
5.	factory-class	Specify Factory for generating class to implement the process when the method of Repository interface is called. `org.springframework.data.jpa.repository.support.JpaRepositoryFac` is used by default. Specify the Factory created for changing default implementation of Spring Data JPA or for adding a new method. For how to add a new method, refer to *Adding the custom methods to all Repository interfaces in batch*.

次のページに続く

Table. 6.16 – 前のページからの続き

Sr. No.	Element	Description
6.	repository-impl-postfix	Specify suffix indicating that it is an implementation class of custom Repository. By default, it is `"Impl"`. For example: when Repository interface name is `OrderRepository`, `OrderRepositoryImpl` will be the implementation class of custom Repository. Use the default settings if there is no specific reason. For custom Repository, refer to "*Adding individual custom method to entity specific Repository interface*".

Settings for using JPA annotations

To inject `javax.persistence.EntityManagerFactory` and `javax.persistence.EntityManager` using the annotations (`javax.persistence.PersistenceContext` and `javax.persistence.PersistenceUnit`) provided by JPA, `org.springframework.orm.jpa.support.PersistenceAnnotationBeanPostProcessor` should be defined as bean.

When `<jpa:repositories>` element is specified, bean is defined by default; hence no need to define it separately.

Settings for converting JPA exception to DataAccessException

To convert JPA exception to `DataAccessException` of Spring Framework, `org.springframework.dao.annotation.PersistenceExceptionTranslationPostProcessor` should be defined as bean.

When `<jpa:repositories>` element is specified, bean is defined by default; hence no need to define it separately.

OpenEntityManagerInViewInterceptor settings

To perform Lazy Fetching of Entity in application layer such as Controller and JSP etc., the lifetime of `EntityManager` should be extended till application layer using `org.springframework.orm.jpa.support.OpenEntityManagerInViewInterceptor`.

When `OpenEntityManagerInViewInterceptor` is not to be used, the lifetime of `EntityManager` becomes same as that of transaction; hence it is necessary to either fetch the data required in application layer as a process of Service class or to use Eager Fetch instead of Lazy Fetch.

Considering the following perspectives, it is recommended that you use Lazy Fetch as fetch method and `OpenEntityManagerInViewInterceptor`.

- Fetching as a process of Service class leads to insignificant implementation such as calling only getter method or accessing the collection fetched by calling getter method.
- When Eager Fetch is used, it is likely that the data which is not used in application layer is also fetched impacting the performance.

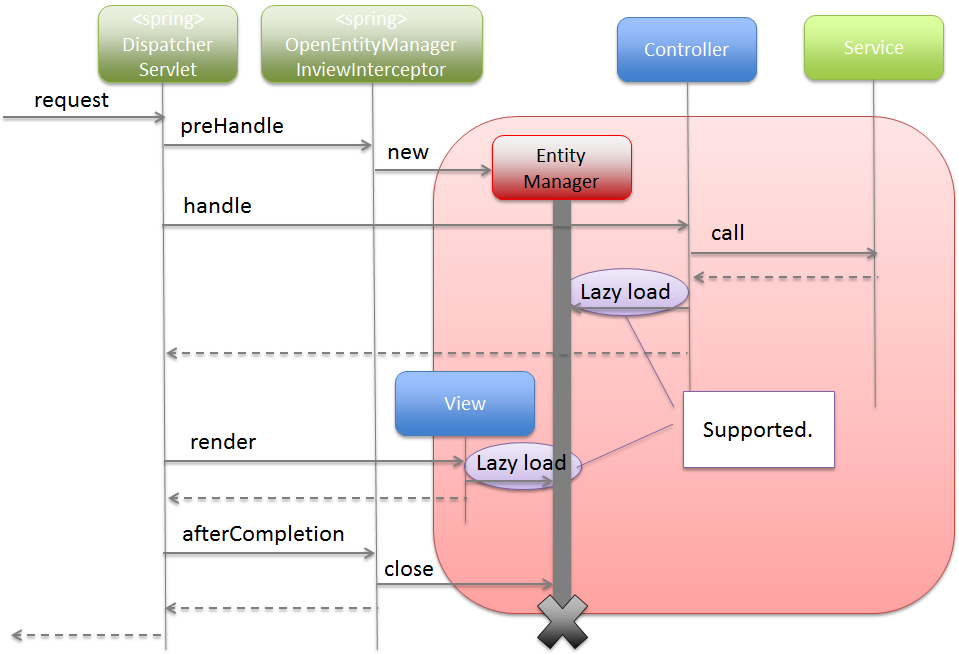


Figure.6.23 **Picture - Lifetime of EntityManager on OpenEntityManagerInViewInterceptor**

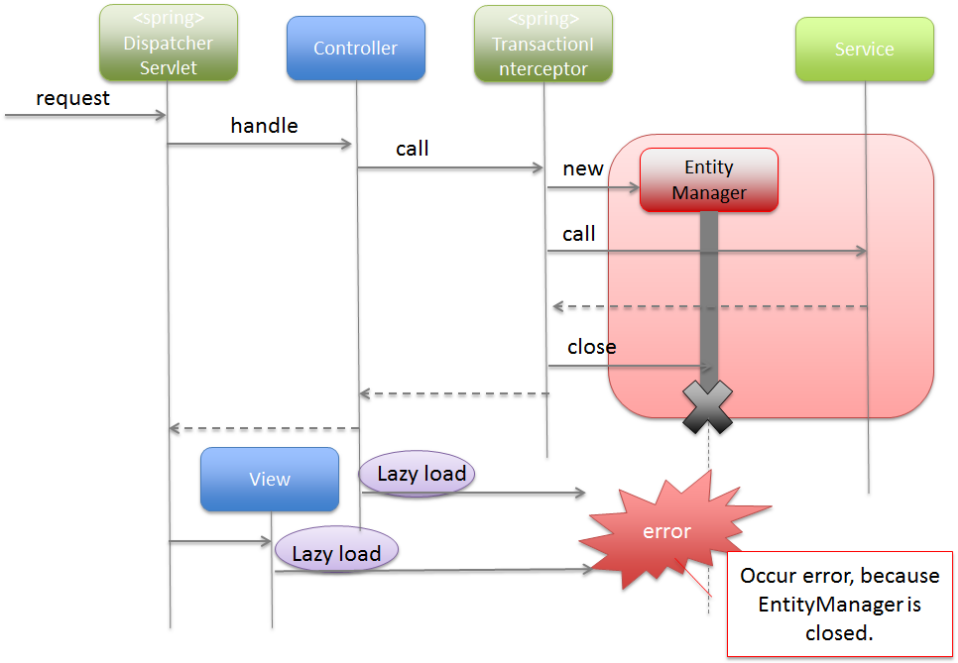


Figure.6.24 **Picture - Default Life time of EntityManager**

See the example of `OpenEntityManagerInViewInterceptor` settings below.

- spring-mvc.xml

```
<mvc:interceptors>
    <mvc:interceptor>
        <mvc:mapping path="/**" /> <!-- (1) -->
        <mvc:exclude-mapping path="/resources/**" /> <!-- (1) -->
        <mvc:exclude-mapping path="/**/*.html" /> <!-- (1) -->
        <!-- (2) -->
        <bean
            class="org.springframework.orm.jpa.support.
↪OpenEntityManagerInViewInterceptor" />
    </mvc:interceptor>
</mvc:interceptors>
```

Sr. No.	Description
(1)	Specify the path for which interceptor is to be applied and path for which interceptor is not to be applied. In this example, interceptor is being applied for paths other than paths of resource files (js, css, image etc.) and static web page (HTML).
(2)	Specify `org.springframework.orm.jpa.support.OpenEntityManagerInViewInterceptor`.

Note: Interceptor not to be applied to the path of static resources

It is recommended that interceptor not be applied to the path of static resources (js, css, image, html etc.), as there is no data access in such cases. Application of interceptor to the path of static resources leads to execution of unnecessary processes (such as instance generation and close process).

When Lazy Fetch is required in Servlet Filter, it is necessary to extend the lifetime of EntityManager till the Servlet Filter layer using `org.springframework.orm.jpa.support.OpenEntityManagerInViewFilter`.

For example, this case is applicable when

`org.springframework.security.core.userdetails.UserDetailsService` of SpringSecurity is inherited and if Entity object is accessed in the inherited logic.

However, if Lazy Fetch is not required, there is no need to extend the lifetime of `EntityManager` till the Servlet Filter layer.

Note: About Lazy Fetch in Servlet Filter layer

It is recommended that you design and implement such that Lazy Fetch does not occur in Servlet Filter layer. If `OpenEntityManagerInViewInterceptor` is used, it is possible to specify the applicable and non-applicable URL patterns; thus the path for which lifetime of "EntityManager" is to be extended till application layer can also be easily specified. For the data access required in Servlet Filter, the data should either be fetched in advance in Service class or should be loaded in advance using Eager Fetch; thereby, avoiding the occurrence of Lazy Fetch.

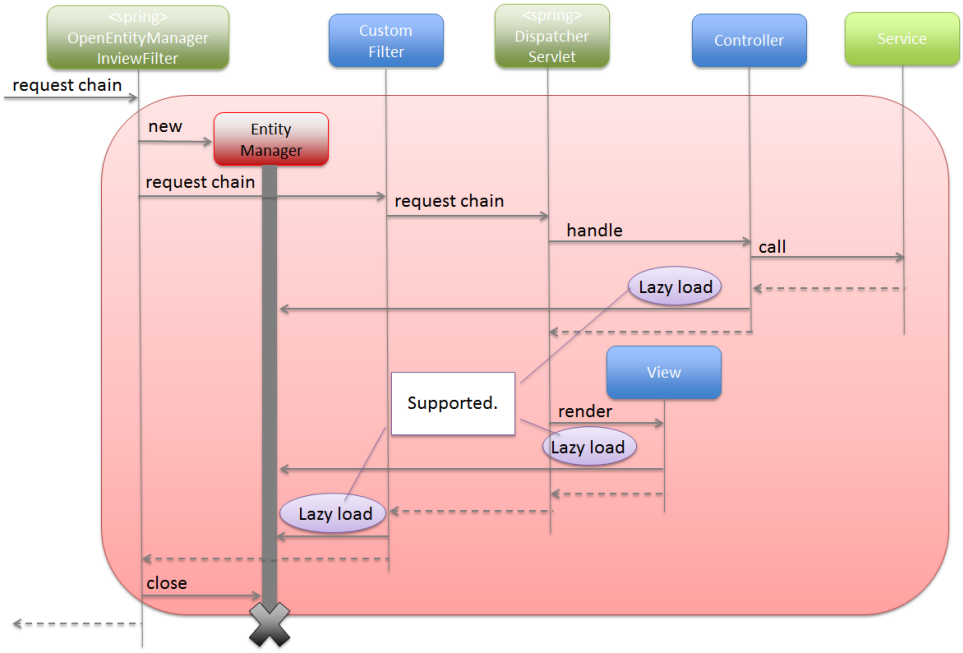


Figure.6.25 **Picture - Lifetime of EntityManager on OpenEntityManagerInViewFilter**

See the example of `OpenEntityManagerInViewFilter` settings below.

- web.xml

```
<!-- (1) -->
<filter>
    <filter-name>Spring OpenEntityManagerInViewFilter</filter-name>
    <filter-class>org.springframework.orm.jpa.support.
↪OpenEntityManagerInViewFilter</filter-class>
</filter>
<!-- (2) -->
<filter-mapping>
    <filter-name>Spring OpenEntityManagerInViewFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

Sr. No.	Description
(1)	Specify `org.springframework.orm.jpa.support.OpenEntityManagerInViewFilter`. This Servlet Filter **needs to be defined before the Servlet Filter in which Lazy Fetch occurs.**
(2)	Specify the pattern of the URL for which filter is to be applied. It is recommended that you apply the filter only to the required path; however, if the settings are complicated, you can also specify "/*" (All Requests).

Note: If "/*" (All Requests) are specified in the pattern of the URL for which `OpenEntityManagerInViewFilter` is to be applied, `OpenEntityManagerInViewInterceptor` settings are not required.

Creating Repository interface

Spring Data provides the following 3 methods to create entity specific Repository interface.

Sr. No.	How to create	Description
1.	*Inheriting the interface of Spring Data*	Create entity specific Repository interface by inheriting from the interface of Spring Data. **If there is no specific reason, then it is recommended that you create the entity specific Repository interface using this method.**
2.	*Inheriting a common project specific interface in which only the required methods are defined*	Out of all the methods of Repository interface of Spring Data, create a common project specific interface wherein only the required methods are specified. Inherit the common interface to create entity specific Repository interface.
3.	*Not inheriting the interface*	Create entity specific Repository interface without inheriting the interface of Spring Data or common project specific common interface.

Inheriting the interface of Spring Data

The method to create entity specific Repository interface by inheriting from the interface of Spring Data is explained below.

Interfaces that can be inherited are as follows:

Sr. No.	Interface	Description
1.	org.springframework.data.repository CrudRepository	Repository interface for generic CRUD operations.
2.	org.springframework.data.repository PagingAndSortingRepository	Repository interface wherein Pagination function and Sort function are added to findAll method of `CrudRepository`.
3.	org.springframework.data.jpa.repository JpaRepository	Repository interface that provides JPA specifications dependent methods. `PagingAndSortingRepository` is inherited; hence methods of `PagingAndSortingRepository` and `CrudRepository` can also be used. **If there is no specific reason, it is recommended that you create entity specific Repository interface by inheriting this interface.**

Note: About default implementation of Repository interface of Spring Data

The methods defined in the above interface are implemented using `org.springframework.data.jpa.repository.support.SimpleJpaRepository` of Spring Data JPA.

The example is given below.

```
public interface OrderRepository extends JpaRepository<Order, Integer> { //␣
↪(1)

}
```

Sr. No.	Description
(1)	Inherit `JpaRepository` and specify entity type in generic type `<T>` and entity ID type in generic type `<ID extends Serializable>`. In the above example, `Order` type is specified in entity and `Integer` type in entity ID.

If entity specific Repository interface is created by inheriting `JpaRepository`, then the following methods can be implemented.

Sr. No.	Method	Description
1.	<S extends T> S save(S entity)	Method to accumulate persistence operations (INSERT/UPDATE) for the specified entity in `javax.persistence.EntityManager`. If the value is not set in ID property (property with `@javax.persistence.Id` annotation or `@javax.persistence.EmbeddedId` annotation), `persist` method of `EntityManager` is called and when the value is set, `merge` method is called. When merge method is called, please note that the returned Entity object is different from the Entity which is passed as an argument.
2.	<S extends T> List<S> save(Iterable<S> entities)	Method to accumulate persistence operations for multiple specified entities in `EntityManager`. The method is implemented by calling `<S extends T> S save(S entity)` method repeatedly.
3.	T saveAndFlush(T entity)	Once the persistence operations for the specified entity are accumulated in `EntityManager`, this method reflects the accumulated persistence operations (INSERT/UPDATE/DELETE) in persistence layer (DB).

次のページに続く

Table. 6.17 – 前のページからの続き

Sr. No.	Method	Description
4.	void flush()	Method to execute persistence operations (INSERT/UPDATE/DELETE) for the entity accumulated in `EntityManager` in persistence layer (DB).
5.	void delete(ID id)	Method to accumulate delete operation for the entity of specified ID, in `EntityManager`. This method calls `T findOne(ID)` method and converts the entity object to "managed" state under `EntityManager` and then deletes that object. If entity is not present when `T findOne(ID)` method is called, `org.springframework.dao.EmptyResultDataAccessExcept` occurs.
6.	void delete(T entity)	Method to accumulate delete operation for the specified entity, in `EntityManager`.
7.	void delete(Iterable<? extends T> entities)	Method to accumulate delete operations for the multiple specified entities, in `EntityManager`. This method is implemented by calling `void delete(T entity)` method repeatedly. In order to delete large number of entities, it is desirable to use `void deleteInBatch(Iterable<T> entities)` method.

次のページに続く

Table. 6.17 – 前のページからの続き

Sr. No.	Method	Description
8.	void deleteAll()	Method to accumulate delete operations for all entities, in `EntityManager`. This method is implemented by repeatedly calling `void delete(T entity)` method for the entities fetched by `List<T> findAll()` method. In order to delete large number of entities, `void deleteAllInBatch()` method should be used. This method loads all the entities to be deleted in the memory, thus causing memory exhaustion.
9.	void deleteInBatch(Iterable<T> entities)	Method to directly delete multiple specified entities from persistence layer (DB). When entities are deleted using this method, entities which are managed (cached) under `EntityManager` are not deleted. Hence if `T findOne(ID id)` method is called after the entity is deleted using this method, note that the entities managed in `EntityManager` will be returned. For a deleted entity, in subsequent processing, if there is a possibility of calling methods like `T findOne(ID id)` (which return the Entity object managed (cached) under `EntityManager`), the corresponding entity should be deleted using `void delete(Iterable<? extends T> entities)` method.
10.	void deleteAllInBatch()	Method to directly delete all entities from persistence layer (DB). Similar to “void deleteInBatch(Iterable<T> entities)” method, note that the entity managed (cached) under “EntityManager” is not deleted.

次のページに続く

Table. 6.17 – 前のページからの続き

Sr. No.	Method	Description
11.	T findOne(ID id)	Method to fetch the entity of specified ID from persistence layer (DB). The entity fetched from persistence layer is managed (cached) by `EntityManager`; hence 2nd time onwards, persistence layer is not accessed and the cached entity is returned.
12.	List<T> findAll()	Method to fetch all entities from persistence layer (DB). The entity fetched from persistence layer is managed (cached) by `EntityManager`.
13.	Iterable<T> findAll(Iterable<ID> ids)	Method to fetch entities of multiple specified IDs, from persistence layer (DB). The entities fetched from persistence layer are managed (cached) by `EntityManager`. The specified IDs are searched using IN clause; hence be careful while using a DB where only limited number of values can be specified in IN clause such as Oracle.
14.	List<T> findAll(Sort sort)	Method to fetch all entities from persistence layer (DB) in the specified sort order. The entity fetched from persistence layer is managed (cached) using `EntityManager`.
15.	Page<T> findAll(Pageable pageable)	Method to fetch the entity matching the specified page (sort order, page number, number of records to be displayed on page) from persistence layer (DB). The entity fetched from persistence layer is managed (cached) by `EntityManager`.

次のページに続く

Table. 6.17 – 前のページからの続き

Sr. No.	Method	Description
16.	boolean exists(ID id)	Method to check whether the entity of specified ID exists.
17.	long count()	Returns the number of entities available.

Warning: Behavior when using optimistic locking (@javax.persistence.Version) of JPA

When the entity is updated or deleted at the time of using optimistic locking (`@Version`) of JPA, `org.springframework.dao.OptimisticLockingFailureException` occurs. `OptimisticLockingFailureException` may occur in the following methods.

- <S extends T> S save(S entity)
- <S extends T> List<S> save(Iterable<S> entities)
- T saveAndFlush(T entity)
- void delete(ID id)
- void delete(T entity)
- void delete(Iterable<? extends T> entities)
- void deleteAll()
- void flush()

For details on optimistic locking of JPA, refer to *Exclusive Control*.

Note: Timing to reflect persistence operations (1)

For the entity managed under `EntityManager`, accumulated persistence operations are executed just before committing a transaction and reflected in persistence layer (DB). Therefore, in order to handle errors such as unique constraint violation in transaction management (Service processing), it is necessary to call "saveAndFlush" method or "flush" method and execute persistence operations for the Entity accumulated in "EntityManager" forcibly. If only an error is to be notified to the client, it is OK to perform exception handling in Controller and set an appropriate message.

`saveAndFlush` and `flush` are JPA dependent methods, hence do not use these methods if there is no specific purpose.

- Normal flow

- flush flow

Note: Timing to reflect persistence operations (2)

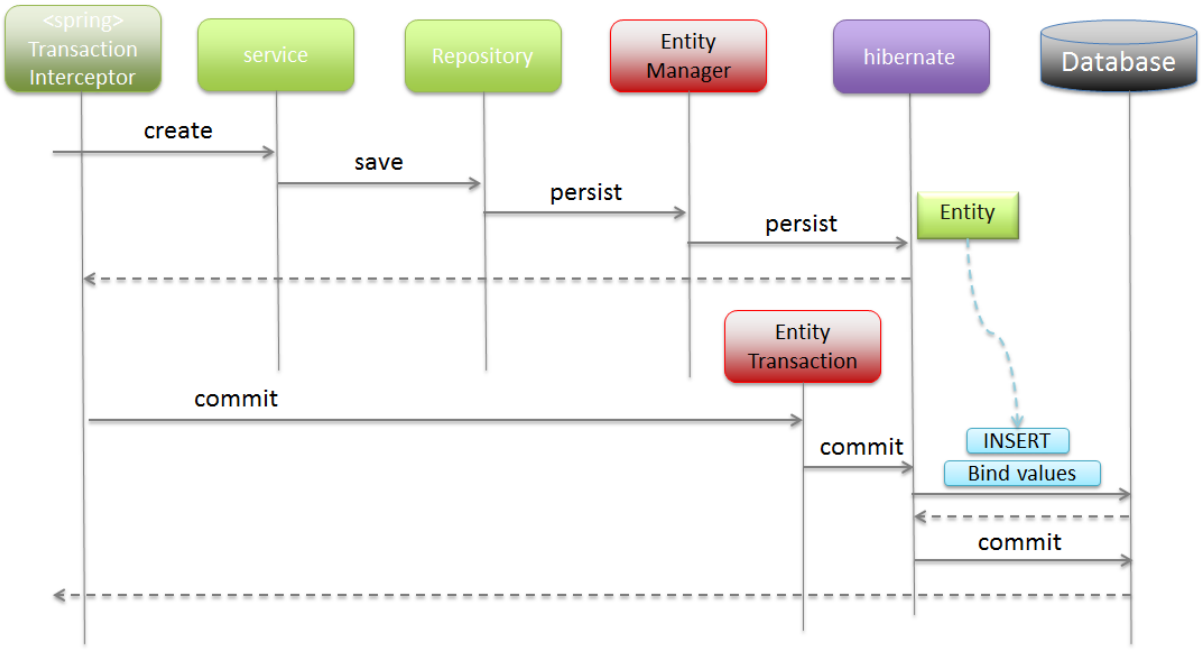


Figure.6.26 **Picture - Normal sequence of persistence processing**

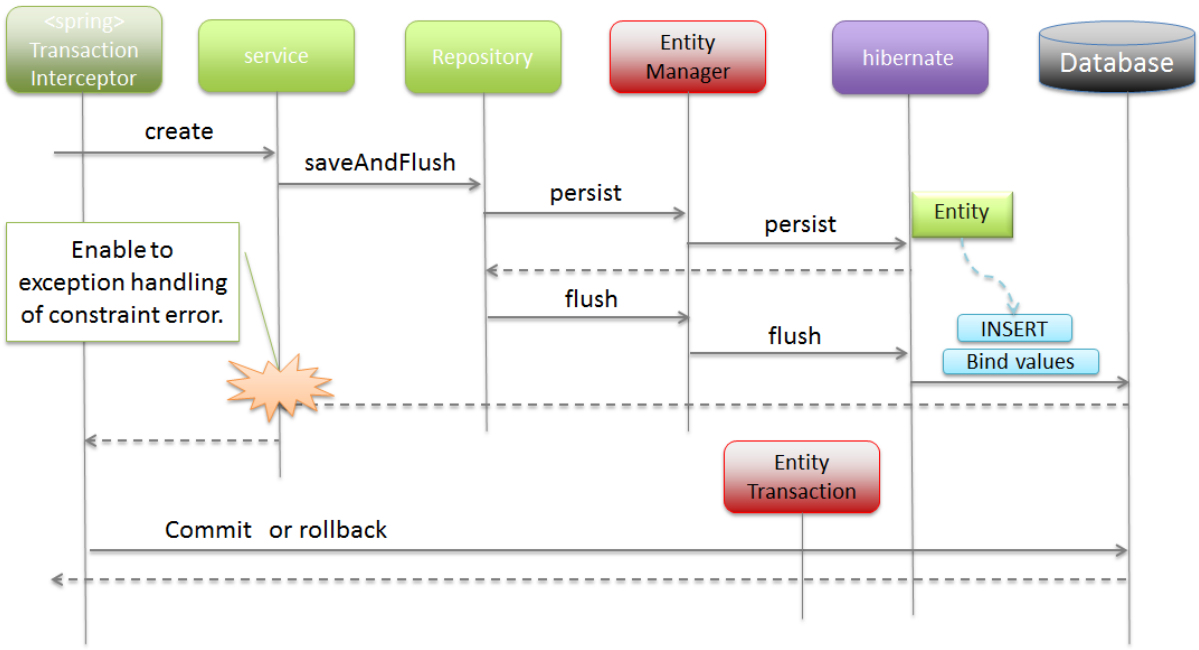


Figure.6.27 **Picture - Sequence of persistence processing when flush method is used**

When the following method is called, in order to avoid inconsistency between the data managed in `EntityManager` and persistence layer (DB), the persistence operations of the entity accumulated in `EntityManager` are reflected in the persistence layer (DB) before the main process is carried out.

- `List<T> findAll` method
- `boolean exists(ID id)`
- `long count()`

In case of above methods, query is executed directly in the persistence layer (DB); hence inconsistency may occur unless the operations are reflected in the persistence layer (DB) before the main process is carried out. Calling of query methods described later also triggers the reflection of persistence operations for the entity accumulated in `EntityManager`, in the persistence layer (DB).

- Flow at the time of issuing queries

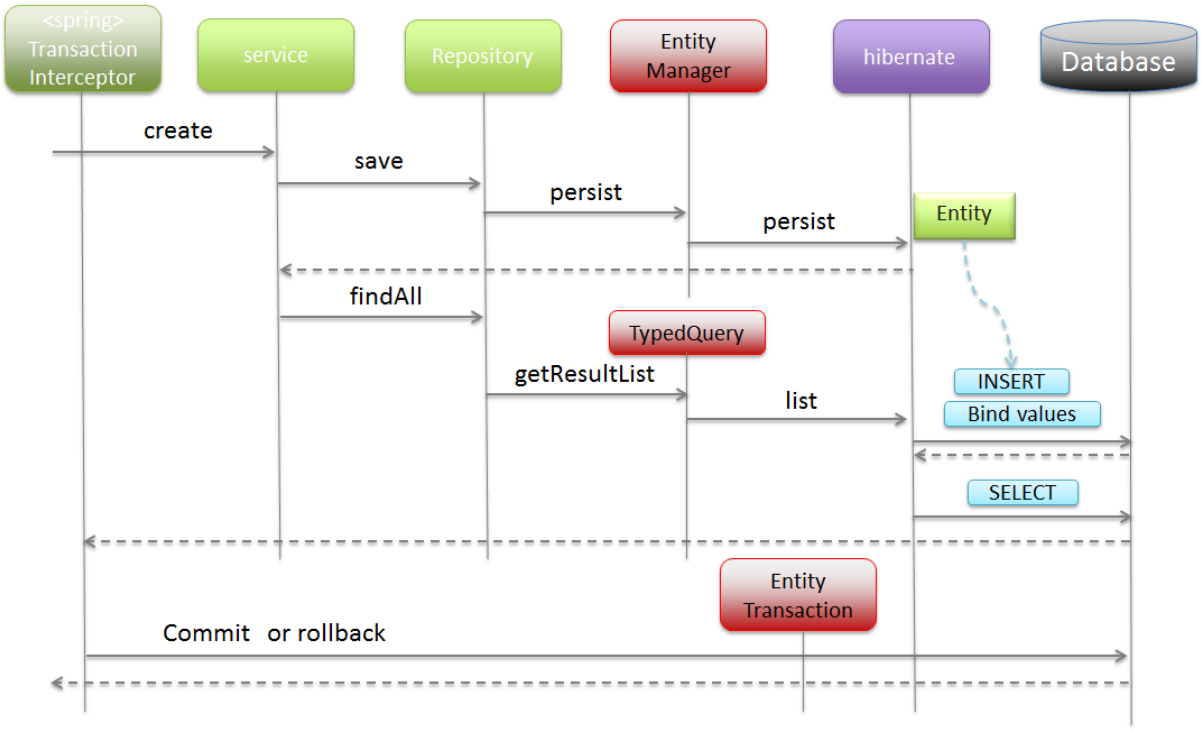


Figure.6.28 **Picture - Sequence of persistence processing when query method is used**

Inheriting a common project specific interface in which only the required methods are defined

Amongst the methods defined in interface of Spring Data, this section defines the method to create entity specific Repository interface by creating and inheriting a common project specific interface in which only the required methods are defined.

The signature of methods should match with the methods of Repository interface of Spring Data;

Note: Assumed cases

Amongst the methods of Repository interface of Spring Data, there are methods which are not used or which are not desirable to be used in the actual application. In order to remove such methods from Repository interface, refer below. The methods defined in interface are implemented using `org.springframework.data.jpa.repository.support.SimpleJpaRepository` of Spring Data JPA.

See the example below.

```
@NoRepositoryBean // (1)
public interface MyProjectRepository<T, ID extends Serializable> extends
        Repository<T, ID> { // (2)

    T findOne(ID id); // (3)

    T save(T entity); // (3)

    // ...

}
```

```
public interface OrderRepository extends MyProjectRepository<Order, Integer>
 ↪{ // (4)

}
```

Sr. No.	Description
(1)	Specify `@NoRepositoryBean` annotation to prevent the instantiation of `Repository` interfaces by Spring Data.
(2)	Define a common interface for the project by inheriting `org.springframework.data.repository.Repository`. Use generic type since it is a not entity specific interface.
(3)	Select and define the required methods from the methods of Repository interface of Spring Data.
(4)	Inherit this common interface and specify the type of entity in generic type `<T>` and type of entity ID in generic type `<ID extends Serializable>`. In this example, `Order` type is specified in entity and `Integer` type in entity ID.

Not inheriting the interface

This section explains how to create entity specific Repository interface without inheriting any interface of Spring Data or common interface.

Specify `@org.springframework.data.repository.RepositoryDefinition` annotation as class annotation and specify entity type in domainClass attribute and entity ID type in idClass attribute.

The methods which have the same signature as methods defined in Repository interface of Spring Data need not be implemented.

Note: Assumed cases

Repository can be created in this way when common entity operations are not required. The methods having same signature as methods defined in Repository interface of Spring Data are implemented using `org.springframework.data.jpa.repository.support.SimpleJpaRepository` provided by Spring Data JPA.

See the example below.

```
@RepositoryDefinition(domainClass = Order.class, idClass = Integer.class) //
↪(1)
public interface OrderRepository { //(2)

    Order findOne(Integer id); // (3)

    Order save(Order entity); // (3)

    // ...
}
```

Sr. No.	Description
(1)	Specify `@RepositoryDefinition` annotation. In the example, `Order` type is specified in domainClass attribute (entity type) and `Integer` type in idClass attribute (entity ID type).
(2)	There is no need to inherit the interface (`org.springframework.data.repository.Repository`) of Spring Data.
(3)	Define the methods required for each entity.

Adding query method

It is difficult to develop the actual application using only the Spring Data interface which is used for performing generic CRUD operations.

Therefore Spring Data provides a mechanism to add "query methods" for performing any persistence operations (SELECT/UPDATE/DELETE) for the entity specific Repository interface.

In the added query method, entity operations are performed using query language (JPQL or Native SQL).

Note: What is JPQL

JPQL is an abbreviation of "Java Persistence Query Language" and is the query language to perform entity operations (SELECT/UPDATE/DELETE) corresponding to the records of persistence layer (DB). The syntax is similar to SQL; however, JPQL operates the entities mapped to the records of persistence layer instead of operating these records directly. The entity operations are reflected to persistence layer (DB) using JPA provider (Hibernate).

For details on JPQL, refer to JSR 338: Java Persistence API, Version 2.1 Specification (PDF) "Chapter 4

Query Language".

Defining query method

Query method is defined as a method of entity specific Repository interface.

```
public interface OrderRepository extends JpaRepository<Order, Integer> {
    List<Order> findByStatusCode(String statusCode);
}
```

Specifying query to be executed

Query to be executed should be specified at the time of calling query method.

The method of specifying the query is as below. For details, refer to *Specifying a query while calling a query method*.

Sr. No.	Method to specify a query	Description
1.	*@Query annotation* (Spring Data functionality)	In the method to be added to entity specific Repository interface, specify `@org.springframework.data.jpa.repository.Query` annotation and the query to be executed. **When there is no specific reason, it is recommended that you specify the query using this method.**
2.	*Method name based on naming conventions* (Spring Data functionality)	Specify the query to be executed by assigning a method name as per Spring Data naming conventions. Query (JPQL) is generated from the method name using Spring Data JPA functionality. Only a SELECT clause of JPQL can be generated. **For a simple query having few conditions, this method can be used instead of using @Query annotation.** However, for a complex query with many conditions, a simple method name indicating behavior should be used and Query should be specified using `@Query` annotation.
3.	*Named query of properties file* (Spring Data functionality)	Specify the query in a properties file. **The location of method definition (entity specific Repository interface) and location wherein the query is specified (properties file) are separated; hence this way of specifying the query is not recommended.** **However, when using Native SQL as query, check whether it is necessary to define the database dependent SQL in the properties file.** In case of applications for which any database can be selected or when the database changes (or is likely to be changed) depending on execution environment, then it is necessary to specify the Query using this method and manage the Properties file as environment dependent material.

Note: Using multiple query specification methods

Particularly, there is no restriction on using multiple query specification methods. Query specification

methods and restriction on their concurrent usage should be determined in accordance with the project.

Note: Query Lookup methods

The operations would be as follows since the Spring Data default setting is `CREATE_IF_NOT_FOUND`.

1. Look for the query specified in `@Query` annotation.
2. Look for the corresponding query from Named query.
3. Create a query (JPQL) from method name and use it.
4. An error occurs when query (JPQL) cannot be created from method name.

For details on Query Lookup methods, refer to Spring Data Commons - Reference Documentation "Defining query methods" - "Query lookup strategies".

Fetching entity lock

To fetch entity lock, add `@org.springframework.data.jpa.repository.Lock` annotation to query method and specify the lock mode.

For details, refer to *Exclusive Control*.

```
@Query(value = "SELECT o FROM Order o WHERE o.status.code = :statusCode ORDER␣
↪BY o.id DESC")
@Lock(LockModeType.PESSIMISTIC_WRITE) // (1)
List<Order> findByStatusCode(@Param("statusCode") String statusCode);
```

```
-- (2) statusCode='accepted'
SELECT
        order0_.id AS id1_5_
        ,order0_.status_code AS status2_5_
    FROM
        t_order order0_
    WHERE
        order0_.status_code = 'accepted'
    ORDER BY
        order0_.id DESC
    FOR UPDATE
```

Sr. No.	Description
(1)	Specify the lock mode in value attribute of `@Lock` annotation. For the details on lock mode that can be specified, refer to Java Platform, Enterprise Edition API Specification.
(2)	Native SQL converted from JPQL.(DB to be used is PostgreSQL) In the example, `LockModeType.PESSIMISTIC_WRITE` has been specified; hence "FOR UPDATE" clause is added to SQL.

Operating the entities of Persistence Layer directly

It is recommended to perform update and delete operations on entity objects managed in `EntityManager`.
However, when entities need to be updated or deleted in a batch, check whether the entities of persistence layer (DB) are operated using query method.

Note: Reducing the causes of performance degradation

Operating the entities of persistence layer directly reduces the frequency of SQLs that would be required to be executed for operating these entities. Therefore, in case of applications that demand high performance, the causes of performance degradation can be reduced by operating the entities in batch using this method. Such SQLs are as follows:

- SQL for loading all entity objects in `EntityManager`. Need not be executed.
- SQL for updating and deleting entity. This SQL was earlier required to be executed n times, but now it is sufficient to execute it only once.

Note: Standards for deciding whether to operate entities of persistence layer directly

When operating the entities of persistence layer directly, since there are certain points to be careful about from functionality point of view, **in case of applications which do not demand high performance, it is recommended that the batch operations must also be performed through the entity objects managed in EntityManager.** For the points to be careful, refer to the example below.

The example of directly operating the entities of persistence layer using query method is shown below.

```
@Modifying // (1)
@Query("UPDATE OrderItem oi SET oi.logicalDelete = true WHERE oi.id.orderId =␣
↪:orderId ") // (2)
int updateToLogicalDelete(@Param("orderId") Integer orderId); // (3)
```

Sr. No.	Description
(1)	Specify `@org.springframework.data.jpa.repository.Modifying` annotation indicating that the method is UPDATE query method. If not specified, error will occur at the time of execution.
(2)	Specify UPDATE or DELETE query.
(3)	If update count or delete count is required, specify `int` or `java.lang.Integer` as return value and if count is not required, specify `void`.

Warning: Consistency with entities managed in EntityManager

When entities of persistence layer are operated directly using query method, there is no change in the entities managed in EntityManager as per the default behavior of Spring Data JPA. Therefore, it should be noted that the entity object fetched immediately after calling `JpaRepository#findOne(ID)` method would be in a state prior to the state of operating the entities.

This behavior can be avoided by setting the clearAutomatically attribute of `@Modifying` annotation to `true`. When clearAutomatically attribute is set to `true`, `clear()` method of `EntityManager` is called after operating the entities of persistence layer directly, and the entity objects managed in `EntityManager` and the accumulated persistence operations are deleted from `EntityManager`. Therefore, if `JpaRepository#findOne(ID)` method is called immediately, the mechanism is such that the latest entity would be fetched from the persistence layer and `EntityManager` status would be synchronized with the persistence layer.

Warning: Points to be noted while using @Modifying(clearAutomatically = true)

By using `@Modifying(clearAutomatically = true)`, it should be noted that the accumulated persistence operations (INSERT/UPDATE/DELETE) are also deleted from `EntityManager`. Bugs may occur as the required persistence operations may not be reflected in the persistence layer.

In order to avoid this problem, `JpaRepository#saveAndFlush(T entity)` or `JpaRepository#flush()` method should be called and the accumulated persistence operations should be reflected in the persistence layer before directly operating the entities of persistence layer.

Setting QueryHints

When it is necessary to set a hint in query, add `@org.springframework.data.jpa.repository.QueryHints` annotation to query method and specify QueryHint (`@javax.persistence.QueryHint`) in value attribute.

```
@Query(value = "SELECT o FROM Order o WHERE o.status.code = :statusCode ORDER␣
↪BY o.id DESC")
@Lock(LockModeType.PESSIMISTIC_WRITE)
@QueryHints(value = { @QueryHint(name = "javax.persistence.lock.timeout",␣
↪value = "0") }) // (1)
List<Order> findByStatusCode(@Param("statusCode") String statusCode);
```

Sr. No.	Description
(1)	Specify hint name in name attribute of `@QueryHint` annotation and hint value in value attribute. In addition to the hint stipulated in JPA specifications, provider specific hint can be specified. In the above example, lock timeout is set to `0` (DB used is Oracle). "FOR UPDATE NOWAIT" clause is added to SQL.

Note: QueryHints that can be specified in Hibernate

QueryHints stipulated in JPA specifications are as follows: For details, refer to JSR 338: Java Persistence API, Version 2.1 Specification (PDF).

- `javax.persistence.query.timeout`
- `javax.persistence.lock.timeout`
- `javax.persistence.cache.retrieveMode`
- `javax.persistence.cache.storeMode`

For Hibernate specific QueryHints, refer to "3.4.1.8. Query hints" of Hibernate EntityManager User guide.

Specifying a query while calling a query method

The method of specifying a query to be executed while calling query method is given below.

- *Specifying the query using @Query annotation*
- *Specifying with the method name based on naming conventions*
- *Specifying as Named query in Properties file*

Specifying the query using @Query annotation

Specify the query(JPQL) to be executed in value attribute of `@Query` annotation.

```
@Query(value = "SELECT o FROM Order o WHERE o.status.code = :statusCode ORDER␣
↪BY o.id DESC") // (1)
List<Order> findByStatusCode(@Param("statusCode") String statusCode);
```

```
-- (2) statusCode='accepted'
SELECT
        order0_.id AS id1_5_
        ,order0_.status_code AS status2_5_
    FROM
        t_order order0_
    WHERE
        order0_.status_code = 'accepted'
    ORDER BY
        order0_.id DESC
```

Sr. No.	Description
(1)	Specify the query(JPQL) to be executed in value attribute of `@Query` annotation. In the above example, query for fetching the `Order` object in the descending order of `id` property has been specified. Here, the `code` property (`String` type) value of `status` property (`OrderStatus` type) stored in `Order` object is matched with the specified parameter value (`statusCode`).
(2)	Native SQL converted from JPQL. Query(JPQL) specified in value attribute of `@Query` annotation is converted to Native SQL of database to be used.

Note: How to specify Native SQL directly instead of JPQL

Native SQL can be specified as query instead of JPQL by setting nativeQuery attribute to `true`. **Fundamentally it is recommended that you use JPQL; however, when there is a need to generate the**

query that cannot be expressed in JPQL, Native SQL can be specified directly. To specify the database dependent SQL, analyze whether it can be defined in the properties file.

For method of defining SQL in properties file, refer to "*Specifying as Named query in Properties file*".

Note: Named Parameters

Named parameter can be used by assigning a name to bind parameter of the query and using this assigned name to specify the value. To use Named Parameter, add `@org.springframework.data.repository.query.Param` annotation to the argument from which the value has to be used to bind to the named parameter in the query. Specify the assigned parameter name in value attribute of param annotation. ON the query side, at the position where parameter is to be bound in the query, specify it in the ":parametername" format.

When there is no specific reason, It is recommended to use Named Parameters considering maintainability and readability of query.

In case of LIKE search, if the type of matching (Forward match, Backward match and Partial match) is fixed, `"%"` can be specified in JPQL.

However, this is in extended Spring Data JPA format and not in standard JPQL, so it can be specified only in JPQL specified with `@Query` annotation.

An error occurs if `"%"` is specified in JPQL specified as Named query.

Sr. No.	Type of matching	Format	Specific example
1.	Forward match	`:parameterName%` or `?n%`	`SELECT a FROM Account WHERE a.firstName LIKE :firstName%` `SELECT a FROM Account WHERE a.firstName LIKE ?1%`
2.	Backward match	`%:parameterName` or `%?n`	`SELECT a FROM Account WHERE a.firstName LIKE %:firstName` `SELECT a FROM Account WHERE a.firstName LIKE %?1`
3.	Partial match	`%:parameterName%` or `%?n%`	`SELECT a FROM Account WHERE a.firstName LIKE %:firstName%` `SELECT a FROM Account WHERE a.firstName LIKE %?1%`

Note: Escaping at the time of LIKE search

Search condition values should be escaped during LIKE search.

The method for escaping these values is provided in `org.terasoluna.gfw.common.query.QueryEscapeUtils` class; this class can be used if it meets the requirements. For details on `QueryEscapeUtils` class, refer to "*Escaping during LIKE search*" of "*Database Access (Common)*".

Note: When the type of matching needs to be changed dynamically

When it is necessary to change the type of matching (Forward match, Backward match and Partial match) dynamically, `"%"` should be added before and after the parameter value (same as conventional method), instead of specifying `%` in JPQL.

The method for converting into search condition value corresponding to the type of matching is provided in `org.terasoluna.gfw.common.query.QueryEscapeUtils` class; this class can be used if it meets the requirements. For details on `QueryEscapeUtils` class, refer to "*Escaping during LIKE search*" of "*Database Access (Common)*".

Sort conditions can be directly specified in query.

See the example below.

```
// (1)
@Query(value = "SELECT o FROM Order o WHERE o.status.code = :statusCode ORDER␣
↪BY o.id DESC")
Page<Order> findByStatusCode(@Param("statusCode") String statusCode, Pageable␣
↪pageable);
```

Sr. No.	Description
(1)	Specify "ORDER BY" in query. Specify DESC for sorting in descending order and ASC for ascending order. By default it is "ASC", when nothing is specified.

In addition to directly specifying the sort conditions in query, they can be specified in `org.springframework.data.domain.Sort` object stored in `Pageable` object.

When specifying the sort conditions using this method, there is no need to specify countQuery attribute.

Example of sorting using `Sort` object stored in `Pageable` object is shown below.

- Controller

```
@RequestMapping("list")
public String list(@PageableDefault(
                        size=5,
                        sort = "id", // (1)
                        direction = Direction.DESC // (1)
                        ) Pageable pageable,
                          Model model) {
    Page<Order> orderPage = orderService.getOrders(pageable); // (2)
    model.addAttribute("orderPage", orderPage);
    return "order/list";
}
```

Sr. No.	Description
(1)	Specify the sort conditions. Sort conditions are set in `Sort` object that can be fetched by `Pageable#getSort()` method. In the above example, DESC is specified as a sort condition for id field.
(2)	Specify `Pageable` object and call Service method.

- Service (Caller)

```
public String getOrders(Pageable pageable){
    return orderRepository.findByStatusCode("accepted", pageable); // (3)
}
```

Sr. No.	Description
(3)	Call Repository method by specifying `Pageable` object passed by Controller.

- Repository interface

```
@Query(value = "SELECT o FROM Order o WHERE o.status.code = :statusCode") //␣
↪(4)
Page<Order> findByStatusCode(@Param("statusCode") String statusCode, Pageable␣
↪pageable);
```

```
-- (5) statusCode='accepted'
SELECT
        COUNT(order0_.id) AS col_0_0_
    FROM
        t_order order0_
    WHERE
        order0_.status_code = 'accepted'

-- (6) statusCode='accepted'
SELECT
```

```
    order0_.id AS id1_5_
    ,order0_.status_code AS status2_5_
FROM
    t_order order0_
WHERE
    order0_.status_code = 'accepted'
ORDER BY
    order0_.id DESC
LIMIT 5
```

Sr. No.	Description
(4)	Do not specify "ORDER BY" clause in query. No need to specify countQuery attribute also.
(5)	Native SQL for count converted from JPQL.
(6)	Native SQL for fetching the entity of the specified page location converted from JPQL. Not specified in query; however "ORDER BY" clause is added to the condition specified in `Sort` object stored in `Pageable` object. In this example, it is SQL for PostgreSQL.

Specifying with the method name based on naming conventions

Specify the query (JPQL) to be executed through method name as per the naming conventions of Spring Data JPA.

JPQL is created from the method name by the functionality of Spring Data JPA.

However, this is only possible for SELECT queries and not for UPDATE and DELETE.

For naming conventions for creating JPQL, refer to the following pages.

Sr. No.	Reference page	Description
1.	Spring Data Commons - “Query creation” of Reference Documentation “Defining query methods”	This section describes method to specify Distinct, ORDER BY and Case insensitive.
2.	Spring Data Commons - “Property expressions” of Reference Documentation “Defining query methods”	This section describes method to specify the nested entity property in condition.
3.	Spring Data Commons - “Special parameter handling” of Reference Documentation “Defining query methods”	This section describes special method arguments (`Pageable` , `Sort`).
4.	Spring Data JPA - “Query creation” of Reference Documentation “Query methods”	This section describes naming conventions (keywords) for creating JPQL.
5.	Spring Data Commons - Reference Documentation “Appendix C. Repository query keywords”	This section describes naming conventions (keywords) for creating JPQL.

See the example below.

- OrderRepositry.java

```
Page<Order> findByStatusCode(String statusCode, Pageable pageable); // (1)
```

Sr. No.	Description
(1)	When the method name matches with `^(find|read|get).*By(.+)` pattern, JPQL is created from method name. In the `(.+)` portion, specify the property of entity which forms the query condition or keywords indicating the operation. In the example, `Order` object where `code` property (`String` type) value of `status` property (`OrderStatus` type) stored in `Order` object is matched with the specified parameter value (`statusCode`), is being fetched in page format.

- Count Query

```
-- (2) JPQL
SELECT
        COUNT(*)
    FROM
        ORDER AS generatedAlias0
            LEFT JOIN generatedAlias0.status AS generatedAlias1
        WHERE
            generatedAlias1.code = ?1

-- (3) SQL statusCode='accepted'
SELECT
```

```
        COUNT(*) AS col_0_0_
    FROM
        t_order order0_
            LEFT OUTER JOIN c_order_status orderstatu1_
                ON order0_.status_code = orderstatu1_.code
    WHERE
        orderstatu1_.code = 'accepted'
```

Sr. No.	Description
(2)	JPQL query for count created from method name.
(3)	Native SQL for count converted from JPQL of step (2).

- Query for fetching entities

```
-- (4) JPQL
SELECT
        generatedAlias0
    FROM
        ORDER AS generatedAlias0
            LEFT JOIN generatedAlias0.status AS generatedAlias1
        WHERE
            generatedAlias1.code = ?1
        ORDER BY
            generatedAlias0.id DESC;

-- (5) statusCode='accepted'
SELECT
        order0_.id AS id1_5_
        ,order0_.status_code AS status2_5_
    FROM
        t_order order0_
            LEFT OUTER JOIN c_order_status orderstatu1_
                ON order0_.status_code = orderstatu1_.code
    WHERE
        orderstatu1_.code = 'accepted'
    ORDER BY
        order0_.id DESC
    LIMIT 5
```

Sr. No.	Description
(4)	JPQL query for fetching entities created from method name.
(5)	Native SQL for fetching the entities converted from JPQL of step (4).

Specifying as Named query in Properties file

Specify the query in the properties file (classpath:META-INF/jpa-named-queries.properties) of Spring Data JPA.

Consider using this method when it is required to write a database platform specific SQL at the time of using NativeQuery.

Even if it is database platform specific SQL, it is recommended that you use a method to directly specify in @Query annotation when there is no dependency on the execution environment.

- OrderRepositry.java

```
@Query(nativeQuery = true)
List<Order> findAllByStatusCode(@Param("statusCode") String statusCode); //␣
↪(1)
```

Sr. No.	Description
(1)	Regarding the lookup name of named query, class name of entity and method name linked by `"."` (dot) is used. In the above example, `"Order.findAllByStatusCode"` is used as Lookup name.

Tip: Specifying Lookup name of Named query

As per default behavior, Lookup name is contructed by connecting class name of the entity linked with method name using `"."` (dot). However, any name can be specified.

- For fetching entities, specify it in name attribute of `@Query` annotation.
- For count at the time of page search, specify it in countName attribute of `@Query` annotation.

```
@Query(name = "OrderRepository.findAllByStatusCode", nativeQuery = true) //␣
↪(2)
List<Order> findAllByStatusCode(@Param("statusCode") String statusCode);
```

Sr. No.	Description
(2)	In the above example, "OrderRepository.findAllByStatusCode" is specified as the Lookup name for the query.

- jpa-named-queries.properties

```
# (3)
Order.findAllByStatusCode=SELECT * FROM order WHERE status_code = :statusCode
```

Sr. No.	Description
(3)	Specify the SQL to be executed by using lookup name for the query as the key. In the above example, the SQL to be executed has been specified by using "Order.findAllByStatusCode" as key.

Tip: The method of specifying Named Query in any properties file instead of the properties file of Spring Data JPA is explained below.

- xxx-infra.xml

```
<!-- (4) -->
<jpa:repositories base-package="xxxxxx.yyyyyy.zzzzzz.domain.repository"
    named-queries-location="classpath:META-INF/jpa/jpa-named-queries.
↪properties" />
```

Sr. No.	Description
(4)	Specify any properties file in named-queries-location attribute of <jpa:repositories> element. In the above example, META-INF/jpa/jpa-named-queries.properties on class path is used.

Implementing the process to search entities

The method to search entities is explained below.

Searching all entities matching the conditions

Call a query method to fetch all entities that match the conditions.

- Repository interface

```
public interface AccountRepository extends JpaRepository<Account, String> {

    // (1)
    @Query("SELECT a FROM Account a WHERE :createdDateFrom <= a.createdDate␣
↪AND a.createdDate < :createdDateTo ORDER BY a.createdDate DESC")
    List<Account> findByCreatedDate(
            @Param("createdDateFrom") Date createdDateFrom,
            @Param("createdDateTo") Date createdDateTo);

}
```

Sr. No.	Description
(1)	Define a query method to return `java.util.List` interface.

- Service

```
public List<Account> getAccounts(Date targetDate) {
    LocalDate targetLocalDate = new LocalDate(targetDate);
    Date fromDate = targetLocalDate.toDate();
    Date toDate = targetLocalDate.dayOfYear().addToCopy(1).toDate();

    // (2)
    List<Account> accounts = accountRepository.findByCreatedDate(fromDate,
            toDate);
    if (accounts.isEmpty()) { // (3)
        // ...
    }
    return accounts;
}
```

Sr. No.	Description
(2)	Call the query method defined in Repository interface.
(3)	If the search result is 0 records, a blank list is returned. As null is not returned, null check is not required. If needed, implement the process when the search result is 0 records.

Searching page of entities matching the conditions

Amongst the entities matching the conditions, call a query method to fetch the entities of the specified page.

- Repository interface

```
public interface AccountRepository extends JpaRepository<Account, String> {

    // (1)
    @Query("SELECT a FROM Account a WHERE :createdDateFrom <= a.createdDate␣
↪AND a.createdDate < :createdDateTo")
    Page<Account> findByCreatedDate(
            @Param("createdDateFrom") Date createdDateFrom,
            @Param("createdDateTo") Date createdDateTo, Pageable pageable);

}
```

Sr. No.	Description
(1)	Receive `org.springframework.data.domain.Pageable` interface as an argument and define query method for returning `org.springframework.data.domain.Page` interface.

- Controller

```
@RequestMapping("list")
public String list(@RequestParam("targetDate") Date targetDate,
                   @PageableDefault(
                       page = 0,
                       value = 5,
                       sort = { "createdDate" },
                       direction = Direction.DESC)
                       Pageable pageable, // (2)
                   Model model) {
    Page<Order> accountPage = accountService.getAccounts(targetDate,␣
↪pageable);
    model.addAttribute("accountPage", accountPage);
    return "account/list";
}
```

Sr. No.	Description
(2)	Create object (`org.springframework.data.domain.Pageable`) for paging search provided by Spring Data. For details, refer to "*Pagination*".

- Service

```
public Page<Account> getAccounts(Date targetDate ,Pageable pageable) {

    LocalDate targetLocalDate = new LocalDate(targetDate);
    Date fromDate = targetLocalDate.toDate();
    Date toDate = targetLocalDate.dayOfYear().addToCopy(1).toDate();

    // (3)
    Page<Account> page = accountRepository.findByCreatedDate(fromDate,
            toDate, pageable);
    if (!page.hasContent()) { // (4)
        // ...
    }
    return page;
}
```

Sr. No.	Description
(3)	Call the query method defined in Repository interface.
(4)	If the search result is 0 records, a blank list will be set in `Page` object and `false` will be returned for `Page#hasContent()` method. If needed, implement the process when the search result is 0 records.

Implementing search process as per the dynamic conditions of entities

To add query method to Repository for searching the entities as per dynamic conditions, search process should be implemented by creating custom Repository interface and custom Repository class for the entity specific Repository interface. For method of creating custom Repository interface and custom Repository class, refer to "*Adding individual custom method to entity specific Repository interface*".

See the description below to search entities by applying dynamic conditions.

Todo

TBD

Following contents will be added in future.

- Example illustrating implementation of dynamic query using QueryDSL.

Searching all entities matching the dynamic conditions

Implement and call the query method for fetching all entities matching the dynamic conditions.

See the example below.

Here, the conditions below are specified as dynamic conditions.

- Order ID
- Product name
- Order status (multiple statuses can be specified)

Further, the search is narrowed down using AND operator for the orders matching the specified conditions. If no condition is specified, a blank list will be returned.

- Criteria (JavaBean)

```
public class OrderCriteria implements Serializable { // (1)

    private Integer id;

    private String itemName;

    private List<String> statusCodes;

    // ...

}
```

Sr. No.	Description
(1)	Create a Criteria object (JavaBean) storing the search conditions.

- Custom Repository interface

```
public interface OrderRepositoryCustom {

    Page<Order> findAllByCriteria(OrderCriteria criteria); // (2)

}
```

Sr. No.	Description
(2)	Define a method in custom Repository interface which receives Criteria object as an argument and returns List of Order objects.

- Custom Repository class

```
public class OrderRepositoryImpl implements OrderRepositoryCustom { // (3)

    @PersistenceContext
    EntityManager entityManager; // (4)

    public List<Order> findAllByCriteria(OrderCriteria criteria) { // (5)

        // Collect dynamic conditions.
        // (6)
        final List<String> andConditions = new ArrayList<String>();
        final List<String> joinConditions = new ArrayList<String>();
        final Map<String, Object> bindParameters = new HashMap<String, Object>
↪();

        // (7)
        if (criteria.getId() != null) {
            andConditions.add("o.id = :id");
            bindParameters.put("id", criteria.getId());
        }
        if (!CollectionUtils.isEmpty(criteria.getStatusCodes())) {
            andConditions.add("o.status.code IN :statusCodes");
            bindParameters.put("statusCodes", criteria.getStatusCodes());
        }
        if (StringUtils.hasLength(criteria.getItemName())) {
            joinConditions.add("o.orderItems oi");
            joinConditions.add("oi.item i");
            andConditions.add("i.name LIKE :itemName ESCAPE '~'");
            bindParameters.put("itemName", QueryEscapeUtils
```

```
                .toLikeCondition(criteria.getItemName()));
        }

        // (8)
        if (andConditions.isEmpty()) {
            return Collections.emptyList();
        }

        // (9)
        // Create dynamic query.
        final StringBuilder queryString = new StringBuilder();

        // (10)
        queryString.append("SELECT o FROM Order o");

        // (11)
        // add join conditions.
        for (String joinCondition : joinConditions) {
            queryString.append(" LEFT JOIN ").append(joinCondition);
        }
        // add conditions.
        Iterator<String> andConditionsIt = andConditions.iterator();
        if (andConditionsIt.hasNext()) {
            queryString.append(" WHERE ").append(andConditionsIt.next());
        }
        while (andConditionsIt.hasNext()) {
            queryString.append(" AND ").append(andConditionsIt.next());
        }

        // (12)
        // add order by condition.
        queryString.append(" ORDER BY o.id");

        // (13)
        // Create typed query.
        final TypedQuery<Order> findQuery = entityManager.createQuery(
                queryString.toString(), Order.class);
        // Bind parameters.
        for (Map.Entry<String, Object> bindParameter : bindParameters
                .entrySet()) {
            findQuery.setParameter(bindParameter.getKey(), bindParameter
                    .getValue());
        }

        // (14)
        // Execute query.
        return findQuery.getResultList();

    }
```

```
}
```

Sr. No.	Description
(3)	Create implementation class of custom Repository interface.
(4)	Inject `EntityManager`. Inject using `@javax.persistence.PersistenceContext` annotation.
(5)	Implement the query method to fetch all the entities matching the dynamic conditions. In the above example, the method is not split for explanation purpose; however it can be split if required.
(6)	Define the variables (list for AND condition, list for join condition, bind parameter map) to build dynamic queries. Set the required information in the variables for the items for which conditions are specified in OrderCriteria object.
(7)	Determine whether conditions are specified in OrderCriteria object and set the required information for building dynamic queries. In the above example, the information is set to fetch the items wherein `id` is completely matched with the specified value, `statusCodes` is included in the specified list, and `itemName` satisfying forward match with the specified value. For `itemName`, the related-entities having the values to be compared have a complex nested relation; hence these entities should be joined.
(8)	In the above example, when conditions are not specified, return a blank list as there is no need to perform search.
(9)	When conditions are specified, build the query for searching entities. In the above example, query to be executed is built using `java.lang.StringBuilder` class.

次のページに続く

Table. 6.19 – 前のページからの続き

Sr. No.	Description
(10)	Build static query elements. In the above example, SELECT clause and FROM clause are built as static query elements.
(11)	Build dynamic query elements. In the above example, conditions to be set in join conditions list (JOIN clause) and AND conditions list (WHERE clause) are built as dynamic query elements.
(12)	Build the static query elements. In the above example, ORDER BY clause is built as static query elements.
(13)	Convert the dynamically built query string into `javax.persistence.TypedQuery`, and set the bind parameters required for executing the query.
(14)	Execute the dynamically built query and fetch all the entities matching the conditions.

- Entity specific Repository interface

```
public interface OrderRepository extends JpaRepository<Order, Integer>,
                                 OrderRepositoryCustom { // (15)
    // ...
}
```

Sr. No.	Description
(15)	Inherit the custom Repository interface in entity specific Repository interface.

- Service (Caller)

```
// condition values for sample.
Integer conditionValueOfId = 4;
List<String> conditionValueOfStatusCodes = Arrays.asList("accepted");
String conditionValueOfItemName = "Wat";

// implementation of sample.
// (16)
OrderCriteria criteria = new OrderCriteria();
criteria.setId(conditionValueOfId);
criteria.setStatusCodes(conditionValueOfStatusCodes);
criteria.setItemName(conditionValueOfItemName);
List<Order> orders = orderRepository.findAllByCriteria(criteria); // (17)
if (orders.isEmpty()) { // (18)
    // ...
}
```

Sr. No.	Description
(16)	Specify the search conditions in OrderCriteria object.
(17)	Use OrderCriteria as an argument and call the query method to get all the entities matching the dynamic conditions.
(18)	Analyze the search results and perform the processing required in case of 0 records.

- Executed JPQL(SQL)

```
-- (19)
--   conditionValueOfId=4
```

```
--   conditionValueOfStatusCodes = ["accepted"]
--   conditionValueOfItemName = "Wat"

-- JPQL
SELECT
        o
    FROM
        ORDER o
            JOIN o.orderItems oi
                JOIN oi.item i
            WHERE
                o.id = :id
                AND o.status.code IN :statusCodes
                AND i.name LIKE :itemName ESCAPE '~'
            ORDER BY
                o.id

-- SQL
SELECT
        order0_.id AS id1_6_
        ,order0_.created_by AS created2_6_
        ,order0_.created_date AS created3_6_
        ,order0_.last_modified_by AS last4_6_
        ,order0_.last_modified_date AS last5_6_
        ,order0_.status_code AS status6_6_
    FROM
        t_order order0_ INNER JOIN t_order_item orderitems1_
            ON order0_.id = orderitems1_.order_id INNER JOIN m_item item2_
            ON orderitems1_.item_code = item2_.code
WHERE
    order0_.id = 4
    AND (
        order0_.status_code IN ('accepted')
    )
    AND (
        item2_.name LIKE 'Wat%' ESCAPE '~'
    )
ORDER BY
    order0_.id
```

Sr. No.	Description
(19)	Example of generated JPQL and SQL when all the conditions are specified.

```
-- (20)
--   conditionValueOfId=4
--   conditionValueOfStatusCodes = ["accepted"]
--   conditionValueOfItemName = ""
```

```
-- JPQL
SELECT
        o
    FROM
        ORDER o
    WHERE
        o.id = :id
        AND o.status.code IN :statusCodes
    ORDER BY
        o.id

-- SQL
SELECT
        order0_.id AS id1_6_
        ,order0_.created_by AS created2_6_
        ,order0_.created_date AS created3_6_
        ,order0_.last_modified_by AS last4_6_
        ,order0_.last_modified_date AS last5_6_
        ,order0_.status_code AS status6_6_
    FROM
        t_order order0_
    WHERE
        order0_.id = 4
        AND (
            order0_.status_code IN ('accepted')
        )
    ORDER BY
        order0_.id;
```

Sr. No.	Description
(20)	Example of generated JPQL and SQL when conditions other than `iteName` are specified.

```
-- (21)
--   conditionValueOfId=4
--   conditionValueOfStatusCodes = []
--   conditionValueOfItemName = ""
-- JPQL
SELECT
        o
    FROM
        ORDER o
    WHERE
        o.id = :id
    ORDER BY
        o.id

-- SQL
```

```
SELECT
        order0_.id AS id1_6_
        ,order0_.created_by AS created2_6_
        ,order0_.created_date AS created3_6_
        ,order0_.last_modified_by AS last4_6_
        ,order0_.last_modified_date AS last5_6_
        ,order0_.status_code AS status6_6_
    FROM
        t_order order0_
    WHERE
        order0_.id = 4
    ORDER BY
        order0_.id;
```

Sr. No.	Description
(21)	Example of generated JPQL and SQL when only `id` is specified.

Page search for the entities matching the dynamic conditions

Implement and call the query method to fetch the entities corresponding to the specified page, amongst the entities matching the dynamic conditions.

As shown in the example below, the specification is same as that for normal search except for fetching the corresponding page. Further, the description for fetching all records is omitted.

- Custom Repository interface

```
public interface OrderRepositoryCustom {

    Page<Order> findPageByCriteria(OrderCriteria criteria, Pageable pageable);
↪ // (1)

}
```

Sr. No.	Description
(1)	Define the query method to fetch the entities corresponding to the specified page, amongst the entities matching the dynamic conditions.

- Custom Repository class

```
public class OrderRepositoryCustomImpl implements OrderRepositoryCustom {

    @PersistenceContext
    EntityManager entityManager;

    public Page<Order> findPageByCriteria(OrderCriteria criteria,
            Pageable pageable) { // (2)

        // collect dynamic conditions.
        final List<String> andConditions = new ArrayList<String>();
        final List<String> joinConditions = new ArrayList<String>();
        final Map<String, Object> bindParameters = new HashMap<String, Object>
↪();

        if (criteria.getId() != null) {
            andConditions.add("o.id = :id");
            bindParameters.put("id", criteria.getId());
        }
        if (!CollectionUtils.isEmpty(criteria.getStatusCodes())) {
            andConditions.add("o.status.code IN :statusCodes");
            bindParameters.put("statusCodes", criteria.getStatusCodes());
        }
        if (StringUtils.hasLength(criteria.getItemName())) {
            joinConditions.add("o.orderItems oi");
            joinConditions.add("oi.item i");
            andConditions.add("i.name LIKE :itemName ESCAPE '~'");
            bindParameters.put("itemName", QueryEscapeUtils.
↪toLikeCondition(criteria
                    .getItemName()));
        }

        if (andConditions.isEmpty()) {
            List<Order> orders = Collections.emptyList();
            return new PageImpl<Order>(orders, pageable, 0); // (3)
        }

        // create dynamic query.
        final StringBuilder queryString = new StringBuilder();
        final StringBuilder countQueryString = new StringBuilder(); // (4)
        final StringBuilder conditionsString = new StringBuilder(); // (4)

        queryString.append("SELECT o FROM Order o");
        countQueryString.append("SELECT COUNT(o) FROM Order o"); // (5)

        // add join conditions.
        for (String joinCondition : joinConditions) {
            conditionsString.append(" JOIN ").append(joinCondition);
        }

        // add conditions.
```

```
        Iterator<String> andConditionsIt = andConditions.iterator();
        if (andConditionsIt.hasNext()) {
            conditionsString.append(" WHERE ").append(andConditionsIt.next());
        }
        while (andConditionsIt.hasNext()) {
            conditionsString.append(" AND ").append(andConditionsIt.next());
        }
        queryString.append(conditionsString); // (6)
        countQueryString.append(conditionsString); // (6)

        // add order by condition.
        // (7)
        String orderByString = QueryUtils.applySorting("", pageable.getSort(),
↪ "o");
        queryString.append(orderByString);

        // create typed query.
        final TypedQuery<Long> countQuery = entityManager.createQuery(
                countQueryString.toString(), Long.class); // (8)

        final TypedQuery<Order> findQuery = entityManager.createQuery(
                queryString.toString(), Order.class);

        // bind parameters.
        for (Map.Entry<String, Object> bindParameter : bindParameters
                .entrySet()) {
            countQuery.setParameter(bindParameter.getKey(), bindParameter
                    .getValue()); // (8)
            findQuery.setParameter(bindParameter.getKey(), bindParameter
                    .getValue());
        }

        long total = countQuery.getSingleResult().longValue(); // (9)
        List<Order> orders = null;
        if (total != 0) { // (10)
            findQuery.setFirstResult(pageable.getOffset());
            findQuery.setMaxResults(pageable.getPageSize());
            // execute query.
            orders = findQuery.getResultList();
        } else { // (11)
            orders = Collections.emptyList();
        }

        return new PageImpl<Order>(orders, pageable, total); // (12)
    }

}
```

Sr. No.	Description
(2)	Implement the query method to fetch the entities corresponding to the specified page, amongst the entities matching the dynamic conditions. In the above example, the method is not split for explanation purpose; however it can be split if required.
(3)	In the above example, when conditions are not specified, return a blank page information as there is no need to perform search.
(4)	Create variables to build the queries to fetch records and to build the conditions (join condition and AND condition). Same conditions need to be used in query for fetching entities and in query for fetching records; hence variables are created to build the conditions (join condition and AND condition).
(5)	Build the static query elements of query to fetch records. In the above example, SELECT clause and FROM clause are built as static query components.
(6)	Build the dynamic query elements for query to fetch entities and query to fetch records.
(7)	Sort conditions (ORDER BY clause) are built as dynamic query elements for the query to fetch entities. Utility component (`org.springframework.data.jpa.repository.query.QueryUtils`) provided by Spring Data JPA is used for building the ORDER BY clause.
(8)	Convert the query string for fetching the dynamically built records into `javax.persistence.TypedQuery` and set the bind parameter for required for executing the query.

次のページに続く

Table. 6.20 – 前のページからの続き

Sr. No.	Description
(9)	Execute the query for fetching the records and get the total number of records matching the conditions.
(10)	If the entities matching the conditions exist, execute the query for fetching the entities and fetch the information of the corresponding page. When executing the query for fetching entities, specify the start index for fetching records (`TypedQuery#setFirstResult`) and the number of records to be fetched (`TypedQuery#setMaxResults`).
(11)	If entity matching the conditions does not exist, a blank list should be fetched.
(12)	Specify entity list of the corresponding page, page information and total number of records matching the conditions as arguments and then create and return the `Page` object.

- Service (Caller)

```
// condition values for sample.
Integer conditionValueOfId = 4;
List<String> conditionValueOfStatusCodes = Arrays.asList("accepted");
String conditionValueOfItemName = "Wat";

// implementation of sample.
OrderCriteria criteria = new OrderCriteria();
criteria.setId(conditionValueOfId);
criteria.setStatusCodes(conditionValueOfStatusCodes);
criteria.setItemName(conditionValueOfItemName);
Page<Order> orderPage = orderRepository.findPageByCriteria(criteria,
        pageable); // (13)
if (!orderPage.hasContent()) {
    // ...
}
```

Sr. No.	Description
(13)	Call the query method to fetch the entities corresponding to the specified page, amongst the entities matching the dynamic conditions.

Implementing the process to fetch entities

The method of fetching entities is explained below.

Fetching 1 record of entity by specifying ID

If the ID (Primary Key) is known, fetch the entity object by calling the findOne method of Repository interface.

```
public Account getAccount(String accountUuid) {
    Account account = accountRepository.findOne(accountUuid); // (1)
    if (account == null) { // (2)
        // ...
    }
    return account;
}
```

Sr. No.	Description
(1)	Specify the ID (Primary Key) of entity and call the findOne(ID) method of Repository interface.
(2)	When the specified entity ID does not exist, the return value would be null, hence null check is necessary. If needed, implement the process which is carried out when the specified entity ID does not exist.

Note: Returned entity objects

When the entity object of specified ID is already managed by `EntityManager`, entity object managed by `EntityManager` is returned without accessing the persistence layer (DB). Therefore, if findOne method is used, unnecessary access to persistence layer can be controlled.

Note: Load timing of the related-entity

Load of the related-entity during query execution is determined based on the value specified in fetch attribute of annotations (`@javax.persistence.OneToOne`, `@javax.persistence.OneToMany`, `@javax.persistence.ManyToOne`, `@javax.persistence.ManyToMany`).

- In case of `javax.persistence.FetchType#LAZY`, related-entity is not covered under JOIN FETCH; hence it is loaded at the time of initial access.
- In case of `javax.persistence.FetchType#EAGER`, related-entity is covered under JOIN FETCH; hence it is loaded at the time of loading the parent-entity.

Default values of fetch attribute differ depending on annotations. See the default values below:

- `@OneToOne` annotation: `EAGER`
- `@ManyToOne` annotation: `EAGER`
- `@OneToMany` annotation: `LAZY`
- `@ManyToMany` annotation: `LAZY`

Note: Sort order of the related-entities having 1:N(N:M) relationship

Specify `@javax.persistence.OrderBy` annotation on the property of related-entities to control the sort order.

See the example below.

```
@OneToMany(mappedBy = "order", cascade = CascadeType.ALL, orphanRemoval =␣
↪true)
@OrderBy // (1)
private Set<OrderItem> orderItems;
```

Sr. No.	Description
(1)	When value attribute is not specified, the entities are sorted in ascending order of ID. For details, refer to JSR 338: Java Persistence API, Specification (PDF) of Version 2.1 "11.1.42 OrderBy Annotation".

Todo

TBD

Following contents will be added in future.

- Example of cases wherein it is better to change fetch attribute from default value.

Fetching 1 record of entity by specifying conditions other than ID

When the ID is not known, call the query method to search entities by conditions other than ID.

- Repository interface

```
public interface AccountRepository extends JpaRepository<Account, String> {

    // (1)
    @Query("SELECT a FROM Account a WHERE a.accountId = :accountId")
    Account findByAccountId(@Param("accountId") String accountId);

}
```

Sr. No.	Description
(1)	Define the query method to search 1 record of entity by specifying items other than ID as conditions.

- Service

```
public Account getAccount(String accountId) {
    Account account = accountRepository.findByAccountId(accountId); // (2)
    if (account == null) { // (3)
        // ...
    }
```

Sr. No.	Description
(2)	Call the query method defined in Repository interface.
(3)	Similar to findOne method of Repository interface, when the entities matching the conditions do not exist, null will be returned. Hence null check is necessary. If needed, implement the process for the scenario when entities matching the specified conditions do not exist. When multiple entities matching the conditions exist, `org.springframework.dao.IncorrectResultSizeDataAccessException` occurs.

Note: Returned entity objects

When a query method is called, the query is always executed on persistence layer (DB). However, when the entities which are fetched by executing the query are already being managed in `EntityManager`, the entity objects fetched from DB are discarded and the entity objects managed in `EntityManager` are returned.

Note: Query method using ID + α as condition

It is recommended not to create a query method wherein ID + α is used as condition. It can be implemented by creating a logic that compares property value of entity objects fetched by calling findOne method.

The reason for not recommending the creation of this query method is that there is a possibility of the entity objects fetched by executing the query getting discarded and unnecessary query getting executed. If the

ID is known, it is desirable to use findOne method which prevents execution of unnecessary queries. This should be consciously implemented especially in case of applications with high performance requirements.

However, if the following conditions are applicable, use of query method may reduce the frequency of query execution; hence query method can be used in such cases.

- Properties of related objects (column of related table) are included in a part of + α condition.
- `LAZY` is included in FetchType of the related-entities as a condition.

Note: Load timing of the related-entities

Related-entities specified in JOIN FETCH are loaded immediately after executing the query.

The related-entities not specified in JOIN FETCH performs the following operations as per the values specified in fetch attribute of associated annotations (`@OneToOne` , `@OneToMany` , `@ManyToOne` , `@ManyToMany`).

- `javax.persistence.FetchType#LAZY` is covered under Lazy Load; hence the related-entities are loaded at the time of initial access.
- In case of `javax.persistence.FetchType#EAGER`, query is executed to load the related-entities and objects of the related-entities are loaded.

Note: Sort order of the related-entities having 1:N(N:M) relationship

- The sort order of the related-entities specified in JOIN FETCH is controlled by specifying "ORDER BY" clause in JPQL.
- The sort order of the related-entities loaded after executing the query is controlled by specifying `@javax.persistence.OrderBy` annotation to the property of the related-entities.

Adding entities

Example of adding entities is shown below.

How to add entities

In order to add an entity, create an entity object and call the save method of Repository interface.

- Service

```
Order order = new Order("accepted"); // (1)
order = orderRepository.save(order); // (2)
```

Sr. No.	Description
(1)	Create an instance of entity object and set the values for required properties. In the above example, ID generator of JPA is used for setting the ID. When ID generator of JPA is to be used, ID should not be set in the application code. If you set the ID in the application code, merge method of `EntityManager` gets called leading to execution of unnecessary processing.
(2)	Call the save method of Repository interface and manage the entity objects created in (1) in `EntityManager`. Take note that entity object passed as an argument of save method will not be the one managed by `EntityManager`, but the entity object returned by save method will be the one managed by `EntityManager`. ID is set by the ID generator of JPA at the time of this process.

Note: Demerits of the merge method getting called

merge method of `EntityManager` has a mechanism to fetch the entities having same ID from persistence layer (DB), when the entities are to be managed in `EntityManager`. Process to fetch the entities becomes unnecessary while adding the entities. In case of application with high performance requirements, ID generation timing should also be taken into account.

Note: Constraint error handling

When save method is called, query (INSERT) is not executed in the persistence layer (DB). Therefore, when constraint error such as unique constraint violation needs to be handled, saveAndFlush method or flush method should be called instead of save method of Repository interface.

- Entity

```
@Entity // (3)
@Table(name = "t_order") // (4)
public class Order implements Serializable {

    // (5)
    @SequenceGenerator(name = "GEN_ORDER_ID", sequenceName = "s_order_id",
                       allocationSize = 1)
    @GeneratedValue(strategy = GenerationType.SEQUENCE,
                    generator = "GEN_ORDER_ID")
    @Id // (6)
    private int id;

    // (7)
    @ManyToOne
    @JoinColumn(name = "status_code")
    private OrderStatus status;

    // ...

    public Order(String statusCode) {
        this.status = new OrderStatus(statusCode);
    }

    // ...

}
```

Sr. No.	Description
(3)	Assign `@javax.persistence.Entity` annotation to the entity class.
(4)	Specify the table name to be mapped with the entity class in name attribute of `@javax.persistence.Table` annotation. The table name need not be specified if it can be resolved from entity name; however, it has to be specified if it cannot be resolved from entity name.
(5)	The annotations required for using ID generator of JPA are being specified. When using ID generator of JPA, specify `@javax.persistence.GeneratedValue` annotation. When using sequence object, `@javax.persistence.SequenceGenerator` annotation should be specified. When using table generator, `@javax.persistence.TableGenerator` annotation should be specified. In the above example, ID is generated using the sequence object called `"s_order_id"` name.
(6)	Assign `@javax.persistence.Id` annotation to the property that holds the primary key. In case of composite key, assign `@javax.persistence.EmbeddedId` annotation.
(7)	Assign the associated annotations (`@OneToOne` , `@OneToMany` , `@ManyToOne` , `@ManyToMany`) to the property that has a relationship with other entities.

Note: Annotations for generating IDs

For details on each annotation, refer to JSR 338: Java Persistence API, Version 2.1 Specification (PDF).

- `@GeneratedValue` : 11.1.20 GeneratedValue Annotation
- `@SequenceGenerator` : 11.1.48 SequenceGenerator Annotation
- `@TableGenerator` : 11.1.50 TableGenerator Annotation

Note: About method of generating IDs

For generating ID, specify the value of `javax.persistence.GenerationType` in strategy attribute of `@GeneratedValue` annotation. Values that can be specified are as follows:

- `TABLE`: Generate ID using persistence layer (DB) table.
- `SEQUENCE`: Generate ID using sequence object of persistence layer (DB).
- `IDENTITY`: Generate ID using identity column of persistence layer (DB).
- `AUTO`: Generate ID by selecting the most appropriate method in persistence layer (DB).

Generally it is recommended to explicitly specify the type to be used instead of using `AUTO`.

Adding parent-entity and related-entity

In order to add parent-entity and related-entity together, call the save method of Repository interface and manage the entity objects under `EntityManager`. Then create the related-entity objects and map them with parent-entity objects.

In order to use this method, `persist` needs to be included in the cascade operation of the related-entity.

Note: Behavior of entities in case of cascade operations

When the related-entity is specified under cascade operations, JPA operations for parent-entity (`persist`, `merge`, `remove`, `refresh`, `detach`) are linked with related-entity and then carried out.

Mapping with the operations of Repository interface of Spring Data JPA is as follows:

- save method : `persist` or `merge`
- delete method : `remove`

`refresh`, `detach` are not executed in default implementation of Spring Data JPA.

- Service

```
String itemCode = "ITM0000001";
int itemQuantity = 10;
String wayToPay = "card";

Order order = new Order("accepted");
order = orderRepository.save(order); // (1)
```

```
OrderItem orderItem = new OrderItem(order.getId(), itemCode,
        itemQuantity);
order.setOrderItems(Collections.singleton(orderItem));     // (2)
order.setOrderPay(new OrderPay(order.getId(), wayToPay)); // (2)
```

Sr. No.	Description
(1)	First, call the save method of Repository interface and bring the entity object under `EntityManager`. In the above example, save method is being called before setting the related-entity objects. This is because it is necessary to generate the ID (orderId) of parent-entity; this ID is used as a part of ID of the related entity.
(2)	Set the related-entity objects for the parent-entity object (that are to be managed under `EntityManager`). When committing the transaction, persist operation (INSERT) on parent-entity object is linked with the related-entity objects.

- Entity

```
@Entity
@Table(name = "t_order")
public class Order implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id
    @SequenceGenerator(name = "GEN_ORDER_ID", sequenceName = "s_order_id",
                       allocationSize = 1)
    @GeneratedValue(strategy = GenerationType.SEQUENCE,
                    generator = "GEN_ORDER_ID")
    private int id;

    @OneToMany(mappedBy = "order", cascade = CascadeType.ALL,  // (3)
               orphanRemoval = true)
    @OrderBy
    private Set<OrderItem> orderItems;

    @OneToOne(mappedBy = "order", cascade = CascadeType.ALL,
              orphanRemoval = true)
    private OrderPay orderPay;

    @ManyToOne
```

```
    @JoinColumn(name = "status_code")
    private OrderStatus status;

    // ...

    public Order(String statusCode) {
        this.status = new OrderStatus(statusCode);
    }

    // ...

}
```

Sr. No.	Description
(3)	Specify the cascade operation type (`javax.persistence.CascadeType`) in cascade attribute in corresponding annotation. In the above example, all operations are target of cascade to the related-entity. If there is no specific reason, it is recommended to consider all the operations for cascade.

Warning: Related-entity for which cascade attribute should not be specified

If the transaction entity is associated with code and master entities, cascade attribute should not be specified. In the above example, `OrderStatus` is the code entity; hence cascade attribute should not be specified in `@ManyToOne` annotation of `status` property of `Order`.

- Related-entity

```
@Entity
@Table(name = "t_order_item")
public class OrderItem implements Serializable {

    @EmbeddedId
    private OrderItemPK id;

    private int quantity;

    @ManyToOne
    @JoinColumn(name = "order_id", insertable = false, updatable = false)
    private Order order;

    // ...

    public OrderItem(Integer orderId, String itemCode, int quantity) {
        this.id = new OrderItemPK(orderId, itemCode);
```

```
        this.quantity = quantity;
    }

    // ...

}

@Entity
@Table(name = "t_order_pay")
public class OrderPay implements Serializable {

    @Id
    @Column(name = "order_id")
    private Integer orderId;

    @Column(name = "way_to_pay")
    private String wayToPay;

    @OneToOne
    @JoinColumn(name = "order_id")
    private Order order;

    // ...

    public OrderPay(int orderId, String wayToPay) {
        this.orderId = orderId;
        this.wayToPay = wayToPay;
    }

    // ...

}
```

Adding the related-entity

In order to add a related-entity, link the newly created related-entity object with the parent-entity object fetched through Repository interface.

For using this method, `persist` and `merge` should be included in the cascade operation of the related-entity.

```
String itemCode = "ITM0000003";
int quantity = 30;

Order order = orderRepository.findOne(orderId); // (1)
```

```
OrderItem orderItem = new OrderItem(order.getId(), itemCode, quantity);
order.getOrderItems().add(orderItem); // (2)

OrderPay orderPay = order.getOrderPay();
if (orderPay == null) {
    order.setOrderPay(new OrderPay(order.getId(), "cash")); // (3)
} else {
    orderPay.setWayToPay("cash");
}
```

Sr. No.	Description
(1)	Fetch the entity object using the Repository interface.
(2)	In case of entity with 1: N relationship, add the related-entity object to the collection fetched from the parent-entity object.
(3)	In case of entity with 1:1 relationship, set the related-entity object in the parent-entity object.

Adding the related-entity directly

When a related-entity object is to be added directly without linking it with the parent-entity object, save it using the Repository interface of related-entity.

```
String itemCode = "ITM0000003";
int quantity = 40;

OrderItem orderItem = new OrderItem(orderId, itemCode, quantity); // (1)

orderItemRepository.save(orderItem); // (2)
```

Sr. No.	Description
(1)	Create an object of related-entity.
(2)	Call the save method of Repository interface of the related-entity.

Note: Merits of saving the related-entity object directly

The number of objects created is less. When fetching the parent-entity object, related-entity objects which are not necessary for the processing may also get created.

Note: Demerits of saving the related-entity directly

When ID of the parent-entity is being used as a part of the related-entity ID, the ID should be set before calling the save method. If ID is set, merge method of `EntityManager` is called as per the default implementation of Spring Data JPA. Therefore, the entity with same ID is always fetched from persistence layer (DB).

Warning: Points to be noted at the time of using parent-entity object after the related-entity is added

When the related-entity is added using Repository save method of related-entity, it cannot be fetched through the parent-entity object since it is not linked with the parent-entity object.

To avoid this problem,

1. Related-entity object should not be added directly. It should first be linked with the parent-entity object, and then added.
2. Synchronization with persistence layer (DB) should be done before fetching the parent-entity, using saveAndFlush method.

In such a case, if there is no specific reason, objects of the related-entity should be added by the former method. In the latter method, the problem cannot be avoided if the parent-entity object is already the "managed" entity.

Updating entities

The example of updating entities is explained below.

How to update entities

In order to update the entity, set the changed value to the entity object fetched using the Repository interface method.

```
Order order = orderRepository.findOne(orderId); // (1)
order.setStatus(new OrderStatus("checking")); // (2)
```

Sr. No.	Description
(1)	Fetch the entity object using Repository interface method.
(2)	Update the state of entity object by calling setter method.

Note: Calling the save method of Repository

The entity object fetched using Repository interface method are managed under `EntityManager`. For the entity objects managed under `EntityManager`, just by changing the state of objects using setter method, the changes are reflected in persistence layer (DB) at the time of committing the transaction. Therefore, there is no need to explicitly call the save method of Repository interface.

However, save method needs to be called when the entity objects are not managed under `EntityManager`. For example, when entity object is created based on the request parameters sent from the screen.

Updating the related-entity

In order to update the related-entity, first fetch the related-entity from the parent entity which in turn can be fetched using Repository interface and then set the values to be updated in related-entity object.

For using this method, `merge` should be included in the cascade operation of related-entity.

```
Order order = orderRepository.findOne(orderId); // (1)

for (OrderItem orderItem : order.getOrderItems()) {
    int newQuantity = quantityMap.get(orderItem.getId().getItemCode());
    orderItem.setQuantity(newQuantity); // (2)
}

order.getOrderPay().setWayToPay("cash"); // (3)
```

Sr. No.	Description
(1)	Use the Repository interface method to fetch entity objects.
(2)	In case of entity with 1:N relationship, the state of related-entity object stored in the collection fetched from parent-entity object is to be updated by calling the setter method.
(3)	In case of entity with 1:1 relationship, the state of related-entity object fetched from parent-entity object is to be updated by calling the setter method.

Updating the related-entity directly

In order to update the related-entity directly without using the parent-entity, fetch the related-entity directly using its corresponding Repository interface and set the value to be changed.

```
int quantity = 43;

OrderItem orderItem = orderItemRepository.findOne(new OrderItemPK(
        orderId, itemCode)); // (1)

orderItem.setQuantity(quantity); // (2)
```

Sr. No.	Description
(1)	Call findOne method of Repository interface of the related-entity and fetch the related-entity object.
(2)	Update the state of related-entity object using setter method.

Note: Behavior at the time of using parent-entity after the related-entity is updated

When the related-entity is updated using save method of Repository for related-entity, the related-entity stored in the parent-entity object is also updated unlike the case wherein the related-entity is added. This is because the parent-entity stores the reference of same instance which is managed under `EntityManager`.

Updating by using query method

Use query method to update the entity of persistence layer (DB) directly.
For details, refer to "*Operating the entities of Persistence Layer directly*".

Deleting entities

Deleting parent-entity and related-entity

In order to delete parent-entity and related-entity together, call delete method of Repository interface.

For using this method, `remove` should be included in the cascade operation of the related-entity or the setting for deleting the related-entity should be enabled (orphanRemoval attribute should be set to `true`).

- Service

```
orderRepository.delete(orderId); // (1)
```

Sr. No.	Description
(1)	Specify ID or entity object and call delete method of Repository interface.

- Entity

```
@Entity
@Table(name = "t_order")
public class Order implements Serializable {

    // ...

    @OneToMany(mappedBy = "order",
               cascade = CascadeType.ALL, orphanRemoval = true) // (2)
    @OrderBy
    private Set<OrderItem> orderItems;

    // ...

}
```

Sr. No.	Description
(2)	Include `remove` in cascade operations and enable the setting to delete the related-entity (set orphanRemoval attribute of the associated annotation to `true`).

Note: Deleting related-entity

If you do not want to delete the related-entity object, perform settings such that `remove` is not included in cascade operation. Also, set orphanRemoval attribute to `false`.

Deleting the related-entity

In order to delete the related-entity, delete the related-entity object from the entity objects fetched through Repository interface.

For using this method, setting to delete the related-entity should be enabled (orphanRemoval attribute of the associated annotation should be set to `true`).

Note: Behavior when the setting to delete the related-entity is enabled

Behavior when the setting to delete the related-entity is enabled is as follows:

- In case of entity with 1:N relationship, if the related-entity object is deleted from the collection, it is also deleted from the persistence layer (DB) at the time of committing the transaction.
- In case of entity with 1:1 relationship, if the related-entity is set to `null`, it is also deleted from the persistence layer (DB) at the time of committing the transaction.

When the value of orphanRemoval attribute is set to `false` (which is also the default value), the related-entity is cleared from the memory; however persistence operation (UPDATE/DELETE) is not carried out for deleted related-entity object.

- Service

```
Order order = orderRepository.findOne(orderId); // (1)

// (2)
Set<OrderItem> orderItemsOfRemoveTarget = new LinkedHashSet<OrderItem>(); //␣
↪(3)
for (OrderItem orderItem : order.getOrderItems()) {
    String itemCode = orderItem.getId().getItemCode();
    if (quantityMap.containsKey(itemCode)) {
        int newQuantity = quantityMap.get(itemCode);
        orderItem.setQuantity(newQuantity);
    } else {
        orderItemsOfRemoveTarget.add(orderItem); // (4)
    }
}
order.getOrderItems().removeAll(orderItemsOfRemoveTarget); // (5)

order.setOrderPay(null); // (6)
```

Sr. No.	Description
(1)	Fetch the entity object using Repository interface.
(2)	Describe the example to delete the entity with 1:N relationship.
(3)	Create a collection for storing the related-entity object to be deleted.
(4)	Add the related-entity object to be deleted to the collection of (3).
(5)	Delete it from the collection fetched from the related-entity object to be deleted.
(6)	In case of entity with 1:1 relationship, set the property to that stores the related-entity object to be deleted to `null`.

- Entity

```
@Entity
@Table(name = "t_order")
public class Order implements Serializable {

    @Id
    @SequenceGenerator(name = "GEN_ORDER_ID", sequenceName = "s_order_id",␣
↪allocationSize = 1)
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "GEN_
↪ORDER_ID")
    private int id;

    @OneToMany(mappedBy = "order", cascade = CascadeType.ALL,
               orphanRemoval = true) // (7)
    @OrderBy
    private Set<OrderItem> orderItems;

    @OneToOne(mappedBy = "order", cascade = CascadeType.ALL,
              orphanRemoval = true) // (7)
    private OrderPay orderPay;

    @ManyToOne
```

```
    @JoinColumn(name = "status_code")
    private OrderStatus status;

    // ...

}
```

Sr. No.	Description
(7)	Enable the setting to delete the related-entity (set orphanRemoval attribute to `true`).

Note: Associated annotations that can be specified for orphanRemoval attribute

There are 2 associated annotations that can be specified for orphanRemoval attribute; `@OneToOne` and `@OneToMany`.

Deleting the related-entity directly

In order to delete the related-entity without using the parent-entity, call the delete method of Repository interface of related-entity.

```
int quantity = 43;

orderItemRepository.delete(new OrderItemPK(orderId, itemCode)); // (1)
```

Sr. No.	Description
(1)	Specify ID or entity object and call delete method of Repository interface of the related-entity.

Warning: Points to be noted while deleting the related-entity directly

When delete method of Repository for the related-entity is called, it should be noted that related-entity may not get deleted from persistence layer (DB) in some cases.

Such cases are as follows:

- findOne method is called in delete method; hence when the relation with parent-entity is `@OneToOne`, the parent-entity ends up getting managed under `EntityManager`. If the parent-entity ends up getting managed under `EntityManager`, the related-entity that was to be deleted using delete method may be loaded in the parent-entity object. Once it is loaded under the parent-entity object, it will not be deleted from persistence layer (DB).
- If the parent-entity object is fetched after calling the delete method, the related-entity which is deleted using delete method may be loaded in the parent-entity object. Once it is loaded under parent-entity object, it cannot be deleted from persistence layer (DB).

How to avoid this problem,

1. Instead of deleting the related-entity object directly, its association with the parent-entity should be deleted.

It may be possible to avoid this problem by reconsidering the associated annotation; however the best way to avoid this problem is not to delete the related-entity directly.

Deleting using query method

Use query method in order to directly delete the entity from persistence layer (DB).
For details, refer to "*Operating the entities of Persistence Layer directly*".

Warning: Handling of related-objects

When the entity is deleted directly from the persistence layer (DB) using query method, irrespective of whether or not the associated annotation is specified, the related-entity object is not deleted from the persistence layer.

`void deleteInBatch(Iterable<T> entities)` and `void deleteAllInBatch()` of Repository interface operate in the same way.

Escaping at the time of LIKE search

Escaping the values to be used as search criteria should be done for LIKE search.

Escaping for LIKE search can be done using `org.terasoluna.gfw.common.query.QueryEscapeUtils` class method of the common library.

For specifications of escaping in common library, refer to "*Escaping during LIKE search*" of "*Database Access (Common)*".

For the escaping method provided by common library, see the description below.

Usage method when type of matching is to be specified in query

When type of matching (Forward match, Backward Match, Partial Match) is to be specified as JPQL, use the method that performs only escaping.

- Repository

```
// (1) (2)
@Query("SELECT a FROM Article a WHERE"
        + " (a.title LIKE %:word% ESCAPE '~' OR a.overview LIKE %:word%␣
↪ESCAPE '~')")
Page<Article> findPageBy(@Param("word") String word, Pageable pageable);
```

Sr. No.	Description
(1)	Specify wildcard characters (`"%"` or `"_"`) for LIKE search in JPQL to be specified in `@Query` annotation. In the above example, the type of matching is set to partial match by specifying wildcard (`"%"`) before and after the argument `word`.
(2)	In case of escaping provided by the common library, `"~"` is being used as escape characters; hence specify `"ESCAPE '~'"` after LIKE clause.

Note: About wildcard character "_"

As described in [*Specifying the query using @Query annotation*], the wildcard character `"%"` can be use

directly in JPQL only if the `@Query` annotation is used. The wildcard character `"_"` cannot use directly in LIKE search of JPQL. It can be used in the following two ways.

1. Include wildcard character `"_"` in the bind variable.
2. Concatenate the wildcard character `"_"` with bind variable using database string concatenation function such as `CONCAT` .

- Service

```
@Transactional(readOnly = true)
public Page<Article> searchArticle(ArticleSearchCriteria criteria,
        Pageable pageable) {

    String escapedWord = QueryEscapeUtils.toLikeCondition(criteria.getWord());
↪ // (3)

    Page<Article> page = articleRepository.findPageBy(escapedWord, pageable);␣
↪// (4)

    return page;
}
```

Sr. No.	Description
(3)	When type of matching of LIKE search is to be specified in query, call `QueryEscapeUtils#toLikeCondition(String)` method and perform only escaping for LIKE search.
(4)	Pass the value escaped for LIKE search as the argument of query method.

Usage method when specifying type of matching in logic

When the type of matching (Forward match, Backward match, Partial match) is to be determined in logic, use the method that assigns wildcard to the escaped values.

- Repository

```
// (1)
@Query("SELECT a FROM Article a WHERE"
        + " (a.title LIKE :word ESCAPE '~' OR a.overview LIKE :word ESCAPE '~
↪')")
Page<Article> findPageBy(@Param("word") String word, Pageable pageable);
```

Sr. No.	Description
(1)	Do not specify wildcard for LIKE search in JPQL to be specified in `@Query` annotation.

- Service

```
@Transactional(readOnly = true)
public Page<Article> searchArticle(ArticleSearchCriteria criteria,
        Pageable pageable) {

    String word = QueryEscapeUtils
            .toContainingCondition(criteria.getWord()); // (2)

    Page<Article> page = articleRepository.findPageBy(word, pageable); // (3)

    return page;
}
```

Sr. No.	Description

(2)	When the type of matching is to be specified in logic, call any of the following methods and perform escaping for LIKE search and assign wildcard for LIKE search.
	`QueryEscapeUtils#toStartingWithCondition(String)`
	`QueryEscapeUtils#toEndingWithCondition(String)`
	`QueryEscapeUtils#toContainingCondition(String)`

(3)	Pass the value escaped for LIKE search + assigned with wildcard as an argument of query method.

JOIN FETCH

JOIN FETCH is a mechanism to reduce the number of queries generated for fetching entities, by joining and fetching the related-entities in batch.

When fetching the entities by executing any query, make sure to perform JOIN FETCH for properties where fetch attribute of associated annotation is `EAGER`.

If JOIN FETCH is not performed, it may impact performance since queries to fetch the related-entities will be generated separately.

For properties where fetch attribute of related-annotation is `LAZY`, consider the frequency of use and determine whether or not to perform JOIN FETCH.

JOIN FETCH should be performed in case of related-entities that are always referenced. However, if the cases wherein related-entities will be referenced are only few, then it is better to fetch the entities by executing the queries at the time of initial access instead of performing JOIN FETCH.

- Repository

```
@Query("SELECT a FROM Article a"
        + " INNER JOIN FETCH a.articleClass"   // (1)
        + " WHERE a.publishedDate = :publishedDate"
        + " ORDER BY a.articleId DESC")
```

```
List<Article> findAllByPublishedDate(
        @Param("publishedDate") Date publishedDate);
```

Sr. No.	Description
(1)	Specify in " [LEFT [OUTER] | INNER] JOIN FETCH Properties to be joined". See the pattern below. 1. LEFT JOIN FETCH Entity is obtained even if the related-entity does not exist. SQL will be "left outer join RelatedTable r on m.fkColumn = r.fkColumn". 2. LEFT OUTER JOIN FETCH Same as 1. 3. INNER JOIN FETCH Entity is not obtained if related-entity does not exist. SQL will be "inner join RelatedTable r on m.fkColumn = r.fkColumn". 4. JOIN FETCH Same as 3. In the above example, articleClass property of Article class is specified as the JOIN FETCH target.

Note: JOIN FETCH is used as one of the solutions for N+1 problem. For N+1 problem, refer to "*How to resolve N+1*".

6.3.3 How to extend

How to add custom method

Spring Data provides mechanism to add any custom methods for the Repository interface.

Sr. No.	How to Add	Use case
1.	*Adding individual custom method to entity specific Repository interface*	Use when it is necessary to execute a query that cannot be expressed using the mechanism of query method of Spring Data. Add methods using this method when executing dynamic queries.
2.	*Adding the custom methods to all Repository interfaces in batch*	Use when it is necessary to execute a generic query that can be used in all Repository interfaces. Add methods using this method when executing project specific generic queries. If it is necessary to change the behavior of default implementation (`SimpleJpaRepository`) of Spring Data JPA, override the methods using this mechanism.

Adding individual custom method to entity specific Repository interface

See the description below for adding individual custom method to entity specific Repository interface.

- Entity specific custom Repository interface

```
// (1)
public interface OrderRepositoryCustom {

    // (2)
    Page<Order> findByCriteria(OrderCriteria criteria, Pageable pageable);

}
```

Sr. No.	Description
(1)	Create a custom interface for defining custom methods. There are no specific restrictions for naming the interface; however it is recommended that you set it to entity specific Repository interface name + "`Custom`".
(2)	Define custom methods.

- Entity specific custom Repository class

```
// (3)
public class OrderRepositoryImpl implements OrderRepositoryCustom {

    @PersistenceContext
    EntityManager entityManager; // (4)

    // (5)
    public Page<Order> findByCriteria(OrderCriteria criteria, Pageable␣
↪pageable) {
        // ...
        return new PageImpl<Order>(orders, pageable, totalCount);
    }

}
```

Sr. No.	Description
(3)	Create implementation class of custom interface. The class name should be, entity specific Repository interface name + "`Impl`".
(4)	Using `@javax.persistence.PersistenceContext` annotation, inject `javax.persistence.EntityManager` which is necessary for executing the query.
(5)	Implement the method defined in custom interface.

- Entity specific Repository interface

```
public interface OrderRepository extends JpaRepository<Order, Integer>,
        OrderRepositoryCustom { // (6)
```

```
    // ...
}
```

Sr. No.	Description
(6)	Inherit custom interface in entity specific Repository interface. Methods of implementation class of custom interface are called at runtime by inheriting Repository interface.

- Service(Caller)

```
public Page<Order> search(OrderCriteria criteria, Pageable pageable) {
    return orderRepository.findByCriteria(criteria, pageable); // (7)
}
```

Sr. No.	Description
(7)	Methods of entity specific Repository interface can be called similar to other methods. In the above example, if `OrderRepository#findByCriteria(OrderCriteria,Pageable)` is called, `OrderRepositoryImpl#findByCriteria(OrderCriteria,Pageable)` is executed.

Adding the custom methods to all Repository interfaces in batch

See the description below for adding the custom methods to all Repository interfaces in batch.

The method added in the example given below checks the version of the fetched entity and throws an optimistic exclusion error in case of a mismatch.

- Common Repository interface

```
// (1)
@NoRepositoryBean // (2)
public interface MyProjectRepository<T, ID extends Serializable> extends
        JpaRepository<T, ID> {

    // (3)
    T findOneWithValidVersion(ID id, Integer version);
```

```
}
```

Sr. No.	Description
(1)	Create a common Repository interface for defining the methods to be added. In the above example, common Repository interface is being created by inheriting `JpaRepository` of Spring Data.
(2)	Annotation to prevent common Repository interface implementation class (in this example, `MyProjectRepositoryImpl`) from being automatically registered as Repository Bean. When common Repository interface implementation class is to be stored under the package specified in base-package attribute of <jpa:repositories> element, there will be an error at start-up unless this annotation is specified.
(3)	Define the method to be added. In the example, method to check the version of fetched entity is being added.

- Common Repository interface implementation class

```
// (6)
public class MyProjectRepositoryImpl<T, ID extends Serializable>
        extends SimpleJpaRepository<T, ID>
        implements MyProjectRepository<T, ID> {

    private JpaEntityInformation<T, ID> entityInformation;
    private EntityManager entityManager;

    // (7)
    public MyProjectRepositoryImpl(
            JpaEntityInformation<T, ID> entityInformation,
            EntityManager entityManager) {
        super(entityInformation, entityManager);

        this.entityInformation = entityInformation; // (8)
        this.entityManager = entityManager; // (8)

        try {
            return domainClass.getMethod("getVersion");
        } catch (NoSuchMethodException | SecurityException e) {
            return null;
        }
```

```
    }

    // (9)
    public T findOneWithValidVersion(ID id, Integer version) {
        if (versionMethod == null) {
            throw new UnsupportedOperationException(
                    String.format(
                            "Does not found version field in entity class.␣
↪class is '%s'.",
                            entityInformation.getJavaType().getName()));
        }

        T entity = findOne(id);

        if (entity != null && version != null) {
            Integer currentVersion;
            try {
                currentVersion = (Integer) versionMethod.invoke(entity);
            } catch (IllegalAccessException | IllegalArgumentException
                    | InvocationTargetException e) {
                throw new IllegalStateException(e);
            }
            if (!version.equals(currentVersion)) {
                throw new ObjectOptimisticLockingFailureException(
                        entityInformation.getJavaType().getName(), id);
            }
        }

        return entity;
    }

}
```

Sr. No.	Description
(6)	Create an implementation class of common Repository interface (in this example, `MyProjectRepository`). In the above example, implementation class is being created by inheriting ‘‘ SimpleJpaRepository‘‘ of Spring Data.
(7)	Create a constructor that takes `org.springframework.data.jpa.repository.support.JpaEntityInformation` and `javax.persistence.EntityManager` as arguments.
(8)	Store `JpaEntityInformation` and `EntityManager` as fields.
(9)	Implement the method defined in common Repository interface.

Note: Changing default implementation

If the behavior of default implementation (`SimpleJpaRepository`) is to be changed, the method for which behavior is to be changed can be overridden in this class.

- Factory class for creating instances of common Repository interface implementation class

```
// (10)
private static class MyProjectRepositoryFactory<T, ID extends Serializable>
        extends JpaRepositoryFactory {

    // (11)
    public MyProjectRepositoryFactory(EntityManager entityManager) {
        super(entityManager);
    }

    // (12)
    protected JpaRepository<T, ID> getTargetRepository(
            RepositoryMetadata metadata, EntityManager entityManager) {

        @SuppressWarnings("unchecked")
        JpaEntityInformation<T, ID> entityInformation =␣
↪getEntityInformation((Class<T>) metadata
                .getDomainType());
```

```
        MyProjectRepositoryImpl<T, ID> repositoryImpl = new␣
↪MyProjectRepositoryImpl<T, ID>(
                entityInformation, entityManager);
        repositoryImpl
                .setLockMetadataProvider(LockModeRepositoryPostProcessor.
↪INSTANCE
                        .getLockMetadataProvider()); // (13)

        return repositoryImpl;
    }

    // (14)
    protected Class<?> getRepositoryBaseClass(RepositoryMetadata metadata) {
        return MyProjectRepository.class;
    }
}
```

Sr. No.	Description
(10)	Create a Factory class for creating instances of common Repository interface implementation class. In the example, `org.springframework.data.jpa.repository.support.JpaRepositoryFactory` is being inherited in order to restrict the implementation of Factory class.
(11)	Define a constructor that takes `EntityManager` as argument and call constructor of parent class.
(12)	Override the method creating instances of implementation class. Create and return instances of the created implementation class (in this example, `MyProjectRepositoryImpl`).
(13)	This code is required to specify the lock mode using `@Lock` annotation for `findOne` method and `findAll` methods of `SimpleJpaRepository` class of Spring Data JPA. Make sure to implement this piece of logic, since it is implemented in the method of `JpaRepositoryFactory` which is being overridden. This process is required when the lock mode is to be specified using `findOne` and `findAll` methods amongst the methods to be extended.
(14)	Override the method that returns base interface of entity specific Repository interface, and return the created interface (in this example, `MyProjectRepository`) type.

- FactoryBean for creating Factory class instances

```
// (15)
public class MyProjectRepositoryFactoryBean<R extends JpaRepository<T, ID>, T,
↪ ID extends Serializable>
        extends JpaRepositoryFactoryBean<R, T, ID> {

    // (16)
    protected RepositoryFactorySupport createRepositoryFactory(
            EntityManager entityManager) {
        return new MyProjectRepositoryFactory<T, ID>(entityManager);
    }
}
```

Sr. No.	Description
(15)	Create FactoryBean class for creating Factory class instances. In the example, `org.springframework.data.jpa.repository.support.JpaRepositoryFactory` is being inherited in order to restrict the implementation of FactoryBean class.
(16)	Override the method used for creating the Factory class instances. Then, create and return instances of the created Factory class.

- Entity specific Repository interface

```
public interface OrderRepository extends MyProjectRepository<Order, Integer>
↪{ // (17)
    // ...
}
```

Sr. No.	Description
(17)	Inherit all entity specific Repository interfaces from the common interface (In this example: `MyProjectRepository`).

- `xxx-infra.xml`

```
<jpa:repositories base-package="x.y.z.domain.repository"
    factory-class="x.y.z.domain.repository.MyProjectRepositoryFactoryBean" />
↪<!-- (18) -->
```

Sr. No.	Description
(18)	Specify class name of the created FactoryBean class (in the example, `MyProjectRepositoryFactoryBean`) in factory-class attribute of <jpa:repositories> element.

- Service(Caller)

```
public Order updateOrder(Order chngedOrder, Integer version) {

    Order order = orderRepository.findOneWithValidVersion(chngedOrder.getId(),
↪ version); // (19)
```

```
    // ....

    return order;
}
```

Sr. No.	Description
(19)	The methods of entity specific Repository interface can be called similar to other methods. In the above example, when `OrderRepository#findOneWithValidVersion(Integer,Integer)` is called, `MyProjectRepositoryImpl#findOneWithValidVersion(Integer,Integer)` is executed.

Storing query fetch results in objects other than entity

Query fetch results can be mapped to other objects than entities. Use this method when records stored in persistence layer (DB) are to be handled as objects (JavaBean) other than entity.

Note: Assumed cases

1. When the aggregated information is to be fetched using Aggregate function in query, it is not possible to map the aggregation result to entity; hence it should be mapped with different objects.
2. In order to refer to only a part of information in a huge entity, or of a related-entity with complex nesting, there may be cases wherein it is desirable to map the results with JavaBean containing only the necessary properties. This is because processing performance may get hampered due to the fact that mapping is carried out for items which are not needed in application processing or fetching of unnecessary information during the processing leading to memory exhaustion. It may be obtained as entity if there is no significant impact on processing performance.

See the example below.

- JavaBean

```
// (1)
public class OrderSummary implements Serializable {

    private Integer id;
    private Long totalPrice;

    // ...

    public OrderSummary(Integer id, Long totalPrice) { // (2)
        super();
        this.id = id;
        this.totalPrice = totalPrice;
    }

    // ...

}
```

Sr. No.	Description
(1)	Create JavaBean for storing query results.
(2)	Create a constructor for generating objects using query execution results.

- Repository interface

```
// (3)
@Query("SELECT NEW x.y.z.domain.model.OrderSummary(o.id, SUM(i.price*oi.
↪quantity))"
        + " FROM Order o LEFT JOIN o.orderItems oi LEFT JOIN oi.item i"
        + " GROUP BY o.id ORDER BY o.id DESC")
List<OrderSummary> findOrderSummaries();
```

Sr. No.	Description
(3)	Specify the constructor created in (2). Specify the constructor by FQCN after “NEW” keyword.

Setting Audit properties

This section introduces a mechanism to set values to Audit properties (Created By, Created Date-Time, Last Modified By, Last Modified Date-Time) of persistence layer and method to apply the same.

Spring Data JPA provides a mechanism to set values to Audit properties for newly created entities as well as modified entities. This mechanism facilitates separation of logic of setting values to Audit properties from application code such as Service etc.

Note: Reasons to separate the logic to set values to Audit properties from application code

1. Setting the values to Audit properties is generally not an application requirement, but is essential as per the audit requirements of data. This logic does not essentially belong to application code such as Service etc.; hence it is better to separate it from application code.

2. As per JPA specifications, only when the values of the entities fetched from persistence layer change, the changes will be reflected in the persistence layer (by executing the SQL), thereby avoiding the unnecessary access to persistence layer. If the values are set unconditionally in Audit properties in the application code such as Service, even if only Audit properties change, the changes are reflected in persistence layer making the effective JPA functionality ineffective. This problem can be avoided if the values are set in Audit properties only when if they are changed; however it cannot be recommended as it makes the application code complex.

Note: When Audit column of the persistence layer needs to be updated irrespective of the change in values

If updating the Audit columns (Last Modified By, Last Modified Date-Time) even if there is no change in values is a part of application specifications, then it becomes necessary to set Audit properties in application code such as Service etc.

However, in such a case, the data modeling or application specifications are likely to be incorrect; hence it is better to revise these specifications.

See the example below.

- Entity class

```
public class XxxEntity implements Serializable {
    private static final long serialVersionUID = 1L;

    // ...

    @Column(name = "created_by")
    @CreatedBy // (1)
    private String createdBy;

    @Column(name = "created_date")
    @CreatedDate // (2)
    @Type(type = "org.jadira.usertype.dateandtime.joda.PersistentDateTime") //
 (3)
    private DateTime createdDate; // (4)

    @Column(name = "last_modified_by")
    @LastModifiedBy // (5)
    private String lastModifiedBy;

    @Column(name = "last_modified_date")
    @LastModifiedDate // (6)
    @Type(type = "org.jadira.usertype.dateandtime.joda.PersistentDateTime") //
 (3)
    private DateTime lastModifiedDate; // (4)

    // ...

}
```

Sr. No.	Description
(1)	Assign `@org.springframework.data.annotation.CreatedBy` annotation to the field that stores “Created by”.
(2)	Assign `@org.springframework.data.annotation.CreatedDate` annotation to the field that stores “Created date-time”.
(3)	When `org.joda.time.DateTime` type is to be used, assign `@org.hibernate.annotations.Type` annotation to the field in order to handle in Hibernate. type attribute is fixed to `"org.jadira.usertype.dateandtime.joda.PersistentDateTime"`. This is applicable for “Last modified date-time” field also.
(4)	Field type that stores the “Created date-time” supports `org.joda.time.DateTime`, `java.util.Date`, `java.util.Calendar`, `java.lang.Long`, `long` types as well Date and Time APIs introduced in Java 8. This is applicable for “Last modified date” field also.
(5)	Assign `@org.springframework.data.annotation.LastModifiedBy` annotation to the field type that stores “Last modified by”.
(6)	Assign `@org.springframework.data.annotation.LastModifiedDate` annotation to the field that stores the “Last modified date-time”.

Warning: `@Type` annotation is a Hibernate specific annotation and not a standard JPA annotation.

- AuditorAware interface implementation class

```
// (7)
@Component // (8)
public class SpringSecurityAuditorAware implements AuditorAware<String> {
```

```
    // (9)
    public String getCurrentAuditor() {
        Authentication authentication = SecurityContextHolder.getContext()
                .getAuthentication();
        if (authentication == null || !authentication.isAuthenticated()) {
            return null;
        }
        return ((UserDetails) authentication.getPrincipal()).getUsername();
    }

}
```

Sr. No.	Description
(7)	Create `org.springframework.data.domain.AuditorAware` interface implementation class. `AuditorAware` interface is used for resolving the entity operator (Created by or Last Modified by). This class should be created for each project.
(8)	By assigning `@Component` annotation, this class comes under the target of component-scan. Without assigning `@Component` annotation, it is also ok to define a bean of this class in bean definition file.
(9)	Return the object to be set in the properties of entity operator (Created by or Last Modified by). In the above example, login user name (string) authenticated by SpringSecurity is returned as entity operator.

- Object/relational mapping file(`orm.xml`)

```
<?xml version="1.0" encoding="UTF-8"?>

<entity-mappings xmlns="http://java.sun.com/xml/ns/persistence/orm"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/persistence/orm
    http://java.sun.com/xml/ns/persistence/orm_2_0.xsd"
    version="2.0">

    <persistence-unit-metadata>
        <persistence-unit-defaults>
```

```
            <entity-listeners>
                <entity-listener
                    class="org.springframework.data.jpa.domain.support.
↪AuditingEntityListener" /> <!-- (10) -->
            </entity-listeners>
        </persistence-unit-defaults>
    </persistence-unit-metadata>

</entity-mappings>
```

Sr. No.	Description
(10)	Specify `org.springframework.data.jpa.domain.support.AuditingEntityListener` class as Entity Listener. Values are set in Audit properties of the methods implemented in this class.

- `infra.xml`

```
<jpa:auditing auditor-aware-ref="springSecurityAuditorAware" /> <!-- (11) -->
```

Sr. No.	Description
(11)	Specify bean of class for resolving entity operator created in (7) in auditor-aware-ref attribute of <jpa:auditing> element. In the above example, component-scan is performed for `SpringSecurityAuditorAware` implementation class; hence bean name `"springSecurityAuditorAware"` is specified.

As default implementation, the values of `java.util.Calendar` instance returned by `getNow()` method of `org.springframework.data.auditing.CurrentDateTimeProvider` are used for the values set in the fields with `@CreatedDate` and `@LastModifiedDate` annotations.

Extended example of changing the method to create the values to be used is shown below.

```
// (1)
@Component // (2)
public class AuditDateTimeProvider implements DateTimeProvider {

    @Inject
```

```
    JodaTimeDateFactory dateFactory;

    // (3)
    @Override
    public Calendar getNow() {
        DateTime currentDateTime = dateFactory.newDateTime();
        return currentDateTime.toGregorianCalendar();
    }

}
```

Sr. No.	Description
(1)	Create `org.springframework.data.auditing.DateTimeProvider` interface implementation class.
(2)	Assign `@Component` annotation to perform component-scan. Bean can be defined in Bean definition file without assigning `@Component` annotation.
(3)	Return the instances to be set in the properties of entity operation date-time (Created Date-Time or Last Modified Date-Time). In the above example, the instances fetched from `org.terasoluna.gfw.common.date.jodatime.JodaTimeDateFactory` of common library are returned as Operation Date-Time. For details on `JodaTimeDateFactory`, refer to "*System Date*".

- `infra.xml`

```
<jpa:auditing
    auditor-aware-ref="springSecurityAuditorAware"
    date-time-provider-ref="auditDateTimeProvider" /> <!-- (4) -->
```

Sr. No.	Description
(4)	In date-time-provider-ref attribute of <jpa:auditing> element, specify the bean of class to return the value set in entity operation date-time created in (1). In the above example, component-scan is performed for `AuditDateTimeProvider` implementation class; hence bean name `"auditDateTimeProvider"` is specified.

Adding common conditions to JPQL to fetch entities from persistence layer

This section introduces a mechanism to add common conditions to JPQL for fetching the entities from persistence layer (DB). This is an extended Hibernate function and not included in standard JPA specifications.

Note: Assumed cases

As per data monitoring and data storage period requirements, when an entity is to be deleted, the application should be designed such that it will delete the record logically (UPDATE of Logical Delete flag) and will not delete (DELETE) the record physically. In case of such applications, the records deleted logically need to be uniformly excluded from the search target; hence rather than writing the condition to exclude logically deleted records in each and every query, it is better to specify it as a common condition.

Warning: Applicable scope

It should be noted that the specified conditions will be added for all queries executed to operate the entities for which this mechanism is used. This mechanism cannot be used even if there is a single Query for which conditions are not to be added.

Adding common conditions in JPQL to fetch entities

The method to add common conditions for JPQL which is executed at the time of calling Repository interface methods is as follows:

- Entity

```
@Entity
@Table(name = "t_order")
@Where(clause = "is_logical_delete = false") // (1)
public class Order implements Serializable {
    // ...
    @Id
    private Integer id;
    // ...
}
```

- SQL executed at the time of calling the findOne method of Repository interface

```
SELECT
        -- ....
    FROM
        t_order order0_
   WHERE
        order0_.id = 1
        AND (
            order0_.is_logical_delete = false -- (2)
        );
```

Sr. No.	Description
(1)	Assign `@org.hibernate.annotations.Where` annotation as entity class annotation and specify the common conditions in clause attribute. The WHERE clause should be specified in SQL instead of JPQL i.e. it is necessary to specify the column name instead of the property name of Java object.
(2)	The condition specified with `@Where` annotation is added.

Note: About Dialect extension

If SQL specific keywords are specified in `@Where` annotation, Hibernate may recognize SQL specific keywords as a common string value and as a result expected SQL may not get generated. It is necessary to extend the `Dialect` in case of SQL specific keywords are used in `@Where` annotation.

- Extending `Dialect` to register standard keywords such as `true`, `false` and `unknown`.

```
package com.example.infra.hibernate;

public class ExtendedPostgreSQL9Dialect extends PostgreSQL9Dialect { // (1)
    public ExtendedPostgreSQL9Dialect() {
        super();
        // (2)
        registerKeyword("true");
        registerKeyword("false");
        registerKeyword("unknown");
    }
}
```

Sr. No.	Description
(1)	By default Hiberanate-4.3 may not correctly process some of the SQL keywords. The BOOLEAN type keywords such as `true`, `false` and `unknown` are not registered in PostgreSQL dialect `org.hibernate.dialect.PostgreSQL9Dialect`. Therefore such keywords are recognized as a common string value and as a result incorrect SQL may get generated. It is necessary to extend `org.hibernate.dialect.Dialect` dialect in order to register such keywords.
(2)	Register the SQL keywords that are likely to be used in `@Where` annotation.

- Settings of extended Dialect

```
<bean id="jpaVendorAdapter"
    class="org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter">
    <property name="databasePlatform" value="com.example.infra.hibernate.
↪ExtendedPostgreSQL9Dialect"/> // (3)
    // ...
</bean>
```

Sr. No.	Description
(3)	The extended `Dialect` is set as the value of `databasePlatform` property in `JpaVendorAdapter` of `EntityManager`.

Note: Class that can be specified

`@Where` annotation is valid only in the class with `@Entity`. i.e. it is not assigned to SQL even if it is assigned to the base entity class with `@javax.persistence.MappedSuperclass`.

Warning: `@Where` annotation is a Hibernate specific annotation and not the standard JPA annotation.

Adding common conditions to JPQL to fetch the related-entities

The method for adding common conditions for JPQL is shown below. JPQL is used for fetching the related-entities of the parent-entity which is fetched by calling Repository interface methods.

- Entity

```
@Entity
@Table(name = "t_order")
@Where(clause = "is_logical_delete = false")
public class Order implements Serializable {
    // ...
    @Id
    private Integer id;

    @OneToMany(mappedBy = "order", cascade = CascadeType.ALL, orphanRemoval =
↪true)
    @OrderBy
    @Where(clause="is_logical_delete = false") // (1)
    private Set<OrderItem> orderItems;
    // ...

}
```

- SQL (Lazy Load) generated when accessing the related-entity

```
SELECT
        -- ...
    FROM
        t_order_item orderitems0_
    WHERE
        (orderitems0.is_logical_delete = false) -- (2)
        AND orderitems0_.order_id = 1
    ORDER BY
        orderitems0_.item_code ASC
        ,orderitems0_.order_id ASC;
```

- SQL(Eager Load/Join Fetch) generated at the time of fetching parent-entity and its related-entity simultaneously

```
SELECT
    -- ...
    FROM
        t_order order0_
            LEFT OUTER JOIN t_order_item orderitems1_
                ON order0_.id = orderitems1_.order_id
                AND (orderitems1_.is_logical_delete = false) -- (2)
    WHERE
        order0_.id = 1
```

```
        AND (
            order0_.is_logical_delete = false
        )
    ORDER BY
        orderitems1_.item_code ASC
        ,orderitems1_.order_id ASC;
```

Sr. No.	Description
(1)	Assign `@org.hibernate.annotations.Where` annotation as a related-entity field allocation and specify common conditions in clause attribute. The WHERE clause should be specified in SQL format instead of JPQL i.e. it is necessary to specify the column name instead of the property name of Java object.
(2)	The condition specified with `@Where` annotation is added.

Note: Types of related-entities that can be specified

Adding common conditions to SQL generated at the time fetching the related-entities is possible only for the entities having `@OneToMany` and `@ManyToMany` relationship.

Warning: `@Where` annotation is a Hibernate specific annotation and not the standard JPA annotation.

How to use multiple PersistenceUnits

Todo

TBD

Following topic will be added later.

- Examples for using multiple PersistenceUnits

How to use Native query

Todo

TBD

Following topic will be added later.

- Examples of query that uses Native query

6.4 Exclusive Control

6.4.1 Overview

Exclusive control is a process to maintain data consistency when same data is to be updated simultaneously by multiple transactions.

Exclusive control should be performed when same data is likely to get updated simultaneously by multiple transactions. These transactions are not necessarily restricted to database transactions only; they also include long transactions.

Note: Long transactions

Long transactions are the transactions where data fetch and data update are performed as separate database transactions.

To illustrate a specific example, the transactions are observed in an application where the fetched data is displayed on the Edit screen and the value edited on the screen is updated in database.

This chapter explains about exclusive control for the data stored in the database.

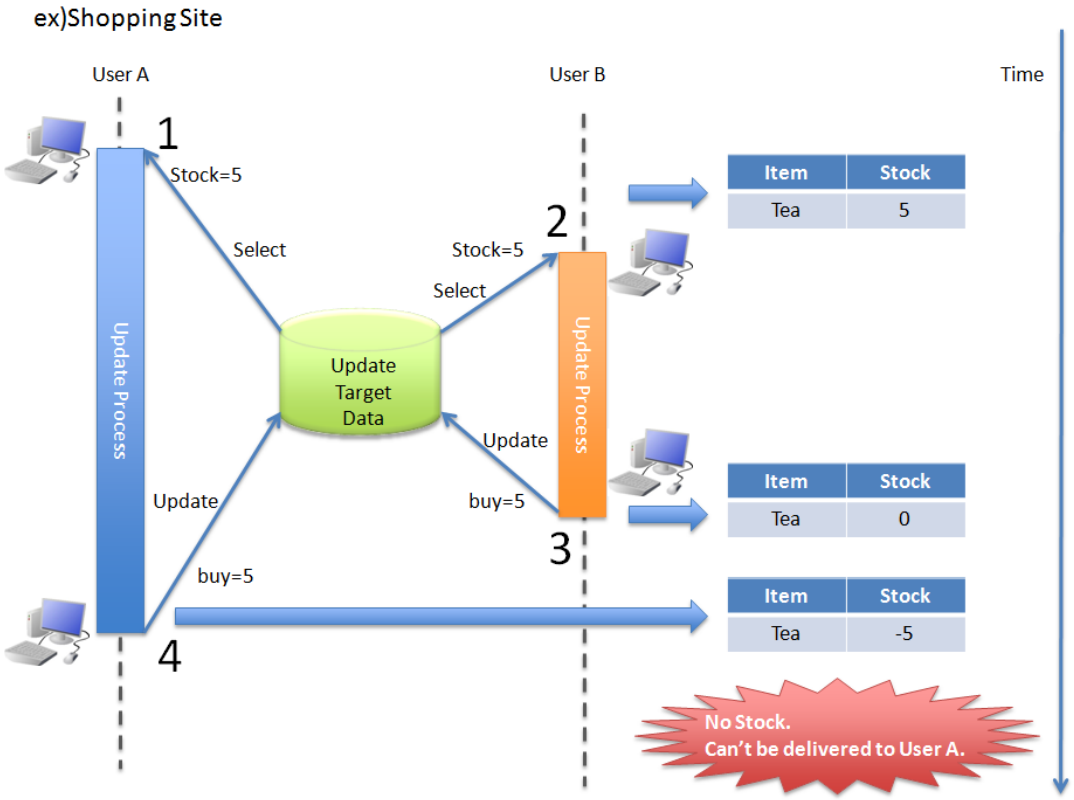
However, it should also be performed in a similar way for the data stored in data-store apart from database (such as memory, files etc.).

Necessity of exclusive control

To understand why exclusive control is necessary, let us first see the issues that occur in the absence of exclusive control using the examples given below.

Problem 1

See the example below of receiving an order for Tea from users on a shopping site.



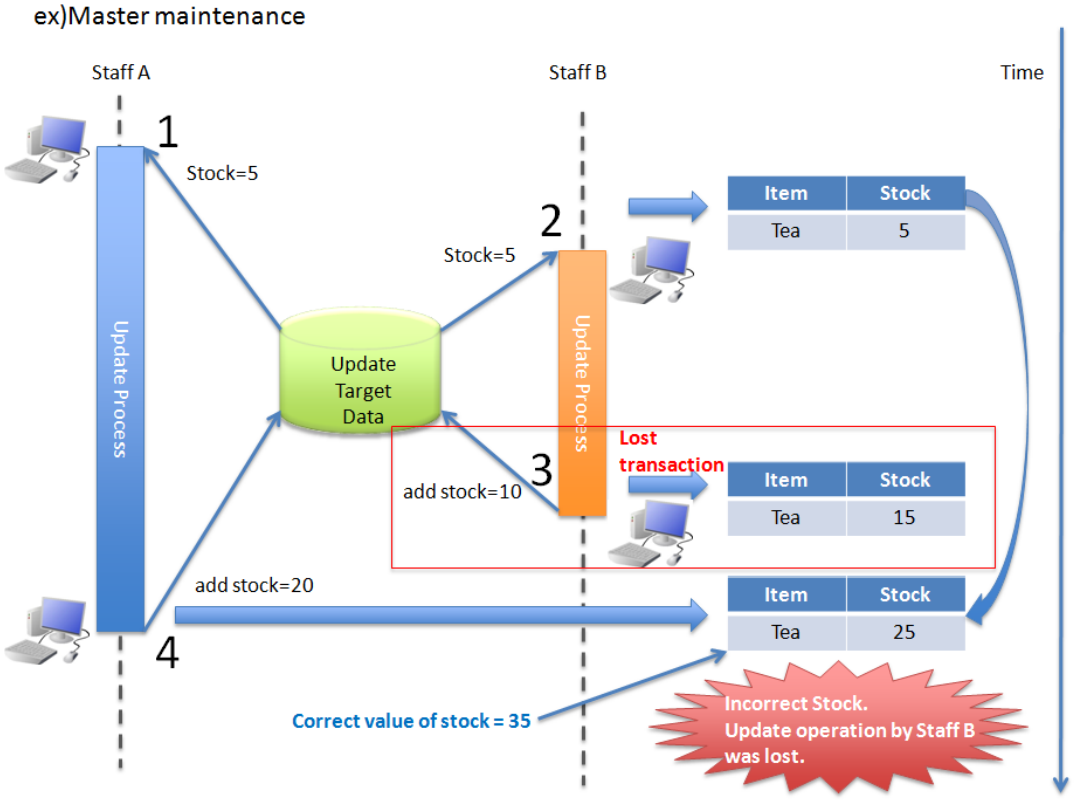
Sr. No.	UserA	UserB	Description
1.	○	-	User A confirms that the quantity of Tea on Product screen is 5nos.
2.	-	○	User B confirms that the quantity of Tea on Product screen is 5nos.
3.	-	○	User B orders 5nos. of Tea. Stock of Tea in the DB reduces by 5 and becomes 0.
4.	○	-	User A orders 5nos. of Tea. Stock of Tea in the DB reduces by 5 and becomes -5.

Order from User A is accepted however an apology is conveyed due to non-availability of actual stock.

The stock quantity of Tea stored in the table also shows a value (minus value) different from actual stock quantity of Tea.

Problem 2

See the example below wherein the staff that manages the stock quantity of Tea on shopping site, displays the stock quantity of Tea, calculates the added stock quantity at Client side and updates the stock quantity of Tea.

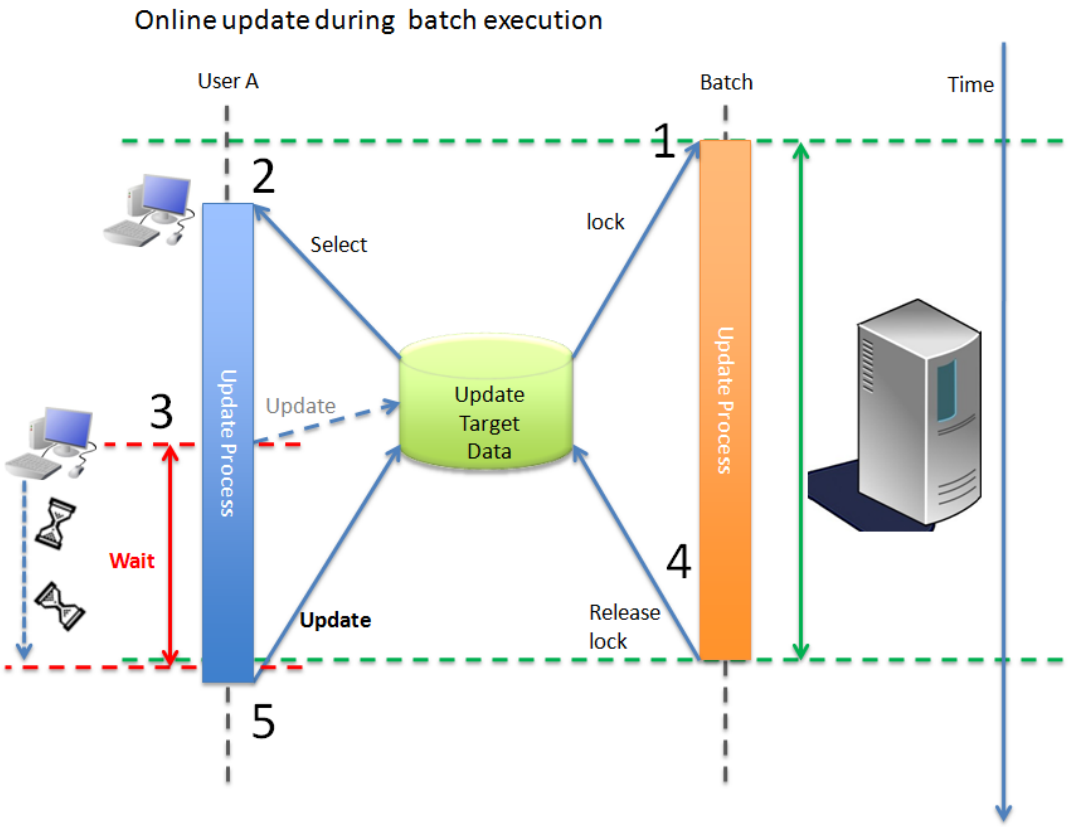


Sr. No.	UserA	UserB	Description
1.	○	-	Staff A confirms that the quantity of Tea is 5nos.
2.	-	○	Staff B confirms that the quantity of Tea is 5nos.
3.	-	○	Staff B adds stock of 10nos. of Tea and updates the stock quantity on Client side as 5+10=15.
4.	○	-	Staff A adds stock of 20nos. of Tea and updates the stock quantity on Client side as 5+20=25.

The stock of tea 10nos. added in process 3 is lost and there is a mismatch in actual stock quantity (35nos.).

Problem 3

See the example below wherein the data locked by batch processing is updated by online processing.



Sr. No.	UserA	Batch	Description
1.	-	○	Batch locks the table row (temporarily all rows) to be updated and does not allow the update by other processes.
2.	○	-	User A searches the updated information. Since batch is not committed at this point, the information prior to batch update can be fetched.
3.	○	-	User A requests for update and waits as it is locked in batch.
4.	-	○	Batch terminates the process to release the lock.
5.	○	-	The update process for which user A was waiting can now be executed.

User A executes update process after waiting for the batch to release the lock. However, data fetched by User A prior to waiting is the data before batch update and batch processing may overwrite the data with the updated data.

Batch takes a longer time as compared to online processing and user has to wait longer.

Exclusive control according to the isolation level of transaction

The easiest way to resolve all the 3 problems given in *Necessity of exclusive control* is to sequentially execute the database processes.

When the processes are sequentially executed, there is no effect on transactions.

However, when the processes are sequentially executed, the number of transactions that can be executed in a unit of time shows reduction resulting in deterioration in the performance.

In ANSI/ISO SQL standards, the guidelines indicating the isolation level (extent of impact by each transaction) are defined. The 4 isolation levels of transaction are given below. Events that occur at each isolation level are explained below.

Sr. No.	Isolation levels	DIRTY READ	NON-REPEATABLE READ	PHANTOM READ
1.	READ UNCOMMITTED	Yes	Yes	Yes
2.	READ COMMITTED	No	Yes	Yes
3.	REPEATABLE READ	No	No	Yes
4.	SERIALIZABLE	No	No	No

Tip: DIRTY READ

Dirty Read occurs when the data written by uncommitted transaction is read by other transaction.

Tip: NON-REPEATABLE READ

When the same record is likely to be read twice in the same transaction, if other transaction is committed during

the first read and second read, the details read at first time and those at second time may differ. The multiple data readings may vary based on commit timing of other transaction.

Tip: PHANTOM READ

In Phantom Read, when a same record is being read twice in the same transaction, if other transaction adds or deletes the record, it causes difference in the number of records (details) fetched during the first read and second read.

The isolation level defined in above table gets higher as we go down.

If the isolation level is high, the data can be protected; however it increases the locking overhead resulting in deterioration of the performance.

It is not desirable to select SERIALIZABLE unless the access frequency is fairly low.

This is because all the data is accessed sequentially one by one including SELECT.

The relationship between level of isolation and degree of concurrency between transactions is a Trade-off relationship.

In other words, high isolation level leads to reduction in concurrency and vice versa.

Thus, it is necessary to balance the level of isolation and degree of concurrency of transactions in accordance with application requirements.

The supported isolation level differs depending on the database to be used, it is necessary to understand the characteristics of the database to be used.

Isolation levels supported by each database and their default values are shown below.

Sr. No.	Database	READ UN-COMMITTED	READ COMMITTED	REPEATABLE READ	SERIALIZABLE
1.	Oracle	×	○ (default)	×	○
2.	PostgreSQL	×	○ (default)	×	○
3.	DB2	○	○ (default)	○	○
4.	MySQL InnoDB	○	○	○ (default)	○

When a balance is to be maintained between isolation and concurrency while maintaining data consistency, it is necessary to perform exclusive control using Database Locking which is described below.

Exclusive control using database locking

It is necessary to lock the data to be updated using an appropriate method due to the following reasons:

- To maintain consistency of data stored in the database
- To prevent conflicts in update process

The three types of methods to lock the data stored in the database are as follows:

The Architect should adequately understand the characteristics of such locking and use the appropriate locking method in accordance with the characteristics of application.

Table.6.21 Types of locking

Sr. No.	Types of locking	Applicable cases	Characteristics
1.	Automatic locking by RDBMS	• When the conditions necessary to ensure data consistency can be specified as update conditions for data. • When concurrency for the same data is less and update process also takes less time.	• Effective since check and update process are executed using single SQL. • Conditions for ensuring data consistency need to be analyzed separately as compared to optimistic locking.
2.	Optimistic locking	• When the already fetched data is being updated by other transaction and if the updated contents need to be verified. • When concurrency for the same data is less and update process also takes less time.	• Ensures that the fetched data is not updated by other transaction. • Column to manage versions needs to be defined in the table.
3.	Pessimistic locking	• When the data that is likely to remain in locked state for a longer period is updated. • When data consistency check needs to be carried out since optimistic locking cannot be used (column cannot be defined to manage versions). • When concurrency for the same data is more and update process takes longer time.	• Possibility of a process failure due to process results of other transaction is eliminated. • Costly since it is necessary to execute SELECT statement for obtaining pessimistic lock.

Note: Standards for adopting types of locking

The Architect should decide the type of locking to be used based on the functional and performance requirements.

- Optimistic locking is necessary to ensure that the database transactions such as returning and changing the data on screen are cut off and the data remains unchanged in subsequent transaction.
- When locking is needed in a single transaction, both pessimistic and optimistic locking can be implemented; however when pessimistic locking is used, lock is implemented at database level thus resulting in the

possible increase in database processing cost. It is always preferable to use optimistic locking unless there are any specific issues.

- If optimistic locking is used in a process with higher update frequency wherein multiple tables are to be updated in a single transaction, the waiting time for obtaining a lock can be minimized. However the possibility of error occurrences increases since an exclusive error may occur in between the process. If pessimistic locking is used, the waiting time for obtaining a lock is likely to increase; however since exclusive error does not occur once lock is obtained, it reduces the possibility of error occurrences.

Tip: Business transaction

In actual application development, there could also be cases where exclusive control is necessary for the transactions at business flow level. A typical example of business flow level transactions could be an application used at a travel agency for making reservations while talking to the customer.

While making travel reservations, means of transport such as railway, accommodation facilities and additional scheme, etc. are discussed. At this point, the reserved accommodation and additional scheme should remain unavailable for other users. In such a case, the status of table should be updated from 'Temporarily Reserved' to 'Reserved'. Even if the reservation is in process, other users should not be able to make the corresponding reservation.

Description of exclusive control for business transaction is skipped in this chapter since it should be analyzed and designed under business process design or functional design.

Exclusive control using row lock function of the database

In most databases, when a record is updated (UPDATE, DELETE), a row lock is obtained to prevent updates by other transactions till the primary transaction is committed or rolled back.

Thus, if the number of updated records is as anticipated, the data consistency can be ensured.

Exclusive control can be performed by using this characteristic and specifying the conditions to ensure data consistency for WHERE clause at the time of update.

The support status of row lock at the time of update for each database is shown below.

Sr. No.	Database	Version	Default setting lock	Remarks
1.	Oracle	11	Row lock	Increased memory usage due to locking.
2.	PostgreSQL	9	Row lock	There is no maximum limit on the number of rows that can be locked simultaneously since information for the modified rows is not stored in the memory. However, it is necessary to perform VACUUM periodically to write to the table.
3.	DB2	9	Row lock	Increased memory usage due to locking.
4.	MySQL InnoDB	5	Row lock	Increased memory usage due to locking.

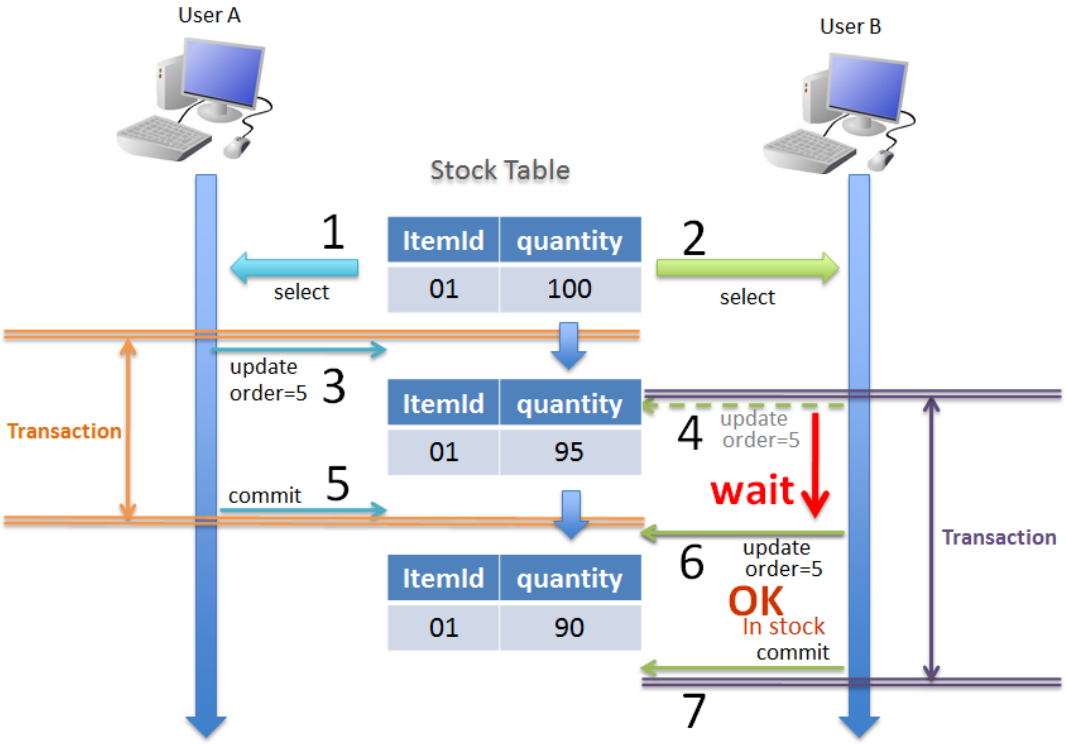
Exclusive control using row lock function of database can be used when it is not necessary to verify the contents updated by other transaction.

For example, it can be used in the purchase process of a shopping site wherein the purchased product quantity is deducted from the records that manage product stock quantity.

Exclusive control is not recommended in the processes used for status management since the earlier status is important in such processes.

See the example below illustrating a specific scenario.

- On a shopping site, both User A and User B are displayed a purchase screen of the same product at the same time. A stock quantity fetched from Stock Table is also displayed at the same time.
- 5 products were purchased at the same time; but since User A clicked “Purchase” button a bit earlier, User A buys the product first and then User B.



Sr. No.	UserA	UserB	Description
1.	○	-	User A displays a purchase screen of the product. A stock quantity of 100nos. is displayed on the screen. `select quantity from Stock where ItemId = '01'`
2.	-	○	User B displays a purchase screen of the product. A stock quantity of 100nos. is displayed on the screen. `select quantity from Stock where ItemId = '01'`
3.	○	-	User A purchases 5 products of ItemId=01. 5nos. are deducted from Stock Table. `Update from Stock set quantity = quantity - 5 where ItemId='01' and quantity >= 5`
4.	-	○	User B purchases 5 products of ItemId=01. The system tries to deduct 5 products from Stock Table; however since transaction of User A is not yet completed, the purchase process of User B is kept on hold.
5.	○	-	Transaction of user A is committed.

次のページに続く

Table. 6.22 – 前のページからの続き

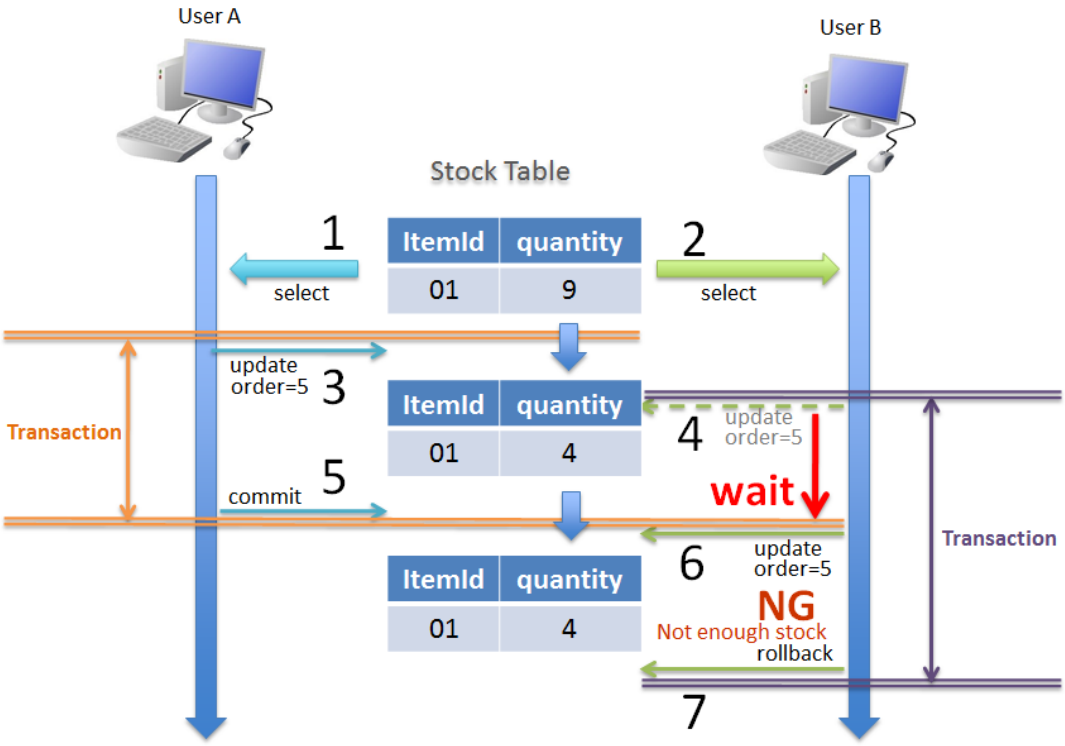
Sr. No.	UserA	UserB	Description
6.	-	○	The transaction of user A is now committed, hence purchase for User B which was kept on hold in step 4 is now resumed. If a stock screen is viewed at this point, the stock quantity is now 95 instead of 100. However, since the balance stock quantity is more than the purchase quantity (in the above example, 5), 5 is deducted from Stock Table. `Update from Stock set quantity = quantity - 5 where ItemId='01' and quantity >= 5`
7.	-	○	Transaction of user B is committed.

Note: Important

It is important to specify the deduction (`"quantity -5"`) and update condition (`"and quantity >= 5"`) in SQL.

In the similar scenario as above, if the stock quantity when a product purchase screen is displayed is 9, the stock quantity when the User B resumes update process becomes 4. As it does not satisfy `quantity >= 5`condition, update count becomes 0.

If the update count in the application is 0, purchase is rolled back and User B is prompted for re-execution.



Note: Important

It is important to verify the update count in the application and an error should occur if it is different from the expected count and transaction should be rolled back.

When this method is used for locking, the process can be continued depending on conditions even if there is a change in the referred information and data consistency can be ensured by using database function.

Exclusive control using optimistic locking

Optimistic locking is a method for ensuring data consistency wherein the data is not locked for transaction and is updated only after checking whether it is same as fetched data.

When optimistic locking is to be used, a Version column for managing versions is created to determine if the data to be updated is same as fetched data.

Data consistency can be ensured by keeping both the versions at the time of data fetch and data update same as a condition for update.

Note: Version column

This column is used in optimistic locking for managing the update count of a record. It is set to 0 when a record is inserted and then it is incremented with each successful update. The Version column can be also be substituted with the latest update timestamp in place of a number. However, when timestamp is used, uniqueness when processes are executed simultaneously cannot be ensured. Hence, in order to ensure uniqueness, it is necessary to use a number in Version column.

Exclusive control with optimistic locking is used when it is necessary to verify the contents updated by other transaction.

For example, consider a case of workflow application wherein an applicant and an approver perform concurrent operations (withdrawal and approval).

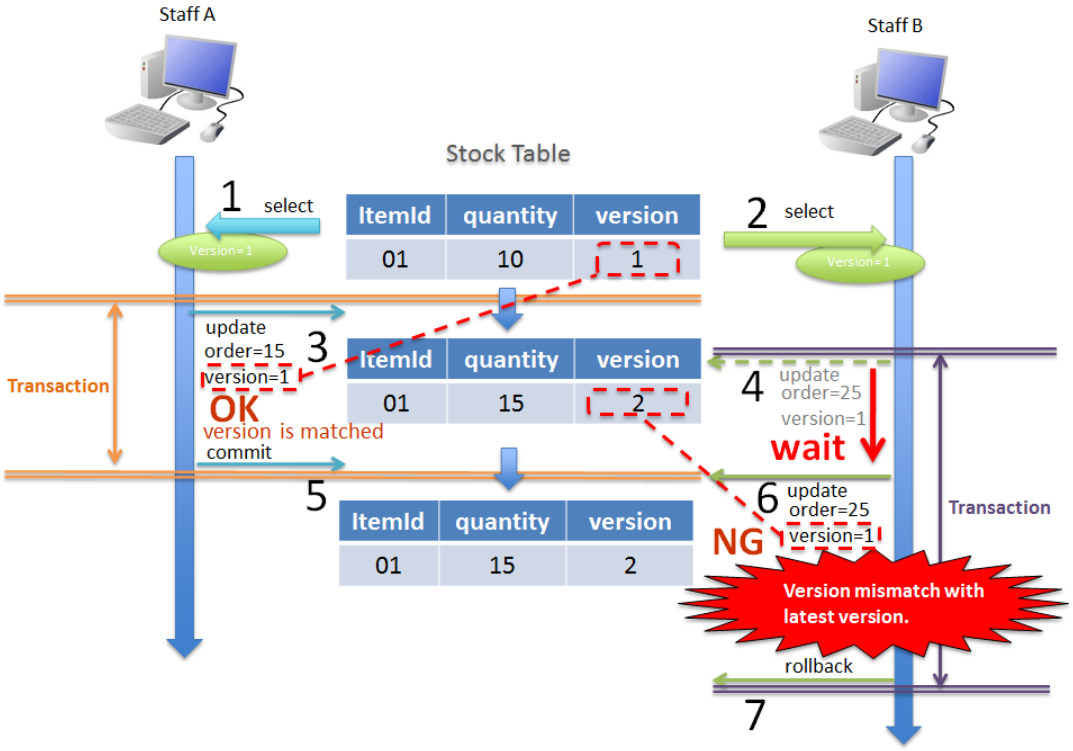
In this case, using exclusive control with optimistic locking, it is possible to notify the applicant and the approver that the operation is yet to be completed since the status before and after the operation are different.

Warning: When an optimistic locking is to be performed, it is not appropriate to update/delete the record by adding conditions other than ID and Version. This is because when the data cannot be updated, it is difficult to determine whether the reason for update failure is version mismatch or condition mismatch. When the conditions for update are different, it is necessary to check whether previous process meets all the conditions.

See the example below illustrating a specific scenario.

- Staff (Staff A, Staff B) who manage the stock quantity of shopping site add the product stock. Let us assume that Staff A adds 5nos. and Staff B adds 15nos.
- A stock management screen is displayed in order to reflect the added stock in the stock management system. The stock quantity being stored in Stock Management System is then displayed.

- Against the displayed stock quantity, the value calculated by summing up the quantity added by the staff is entered in the Update form and the total stock is updated.



Sr. No.	StaffA	StaffB	Description
1.	○	-	Staff A displays stock management screen of the product. The stock quantity of 10nos. is displayed on the screen. Version of the referred data is `1`.
2.	-	○	Staff B displays stock management screen of the product. The stock quantity of 10nos. is displayed on the screen. Version of the referred data is `1`.

3.	○	-	Staff A adds the stock of 5 nos. against the stock quantity of 10 nos. displayed on the screen and updates stock quantity as 15nos. Version of the referred data is included as an update condition.
			`UPDATE Stock SET quantity = 15, version = version + 1`
			`WHERE itemId = '01' and version = 1`

4.	-	○	Although Staff B adds 15nos. to the stock quantity of 10 nos. displayed on the screen and attempts to update the stock quantity to 25 nos.; however the transaction is kept on hold since the transaction of Staff A is not completed yet. Version of the referred data is included as an update condition.
5.	○	-	Transaction of Staff A is committed. **Version becomes 2 at this point.**

6.	○	-	Update process of Staff B which was kept on hold in step 4 is now resumed since the transaction of Staff A is committed. At this time, since the version of Stock Table data is `2`, update result is 0 records. When the update result is 0 records, an exclusive error occurs.
			`UPDATE Stock SET quantity = 25, version = version + 1`
			`WHERE itemId = '01' and version = 1`

7.	○	-	The transaction of Staff B is rolled back.

Note: Points

Version (`"version + 1"`) should be incremented and update condition (`"and version = 1"`) should be specified in SQL.

Exclusive control using pessimistic locking

Pessimistic locking is a method to lock the data to be updated at the time of fetching so that it is not updated by other transaction.

When pessimistic locking is to be used, lock is obtained for the record to be updated immediately after starting a transaction.

Since the locked record is not updated by other transaction till the transaction is committed or rolled back, data

consistency can be ensured.

Table.6.23 RDBMS-wise pessimistic lock acquisition method

Sr. No.	Database	Pessimistic locking method
1.	Oracle	FOR UPDATE
2.	PostgreSQL	FOR UPDATE
3.	DB2	FOR UPDATE WITH
4.	MySQL	FOR UPDATE

Note: About pessimistic locking timeout

At the time of obtaining a pessimistic lock, if a lock is obtained by some other transaction, then the expected behavior is specified as an option in some cases. In case of Oracle,

- Default value is `select for update [wait]`, and it waits till lock is released.
- If set to `select for update nowait`, it immediately gives a resource busy error, if locked by other transaction.
- If set to `select for update wait 5`, it waits for 5 seconds, and gives a resource busy error, if the lock is not released within 5 seconds.

Although there are variations in the functions as per DB, it is necessary to analyze the method to be used when using pessimistic locking.

Note: When using JPA (Hibernate)

Although the method of fetching a pessimistic lock varies for each database, such differences are absorbed by JPA (Hibernate). Refer to Hibernate Developer Guide for RDBMS that supports Hibernate.

Exclusive control with pessimistic locking is used when it is applicable to any of the 3 cases given below.

1. Data to be updated is managed by dividing it in multiple tables.

   When the data to be updated is divided into multiple tables, it is necessary to ensure that there are no updates by other transaction, till the update of each table is completed.

2. Status of the fetched data needs to be checked before performing update.

   After completing the checks, it is necessary to ensure that there are no updates by other transaction.

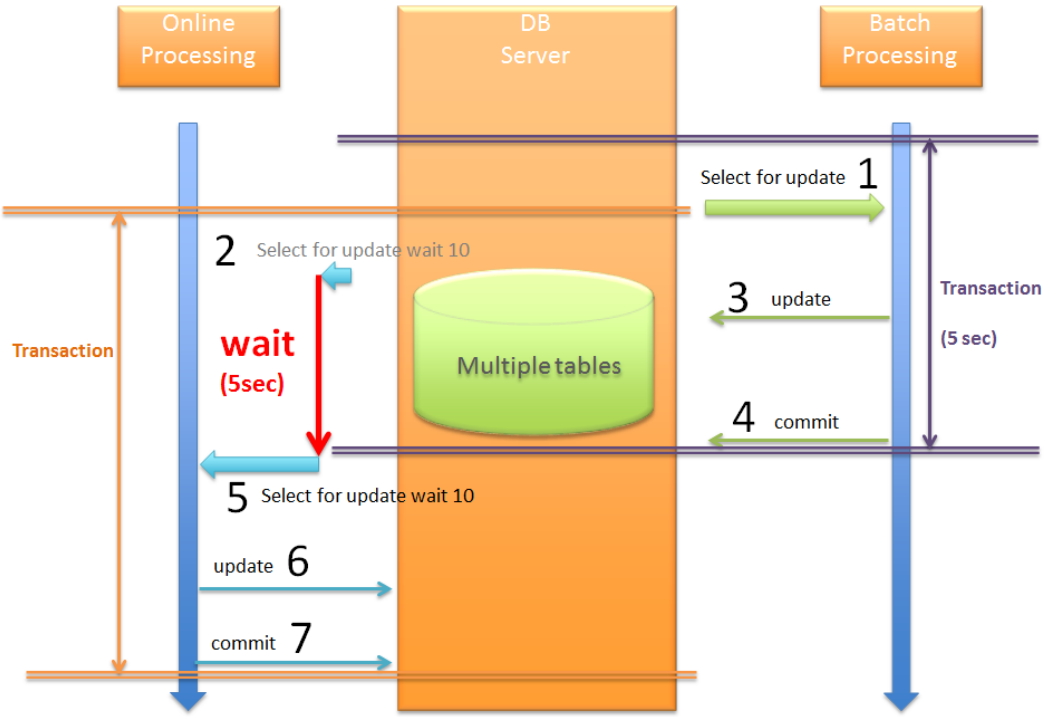
3. Online processing may be executed during batch execution.

   In batch processing, locks are collectively obtained for the data to be updated so as to ensure that exclusive error does not occur in between the execution.

   In case of locks which are obtained collectively, the locking period for online processing may be longer. In such a case, it is advisable to use pessimistic locking by specifying a timeout period.

See the example below illustrating specific scenario.

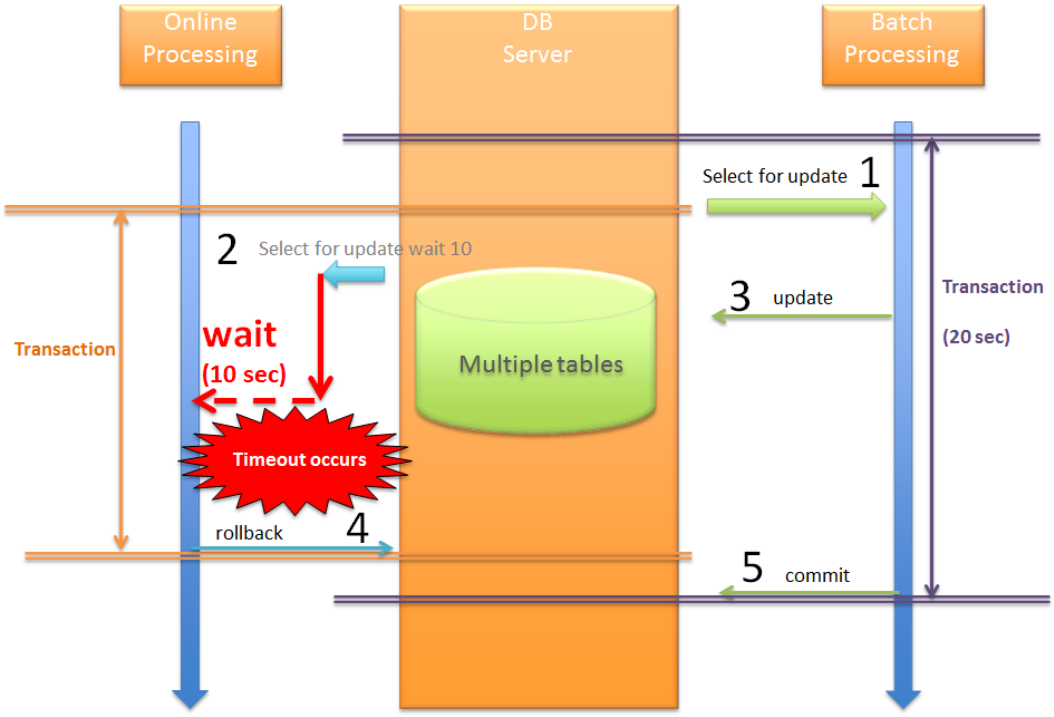
- Batch processing has already started execution, and data to be updated online is locked by pessimistic locking.
- Timeout period of 10 seconds is specified for the online processing and lock is obtained for the data to be updated.
- Batch processing is terminated after 5 seconds (before timeout).



Sr. No.	Online	Batch	Description
1.	-	○	Batch processing obtains the pessimistic lock for the data to be updated in online processing.
2.	○	-	Online processing tries to perform pessimistic locking for the data to be updated; however it is kept on hold since the pessimistic locking is performed by the transaction of batch processing. `SELECT * FROM Stock WHERE quantity < 5 FOR UPDATE WAIT 10`
3.	-	○	Batch processing updates data.
4.	-	○	Batch processing transaction is committed.
5.	○	-	Since the batch processing transaction is committed, online processing is resumed. As the fetched data reflects the update results of batch processing, data inconsistency does not occur.
6.	○	-	Online processing updates data.
7.	○	-	Online processing transaction is committed.

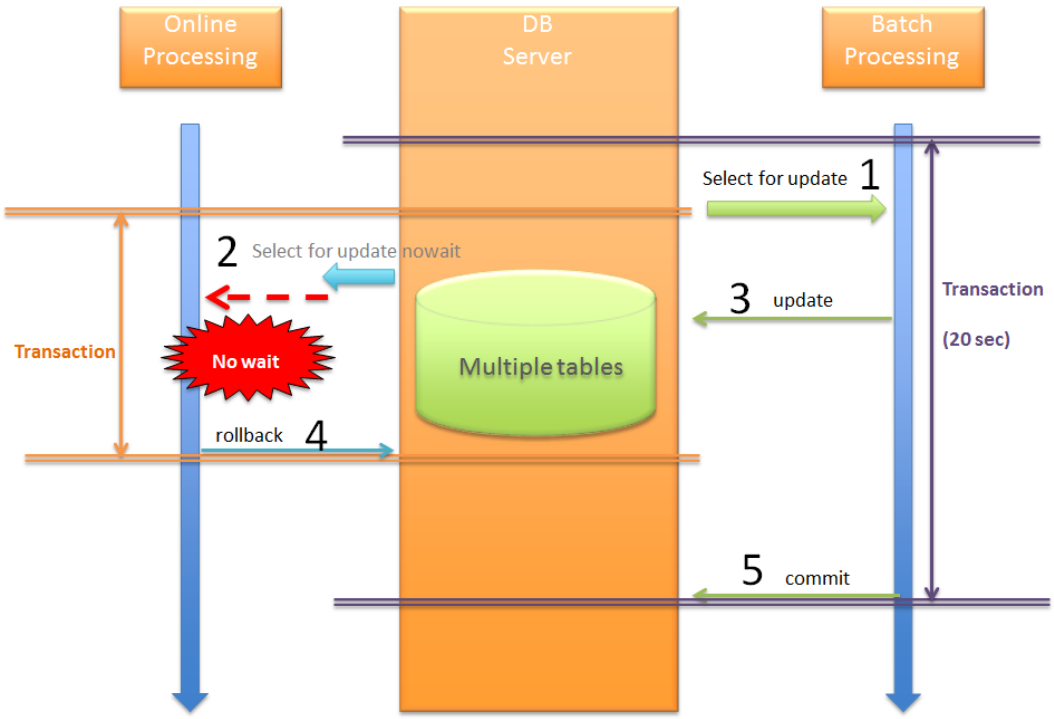
The flow at the time of timeout is given below.

Exclusive error occurs without waiting for the batch processing to end.



The flow given below illustrates a case wherein pessimistic lock is being obtained by other transaction in case of "pessimistic lock no wait" setting.

An exclusive error occurs immediately without waiting for the release of pessimistic lock.



When there is a possibility of conflict between batch processing and online processing and if batch processing is going to take longer time, it is recommended to specify timeout period of pessimistic exclusive locking. The timeout period should be determined based on the online processing requirements.

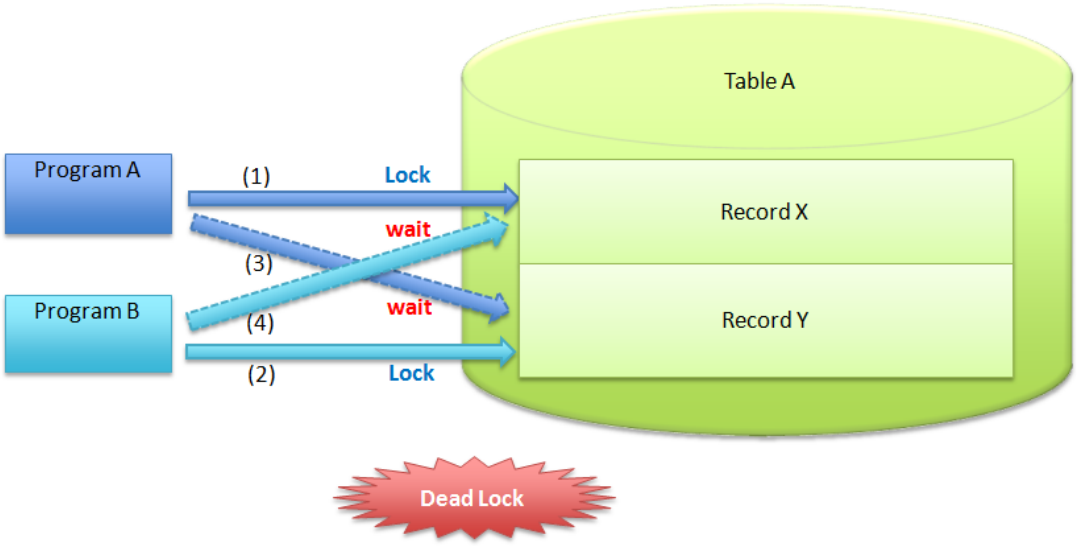
Prevention of deadlock

When using database locks, it should be noted that if multiple records are updated in same transaction, 2 deadlocks shown below are likely to occur.

- *Deadlock in table*
- *Deadlock between tables*

Deadlock in table

This deadlock occurs when the records of the same table are locked by multiple transactions as shown in the flow of (1)-(5) below.



Sr. No.	Program A	Program B	Description
(1)	○	-	Program A obtains the lock for Record X.
(2)	○	-	Program B obtains the lock for Record Y.
(3)	○	-	Program A tries to obtain the lock for Record Y that is locked by the transaction of Program B, however since lock of (2) is not released, the status is changed to “Waiting for release”.
(4)	-	○	Program B tries to obtain the lock for Record X that is locked by the transaction of Program A, however since lock of (1) is not released, the status is changed to “Waiting for release”.
(5)	-	-	Since Program A and Program B both have the “Waiting for release” status for each other, it results into a deadlock. When a deadlock occurs, it is detected by the database and error is thrown.

Note: How to resolve a deadlock

The deadlock can be resolved by timeout or retry; however, it is important to determine the rules for the update sequence of records in the same table. When rows are to be updated one by one, the rules such as updating in ascending order of PK (PRIMARY KEY) should be set.

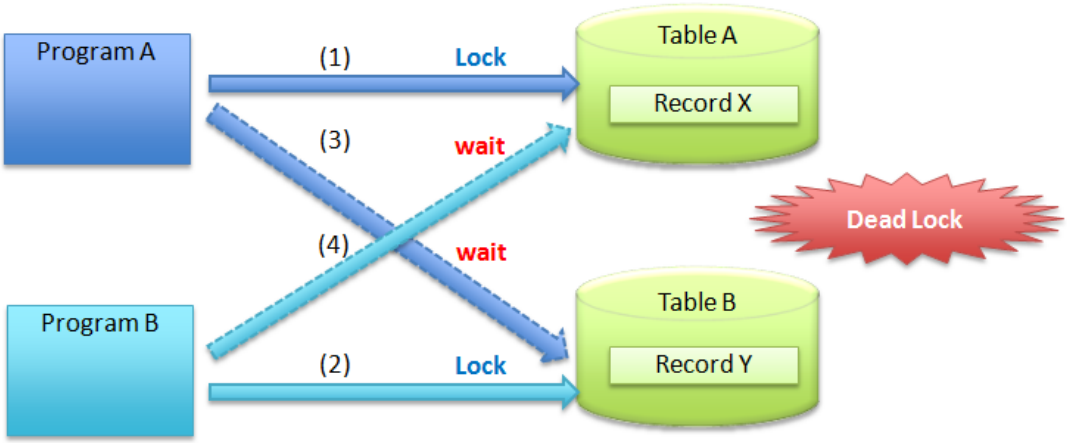
Let us say if both Program A and Program B follow the rule of starting the update from Record X, the

deadlock shown in figure *Deadlock in table* above no longer occurs.

Deadlock between tables

This deadlock occurs when records of different tables are locked by multiple transactions as shown in the flow of (1)-(5) below.

The basic concept is same as *Deadlock in table*.



Sr. No.	Program A	Program B	Description
(1)	○	-	Program A obtains the lock for Record X of Table A.
(2)	○	-	Program B obtains the lock for Record Y of Table B.
(3)	○	-	Program A tries to obtain the lock for Record Y of Table B that is locked by the transaction of Program B, however since lock of (2) is not released, the status is changed to “Waiting for release”.
(4)	-	○	Program B tries to obtain the lock of Record X of Table A that is locked by the transaction of Program A, however since lock of (1) is not released, the status is changed to “Waiting for release”.
(5)	-	-	Since Program A and Program B both have the “Waiting for release” status for each other, it results into a deadlock. When a deadlock occurs, it is detected by the database and error is thrown.

Note: How to resolve a deadlock

Although deadlocks can be resolved by a timeout or retry; it is important to define rules for update sequence across the tables.

Let us say if both Program A and Program B follow the rule of starting the update from Table A, the deadlock shown in figure *Deadlock between tables* above no longer occurs.

Warning: A deadlock might still occur due to sequence of locking records even after adopting either of the methods as a precaution. The rules should be defined for the lock sequence of tables and records.

6.4.2 How to use

From here, implementation method of exclusive control using O/R Mapper is described.

First confirm the implementation method of O/R Mapper to be used.

- *How to implement while using MyBatis3*
- *How to implement while using JPA (Spring Data JPA)*

For method of handling exclusive error, refer to:

- *How to handle an exclusive error*

How to implement while using MyBatis3

Row lock function of RDBMS

When exclusive control is to be performed using a row lock function of RDBMS, the following should be considered in SQL.

- Update contents to be specified in SET clause
- Update conditions to be specified in WHERE clause

- Define a method in Repository interface.

```
public interface StockRepository {
    // (1)
    boolean decrementQuantity(@Param("itemCode") String itemCode,
                              @Param("quantity") int quantity);
}
```

Sr. No.	Description
(1)	In Repository interface, define a method to update data using row lock function of RDBMS. In above example, a method to decrease stock quantity is defined. When stock quantity is decreased, `true` is returned.

- Define SQL wherein exclusive control using a row lock function of RDBMS becomes valid.

```
<!-- (2) -->
<update id="decrementQuantity">
<![CDATA[
    UPDATE
        m_stock
    SET
        /* (3) */
        quantity = quantity - #{quantity}
    WHERE
        item_code = #{itemCode}
    AND
        /* (4) */
        quantity >= #{quantity}
]]>
</update>
```

Sr. No.	Description
(2)	Define statement (SQL) to update the data using row lock function of RDBMS. In above example, SQL to decrease stock quantity is defined. When a row lock function of RDBMS is to be used, the data can be safely updated because of the operations given below. • When other transaction has obtained lock for the same data, SQL is executed after releasing (commit or rollback) the lock. • When stock quantity is decreased successfully, row lock of RDBMS is fetched, and update from other transactions gets locked.
(3)	Subtract the stock quantity (`quantity = quantity -#{quantity}`) in SQL.
(4)	As an update condition, add "stock quantity should be greater than or equal to the order quantity (`quantity >= #{quantity}`).

- Call the Repository method to safely update the data using row lock function of RDBMS.

```
// (5)
boolean updated = stockRepository.decrementQuantity(itemCode,␣
↪quantityOfOrder);
// (6)
if (!updated) {
    // (7)
    ResultMessages messages = ResultMessages.error().add(ResultMessage
            .fromText("Not enough stock. Please, change quantity."));
    throw new BusinessException(messages);
}
```

Sr. No.	Description
(5)	Call the Repository method to perform update.
(6)	Call the Repository method to determine result. In case of `false`, stock quantity will be insufficient as update conditions are not fulfilled.
(7)	Business error occurs. In above example, only business rules are checked (stock quantity check) while performing exclusive control. Therefore, when update conditions are not fulfilled, it is being treated as business error and not exclusive error. Business errors should be appropriately handled in Controller.

Optimistic locking

In MyBatis3, a mechanism to perform optimistic locking as library is not provided.

Therefore, when performing optimistic locking, the version should be considered in SQL.

- Define a property for version control in entity.

```
public class Stock implements Serializable {
    private static final long serialVersionUID = 1L;

    private String itemCode;
    private int quantity;
    // (1)
    private long version;

    // ...

}
```

Sr. No.	Description
(1)	Create a property for version control in entity.

- Define a method in Repository interface.

```
public interface StockRepository {
    // (2)
    Stock findOne(String itemCode);
    // (3)
    boolean update(Stock stock);
}
```

Sr. No.	Description
(2)	Define a method to fetch entity, in Repository interface.
(3)	Define a method to update data using optimistic lock function, in Repository interface. In above example, a method to update records with specified entity details is defined. When update is successful, `true` is returned.

- Define SQL in mapping file.

```
<!-- (4) -->
<select id="findOne" parameterType="string" resultType="Stock">
    SELECT
        item_code,
        quantity,
        version
    FROM
        m_stock
    WHERE
        item_code = #{itemCode}
</select>

<!-- (5) -->
<update id="update" parameterType="Stock">
    UPDATE
        m_stock
    SET
        quantity = #{quantity},
        /* (6) */
        version = version + 1
    WHERE
        item_code = #{itemCode}
    AND
        /* (7) */
        version = #{version}
```

```
</update>
```

Sr. No.	Description
(4)	Define statement (SQL) to fetch entity. When using optimistic lock, it is necessary to obtain version at the time of fetching entity.
(5)	Define statement (SQL) to update data using optimistic lock function. In above example, SQL to update records with specified entity details is defined.
(6)	Update the version (`version = version + 1`) in SQL.
(4)	As an update condition, add "version should not be changed (`version = #{version}`).

- Call the Repository method to safely update the data using optimistic lock function.

```
// (5)
Stock stock = stockRepository.findOne(itemCode);
if (stock == null) {
    ResultMessages messages = ResultMessages.error().add(ResultMessage
            .fromText("Stock not found. itemCode : " + itemCode));
    throw new ResourceNotFoundException(messages);
}

// (6)
stock.setQuantity(stock.getQuantity() + addedQuantity);

// (7)
boolean updated = stockRepository.update(stock);
if(!updated) {
    // (8)
    throw new ObjectOptimisticLockingFailureException(Stock.class, itemCode);
}
```

Sr. No.	Description
(5)	Call findOne method of Repository interface to fetch the entity.
(6)	Specify the value to be updated for the entity fetched in step (5). In above example, procured stock quantity is added.
(7)	Call the update method of Repository interface to reflect the entity updated in step (5) in persistence layer (DB).
(8)	Determine the update result. In case of `false`, entity gets updated by other transaction, thereby leading to optimistic lock error (`org.springframework.orm.ObjectOptimisticLockingFailureException`).

When performing optimistic lock for long transactions, points given below should be noted.

> **Warning:** When performing optimistic locking for long transactions, apart from checking version at the time of update, it should also be checked at the time of fetching data.

Example of implementation is given below.

- Check version even at the time of fetching data.

```
Stock stock = stockRepository.findOne(itemCode);
if (stock == null || stock.getVersion() != version) {
    // (9)
    throw new ObjectOptimisticLockingFailureException(Stock.class, itemCode);
}

stock.setQuantity(stock.getQuantity() + addedQuantity);
boolean updated = stockRepository.update(stock);
// ...
```

Sr. No.	Description
(9)	Compare the version of entity fetched in (5) with that of the entity fetched in other database transaction. If version differs, the data is updated by other transaction, thereby leading to optimistic lock error (`org.springframework.dao.ObjectOptimisticLockingFailureException`). It is also necessary to consider a case wherein data does not exist (`stock == null`), The implementation should be as per the application specifications. In above example, it is being treated as optimistic locking error.

Following points should be noted when application uses optimistic lock function in combination with row lock function of RDBMS.

> **Warning:** In case of application in which a process to perform exclusive control using row lock function of RDBMS and a process to perform exclusive control using optimistic lock function of RDBMS coexist, **version should be updated (incremented)** in SQL using row lock function of RDBMS.
>
> If version is not updated in SQL that performs exclusive control using row lock function of RDBMS, the data may get overwritten by SQL that performs exclusive control using optimistic lock function.

Example of implementation is shown below.

- Update the version in SQL.

```
<update id="decrementQuantity">
<![CDATA[
    UPDATE
        m_stock
    SET
        quantity = quantity - #{quantity},
        /* (10) */
        version = version + 1
    WHERE
        item_code = #{itemCode}
    AND
        quantity >= #{quantity}
]]>
</update>
```

Sr. No.	Description
(10)	Update (increment) the version.

Pessimistic locking

In MyBatis3, a mechanism to perform pessimistic locking as library is not provided.

Therefore, in case of pessimistic locking, a keyword should be specified to fetch lock in SQL.

- Specify the keyword to fetch lock in SQL.

```
<select id="findOneForUpdate" parameterType="string" resultType="Stock">
    SELECT
        item_code,
        quantity,
        version
    FROM
        m_stock
    WHERE
        item_code = #{itemCode}
    /* (1) */
    FOR UPDATE
</select>
```

Sr. No.	Description
(1)	For SQL in which pessimistic lock needs to be fetched, specify the keyword to fetch the same. Keyword and location to specify the keyword differ depending on database.

How to implement while using JPA (Spring Data JPA)

Row lock function of RDBMS

When exclusive control is to be performed using a row lock function of RDBMS, Query method is added to Repository interface for implementation.

For Query method, refer to *Adding query method* and *Operating the entities of Persistence Layer directly*.

- Repository interface

```
public interface StockRepository extends JpaRepository<Stock, String> {

    @Modifying
    @Query("UPDATE Stock s"
            + " SET s.quantity = s.quantity - :quantity"
            + " WHERE s.itemCode = :itemCode"
            + " AND :quantity <= s.quantity")  // (1)
    public int decrementQuantity(@Param("itemCode") String itemCode,
            @Param("quantity") int quantity);

}
```

Sr. No.	Description
(1)	When the stock quantity is equal to or more than the order quantity, JPQL is specified in Query method to reduce stock quantity. Since it is necessary to check the update count, `int` is specified as the return value of Query method.

- Service

```
String itemCodeOfOrder = "ITM0000001";
int quantityOfOrder = 31;

int updateCount = stockRepository.decrementQuantity(itemCodeOfOrder,␣
↪quantityOfOrder); // (2)
if (updateCount == 0) { // (3)
    ResultMessages message = ResultMessages.error();
    message.add(ResultMessage
            .fromText("Not enough stock. Please, change quantity."));
    throw new BusinessException(message); // (4)
}
```

```
update m_stock set quantity=quantity-31
               where item_code='ITM0000001' and 31<=quantity -- (5)
```

Sr. No.	Description
(2)	Call the Query method.
(3)	Determine the call results of Query method. In case of `0`, the stock quantity becomes inadequate, since update conditions are not satisfied.
(4)	Store the message notifying "No stock" or "Not enough stock" to generate a business error. The generated error should be handled appropriately in Controller as per the requirements. In the above example, only business rules are checked while performing exclusive control; hence when update conditions are not satisfied, it is treated as business error and not exclusive error. For error handling methods, refer to *Coding Points (Controller)*.
(5)	SQL that is executed while calling the Query method.

Optimistic locking

In JPA, optimistic locking can be performed by specifying `@javax.persistence.Version`annotation in property for version control.

- Entity

```
@Entity
@Table(name = "m_stock")
public class Stock implements Serializable {

    @Id
    @Column(name = "item_code")
    private String itemCode;

    private int quantity;

    @Version // (1)
    private long version;

    // ...

}
```

Sr. No.	Description
(1)	Specify the `@Version` annotation in property for version control.

- Service

```
String itemCode = "ITM0000001";
int newQuantity = 30;

Stock stock = stockRepository.findOne(itemCode); // (2)
if (stock == null) {
    ResultMessages messages = ResultMessages.error().add(ResultMessage
            .fromText("Stock not found. itemCode : " + itemCode));
    throw new ResourceNotFoundException(messages);
}

stock.setQuantity(newQuantity); // (3)

stockRepository.flush(); // (4)
```

```
update m_stock set quantity=30, version=7
               where item_code='ITM0000001' and version=6 -- ( 5)
```

Sr. No.	Description
(2)	Call findOne method of Repository interface to fetch the entity.
(3)	Specify the value to be updated for the entity fetched in step (2).
(4)	Reflect the changes of (3) in the persistence layer (DB). This process is usually not required since it is performed for the description purpose. Normally, it is reflected automatically when the transaction is committed. In the above example, when the version held by the entity fetched in step (2) and the version stored in persistence layer (DB) do not match, optimistic locking error (`org.springframework.dao.OptimisticLockingFailureException`) occurs.
(5)	SQL that is executed while reflecting to persistence layer (DB) of step (4).

It is important to note the following points while performing optimistic locking for long transactions.

Warning: It is not sufficient to simply assign `@Version` annotation for optimistic locking which is to be performed for long transactions. When optimistic locking is to be performed for long transactions, version check should also be carried out while fetching the data to be updated, in addition to the check at the time of update performed using JPA function.

An implementation example is given below.

- Service

```
long version = 12;
String itemCode = "ITM0000001";
int newQuantity = 30;

Stock stock = stockRepository.findOne(itemCode); // (1)
if (stock == null || stock.getVersion() != version) { // (2)
    throw new ObjectOptimisticLockingFailureException(Stock.class, itemCode);
↪// (3)
}

stock.setQuantity(newQuantity);

stockRepository.flush();
```

Sr. No.	Description
(1)	Fetch an entity from persistence layer (DB).
(2)	Compare the version of the entity fetched by a different database transaction in advance with the latest version of persistence layer (DB) fetched in step (1). If versions match, optimistic locking which uses `@Version` annotation becomes valid in subsequent processes.
(3)	If the versions are different, generate the optimistic locking error (`org.springframework.dao.ObjectOptimisticLockingFailureException`).

Warning: Setting a value in property for Version control

Entity fetched using Repository interface is called “Managed entity”.

For “Managed entity”, it should be noted that a value cannot be set in a process for the property

for Version control.

Even if a process as shown below is carried out, the value of version that is set to "Managed entity" is not reflected. Hence it is not used to fetch an optimistic lock. The version when a value is fetched using findOne method is used for optimistic locking.

```
long version = 12;
String itemCode = "ITM0000001";
int newQuantity = 30;

Stock stock = stockRepository.findOne(itemCode);
if (stock == null) {
    ResultMessages messages = ResultMessages.error().add(ResultMessage
            .fromText("Stock not found. itemCode : " + itemCode));
    throw new ResourceNotFoundException(messages);
}
stock.setVersion(version); // ★ Invalid Processing
stock.setQuantity(newQuantity);

stockRepository.flush();
```

For example, even if the value of the version sent from the screen is overwritten, it is not reflected in entity. Hence the exclusive control can no longer be appropriately performed.

Note: Standardization of optimistic locking for long transactions

When optimistic locking for long transactions becomes necessary in multiple processes, it is desirable to standardize the processes of (1) ~ (3) described above. For standardization method, refer to *How to add custom method* .

It is important to note the following points when both row lock function of RDBMS and optimistic locking function are to be used.

Warning: Version must always be updated in Query method that uses row lock function of RDBMS in case of applications wherein a process performing exclusive control using a row lock function of RDBMS and a process performing exclusive control using optimistic locking function co-exist, for the same data.

If version is not updated in Query method that performs exclusive control using row lock function of RDBMS, the contents updated by Query method may be overwritten by the process of a different transaction. Hence, the exclusive control is not performed properly.

An implementation example is given below.

- Repository interface

```
public interface StockRepository extends JpaRepository<Stock, String> {

    @Modifying
    @Query("UPDATE Stock s SET s.quantity = s.quantity - :quantity"
            + ", s.version = s.version + 1" // (1)
            + " WHERE s.itemCode = :itemCode"
            + " AND :quantity <= s.quantity")
    public int decrementQuantity(@Param("itemCode") String itemCode,
            @Param("quantity") int quantity);

}
```

Sr. No.	Description
(1)	Version needs to be updated (`s.version = s.version + 1`).

Pessimistic locking

In Spring Data JPA, the pessimistic lock can be performed by specifying `@org.springframework.data.jpa.repository.Lock` annotation.

- Repository interface

```
public interface StockRepository extends JpaRepository<Stock, String> {

    @Lock(LockModeType.PESSIMISTIC_WRITE) // (1)
    @Query("SELECT s FROM Stock s WHERE s.itemCode = :itemCode")
    Stock findOneForUpdate(@Param("itemCode") String itemCode);

}
```

```
-- (2)
SELECT
        stock0_.item_code AS item1_5_
        ,stock0_.quantity AS quantity2_5_
        ,stock0_.version AS version3_5_
    FROM
        m_stock stock0_
    WHERE
        stock0_.item_code = 'ITM0000001'
    FOR UPDATE;
```

Sr. No.	Description
(1)	Specify `@Lock` annotation in Query method.
(2)	Executed SQL. In the above example, the SQL executed while using PostgreSQL is given.

The types of pessimistic locking that can be specified using `@Lock` annotation are as given below.

Sr. No.	LockModeType	Description	Issued SQL
1.	PESSIMISTIC_READ	Pessimistic lock for read is obtained. It acts as a shared lock rather than an exclusive lock depending on the database. Lock is released at the time of commit or rollback	select ... for update / select ... for share
2.	PESSIMISTIC_WRITE	Pessimistic lock for update is obtained and an exclusive lock is applied. In case of exclusive lock, if a lock has already been applied, the entity is fetched after waiting for the lock to be released. Lock is released at the time of commit or rollback	select ... for update
3.	PESSIMISTIC_FORCE_INCREMENT	Exclusive lock is applied to the target data when entity is fetched. Version is forcibly updated immediately after fetching the entity. Lock is released at the time of commit or rollback	select ... for update + update

Note: Lock timeout period

Timeout period can be specified by specifying `"javax.persistence.lock.timeout"` as JPA (`EntityManager`) settings or Query hint.

There are 2 methods for specifying the locking timeout period: 1. Method wherein it is specified for the entire process 2. Method wherein it is specified for each Query.

Method wherein it is specified for the entire process is as follows:

- `xxx-infra.xml`

```
<bean id="entityManagerFactory"
    class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean
↪">
    <property name="packagesToScan" value="xxxxxx.yyyyyy.zzzzzz.domain.model"␣
↪/>
    <property name="dataSource" ref="dataSource" />
    <property name="jpaVendorAdapter" ref="jpaVendorAdapter" />
    <property name="jpaPropertyMap">
        <util:map>
            <!-- ... -->
            <entry key="javax.persistence.lock.timeout" value="1000" /> <!--␣
↪(1) -->
        </util:map>
    </property>
</bean>
```

Sr. No.	Description
(1)	Specify the timeout in milliseconds. If `1000` is specified, it equals 1 second.

Note: nowait support

When `0` is specified for Oracle and PostgreSQL, `nowait` is added, and when locked by another transaction, an exclusive error occurs without waiting for release of lock.

Warning: Regarding the issue that "nowait" clause is not added while using PostgreSQL

Due to an issue in Hibernate 5.0.X which is a dependent library of TERASOLUNA Server Framework for Java 5.3.0 RELEASE version (HHH-10797), "nowait" clause is not added to SQL when PostgreSQL is used even when timeout period is set to "0".

Hence, it is necessary to review following countermeasures.

- Apply a patch to `Dialect` class offered by Hibernate.
- Upgrade to higher version of Hibernate 5.2.1.

> **Warning: Restrictions of PostgreSQL**
>
> Although nowait can be specified in PostgreSQL, it is not possible to specify waiting time. Therefore, the measures such as separately providing timeout for Query etc. should be implemented.

Method wherein it is specified for each Query is as follows:

- Repository interface

```
@Lock(LockModeType.PESSIMISTIC_WRITE)
@QueryHints(@QueryHint(name = "javax.persistence.lock.timeout", value = "2000
↪")) // (1)
@Query("SELECT s FROM Stock s WHERE s.itemCode = :itemCode")
Stock findOneForUpdate(@Param("itemCode") String itemCode);
```

Sr. No.	Description
(1)	Specify the timeout in milliseconds. If `2000` is specified, it equals 2 seconds. All the specified values are overwritten.

How to handle an exclusive error

Error handling in case of optimistic locking failure

When optimistic locking fails, `org.springframework.dao.OptimisticLockingFailureException`occurs; hence it is necessary to handle it appropriately in Controller.

The handling method varies with the operation specifications of application when optimistic locking error occurs.

When there is no need to change the operation at request level, it is handled by using `@ExceptionHandler` annotation.

```
@ExceptionHandler(OptimisticLockingFailureException.class) // (1)
public ModelAndView handleOptimisticLockingFailureException(
        OptimisticLockingFailureException e) {
    // (2)
    ExtendedModelMap modelMap = new ExtendedModelMap();
    ResultMessages resultMessages = ResultMessages.warning();
    resultMessages.add(ResultMessage.fromText("Other user updated!!"));
    modelMap.addAttribute(setUpForm());
    modelMap.addAttribute(resultMessages);
```

```
    String viewName = top(modelMap);
    return new ModelAndView(viewName, modelMap);
}
```

Sr. No.	Description
(1)	Specify `OptimisticLockingFailureException.class` in the value attribute of `@ExceptionHandler` annotation.
(2)	Carry out error handling. Generate the message to notify error and information required for screen display (form or other model) and return `ModelAndView` specifying the destination. For details on error handling, refer to *Method to handle exception at use case level*.

If there is a need to change the operation at request level, it is to be handled using `try -catch` in the handler method of Controller.

```
@RequestMapping(value = "{itemId}/update", method = RequestMethod.POST)
public String update(StockForm form, Model model, RedirectAttributes␣
↪attributes){

    // ...

    try {
        stockService.update(...);
    } catch (OptimisticLockingFailureException e) { // (1)
        // (2)
        ResultMessages resultMessages = ResultMessages.warn();
        resultMessages.add(ResultMessage.fromText("Other user updated!!"));
        model.addAttribute(resultMessages);
        return updateRedo(modelMap);
    }

    // ...

}
```

Sr. No.	Description
(1)	Catch `OptimisticLockingFailureException`.
(2)	Carry out error handling. Generate the message to notify error and information required for screen display (form or other model) and return the destination view name. For details on error handling, refer to *Method to handle exception at use case level*.

Error handling in case of pessimistic locking failure

When pessimistic locking fails, `org.springframework.dao.PessimisticLockingFailureException` occurs; hence it is necessary to handle it appropriately in Controller.

The handling method varies with the operation specifications of the application when pessimistic locking error occurs.

If there is no need to change the operation at request level, it is handled using `@ExceptionHandler` annotation.

```
@ExceptionHandler(PessimisticLockingFailureException.class) // (1)
public ModelAndView handlePessimisticLockingFailureException(
        PessimisticLockingFailureException e) {
    // (2)
    ExtendedModelMap modelMap = new ExtendedModelMap();
    ResultMessages resultMessages = ResultMessages.warning();
    resultMessages.add(ResultMessage.fromText("Other user updated!!"));
    modelMap.addAttribute(setUpForm());
    modelMap.addAttribute(resultMessages);
    String viewName = top(modelMap);
    return new ModelAndView(viewName, modelMap);
}
```

SR. No.	Description
(1)	Specify `PessimisticLockingFailureException .class` in value attribute of `@ExceptionHandler` annotation.
(2)	Carry out error handling. Generate the message to notify error and information required for screen display (form or other model) and return `ModelAndView` specifying the destination. For details on error handling, refer to *Method to handle exception at use case level*.

If there is need to change the operation at request level, it is to be handled using `try -catch` in the handler

method of Controller.

```
@RequestMapping(value = "{itemId}/update", method = RequestMethod.POST)
public String update(StockForm form, Model model, RedirectAttributes␣
↪attributes){

    // ...

    try {
        stockService.update(...);
    } catch (PessimisticLockingFailureException e) { // (1)
        // (2)
        ResultMessages resultMessages = ResultMessages.warn();
        resultMessages.add(ResultMessage.fromText("Other user updated!!"));
        model.addAttribute(resultMessages);
        return updateRedo(modelMap);
    }

    // ...

}
```

Sr. No.	Description
(1)	Catch `PessimisticLockingFailureException`.
(2)	Carry out error handling. Generate the message to notify error and information required for screen display (form or other model) and return destination view name. For details on error handling, refer to *Method to handle exception at use case level*.

7

General-purpose functions that do not depend on the application

7.1 Logging

Note: This contents described in this chapter are just guidelines that can be customized as per the business requirements.

7.1.1 Overview

When a system is being operated, logs and messages are output as information sources to analyze the business process usage and

to determine the causes when the system is down or when a business error etc. occurs.

It is important to output the log since effectiveness of analysis improves significantly if it is output at the time of debugging.

If only the behavior is to be checked, it can be done by debugging in IDE or through a simple output such as `System.out.println`.

However, if the output result is not manually stored, it is not possible to check the results later. This hampers the effectiveness of analysis.

Obtaining the log by implementing logging library is just writing the code to be output.

Log can be verified at any convenient time later on.

Considering operating time, trail and analysis, it is recommended to implement logging library.

There are various log output methods in Java and a number of methods can be selected; however this guideline recommends SLF4J (interface) + Logback (implementation)

due to simplicity of coding, ease of modification and performance.

Types of Logs

A typical log at the time of application development is shown below.

Log level	Category	Purpose of output	Output contents
TRACE	Performance log	Measurement of request processing time (Should not be output during production environment operations)	Process start and end time, process elapsed time (ms), Information that can identify the execution process etc. (execution controller + method, request URL etc.).
DEBUG	Debug log	Debug at the time of development (Should not be output during production environment operations)	Optional (Executed query, Input parameter, Return value etc.)
INFO	Access log	Assessing business process volume	Information that can identify access date and time, user (IP address, authentication information) Information that can identify execution process (request URL) etc., information for leaving a trail
INFO	External communication log	Analysis of error that occurs while communicating with external system	Send and receive time, send and receive data etc.

次のページに続く

Table. 7.1 – 前のページからの続き

Log level	Category	Purpose of output	Output contents
WARN	Business error log	Business error records	Business error occurrence time, Message ID and message corresponding to business error Information necessary for analysis such as input information, exception message etc.
ERROR	System error log	Record the events where it is difficult to continue a system operation	System error occurrence time, Message and message ID corresponding to system error Information necessary for analysis such as input information, exception message etc. Basically, framework is output and business process logic is not output.
ERROR	Monitoring log	Monitoring the occurrence of an exception	Exception occurrence time, Message ID corresponding to system error Use tools for monitoring and keep the output contents to a minimum.

Debug log, Access log, External communication log, Business error log and System error log are output to same file.

In this guideline, the log file that outputs the above mentioned logs is called as application log.

Note: The sequence of log levels of SLF4J or Logback is TRACE < DEBUG < INFO < WARN < ERROR. It does not include FATAL level provided in commons-loggins or Log4J.

Output contents of log

It is important to note the points given below for the output contents of log.

1. ID to be output to log

   When log is to be monitored during the operation, it is recommended to include a message ID in the log to be monitored.

   Further, when the business process volume is to be assessed using an access log, the log should be output according to the ID assigned to each business process as explained in Message Management.

   This facilitates overall data compilation.

   Note: The readability of log is enhanced by including an ID in the log thereby reducing the time required for primary isolation of failure analysis. Refer to *Message Management* for log ID structure. However, there is no need to assign an ID to all the logs. ID is not required at the time of debugging. It is recommended when the system is operational so as to isolate the log quickly.

   During failure, when a system user is notified by displaying a log ID (or a message ID) on the error screen and the ID is then notified to the call center, for that user, failure analysis becomes easier.

   However, note that the vulnerabilities of the system may be exposed if errors are displayed on the screen along with the failure details.

   In common library, the mechanism(component) is provided to include the message ID(exception code) into the log and the screen when an exception is occurred. Details refer to "*Exception Handling*".

   ---

2. Traceability

   To improve the traceability, it is recommended to output a unique track ID (hereafter referred to as X-Track) at request level in each log.

   Example of logs including X-Track is given below.

```
date:2013-09-06 19:36:31    X-Track:85a437108e9f4a959fd227f07f72ca20 ␣
↪message:[START CONTROLLER] (omitted)
date:2013-09-06 19:36:31    X-Track:85a437108e9f4a959fd227f07f72ca20 ␣
↪message:[END CONTROLLER  ] (omitted)
date:2013-09-06 19:36:31    X-Track:85a437108e9f4a959fd227f07f72ca20 ␣
↪message:[HANDLING TIME   ] (omitted)
date:2013-09-06 19:36:33    X-Track:948c8b9fd04944b78ad8aa9e24d9f263 ␣
↪message:[START CONTROLLER] (omitted)
date:2013-09-06 19:36:33    X-Track:142ff9674efd486cbd1e293e5aa53a78 ␣
↪message:[START CONTROLLER] (omitted)
date:2013-09-06 19:36:33    X-Track:142ff9674efd486cbd1e293e5aa53a78 ␣
↪message:[END CONTROLLER  ] (omitted)
date:2013-09-06 19:36:33    X-Track:142ff9674efd486cbd1e293e5aa53a78 ␣
↪message:[HANDLING TIME   ] (omitted)
date:2013-09-06 19:36:33    X-Track:948c8b9fd04944b78ad8aa9e24d9f263 ␣
↪message:[END CONTROLLER  ] (omitted)
date:2013-09-06 19:36:33    X-Track:948c8b9fd04944b78ad8aa9e24d9f263 ␣
↪message:[HANDLING TIME   ] (omitted)
```

Logs can be linked together using Track IDs even when the output is irregular.

In the above example, it can be clearly understood that 4th, 8th and 9th rows in the log are pertaining to the same request.

In common library, `org.terasoluna.gfw.web.logging.mdc.XTrackMDCPutFilter` to be added to MDC is provided by generating a unique key for each request.

`XTrackMDCPutFilter` sets Track ID in "X-Track" of HTTP response header as well. X-Track is used as a Track ID label in the log.

Refer to *About MDC* for the usage methods.

3. Log mask

   If personal information, credit card number etc. are output to the log file as is, the information that has security threat should be masked if needed.

Log output points

Category	Output points
Performance log	The processing time of business process is measured and it is output after executing business process. Request processing time is measured and log is output when response is returned. It is usually implemented in AOP or Servlet filter. Common library provides `org.terasoluna.gfw.web.logging.TraceLoggingInterceptor` which outputs processing time of SpringMVC Controller method in TRACE log after the execution of handler method of Controller.
Debug log	When it is necessary to output debug information at the time of development, a suitable log output process is implemented in source code. Common library provides `org.terasoluna.gfw.web.logging.HttpSessionEventLoggingListener` listener which outputs DEBUG log at the time of HTTP session creation/destruction/attribute addition.
Access log	INFO log is output at the time of receiving a request and returning the response. It is usually implemented in AOP or Servlet filter.
External communication log	INFO log is output before and after external system linking.

次のページに続く

Table. 7.2 – 前のページからの続き

Category	Output points
Business error log	WARN log is output when business process exception is thrown. It is usually implemented in AOP. In common library, when *org.terasoluna.gfw.common.exception.BusinessException* is thrown at the time of business process execution, `org.terasoluna.gfw.common.exception.BusinessExceptionLoggingInterceptor` that outputs WARN log is provided. Refer to *Exception Handling* for details.
System error log	An ERROR log is output when system exception or unexpected exception occurs. It is usually implemented in AOP or Servlet filter. In common library, `org.terasoluna.gfw.web.exception.HandlerExceptionResolverLoggingIntercept` and `org.terasoluna.gfw.web.exception.ExceptionLoggingFilter` are provided. Refer to *Exception Handling* for details.
Monitoring log	It is output at the same time as business error log and system error log.

Note: Note that when the log is output, the contents should not be exactly identical to other logs. This is helpful in easily identifying the location of the output.

7.1.2 How to use

The following are required to output the log in SLF4J + Logback.

1. Settings of Logback
2. Calling API of SLF4J

Settings of Logback

Logback settings are described in logback.xml under the class path. An example of configuration is shown below. Refer to Logback Official Manual -Logback Configuration- for the detailed configuration of logback.xml.

Note: Settings of Logback are read automatically as per the rules given below.

1. logback.groovy on class path
2. If file "1" is not found, logback-test.xml on class path
3. If file "2" is not found, logback.xml on class path
4. If file "3" is not found, settings of class which implements `com.qos.logback.classic.spi.Configurator` interface (Specify a implementation class using ServiceLoader mechanism)
5. If class which implements `Configurator` interface is not found, settings of BasicConfigurator class (console output)

In this guideline, it is recommended to place logback.xml in the class path. Moreover, apart from automatic reading, it is possible to read programmatically through an APIand specify the configuration file in system properties.

logback.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>

    <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender"> <!-- (1) -
↪->
        <encoder>
            <pattern><![CDATA[date:%d{yyyy-MM-dd HH:mm:ss}\tthread:%thread\tX-
↪Track:%X{X-Track}\tlevel:%-5level\tlogger:%-48logger{48}\tmessage:%msg%n]]></
↪pattern> <!-- (2) -->
        </encoder>
    </appender>

    <appender name="APPLICATION_LOG_FILE" class="ch.qos.logback.core.rolling.
↪RollingFileAppender"> <!-- (3) -->
        <file>${app.log.dir:-log}/projectName-application.log</file> <!-- (4) -->
        <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
            <fileNamePattern>${app.log.dir:-log}/projectName-application-%d
↪{yyyyMMddHH}.log</fileNamePattern> <!-- (5) -->
            <maxHistory>7</maxHistory> <!-- (6) -->
        </rollingPolicy>
        <encoder>
            <charset>UTF-8</charset> <!-- (7) -->
            <pattern><![CDATA[date:%d{yyyy-MM-dd HH:mm:ss}\tthread:%thread\tX-
↪Track:%X{X-Track}\tlevel:%-5level\tlogger:%-48logger{48}\tmessage:%msg%n]]></
↪pattern>
        </encoder>
    </appender>

    <appender name="MONITORING_LOG_FILE" class="ch.qos.logback.core.rolling.
↪RollingFileAppender"> <!-- (8) -->
        <file>${app.log.dir:-log}/projectName-monitoring.log</file>
        <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
            <fileNamePattern>${app.log.dir:-log}/projectName-monitoring-%d
↪{yyyyMMdd}.log</fileNamePattern>
            <maxHistory>7</maxHistory>
        </rollingPolicy>
        <encoder>
            <charset>UTF-8</charset>
            <pattern><![CDATA[date:%d{yyyy-MM-dd HH:mm:ss}\tX-Track:%X{X-Track}
↪\tlevel:%-5level\tmessage:%msg%n]]></pattern>
        </encoder>
    </appender>

    <!-- Application Loggers -->
    <logger name="com.example.sample"> <!-- (9) -->
        <level value="debug" />
    </logger>

    <!-- TERASOLUNA -->
    <logger name="org.terasoluna.gfw">
```

```
        <level value="info" />
    </logger>
    <logger name="org.terasoluna.gfw.web.logging.TraceLoggingInterceptor">
        <level value="trace" />
    </logger>
    <logger name="org.terasoluna.gfw.common.exception.ExceptionLogger">
        <level value="info" />
    </logger>
    <logger name="org.terasoluna.gfw.common.exception.ExceptionLogger.Monitoring"␣
↪additivity="false"><!-- (10) -->
        <level value="error" />
        <appender-ref ref="MONITORING_LOG_FILE" />
    </logger>

    <!-- 3rdparty Loggers -->
    <logger name="org.springframework">
        <level value="warn" />
    </logger>

    <logger name="org.springframework.web.servlet">
        <level value="info" />
    </logger>

    <!--  REMOVE THIS LINE IF YOU USE JPA
    <logger name="org.hibernate.engine.transaction">
        <level value="debug" />
    </logger>
          REMOVE THIS LINE IF YOU USE JPA  -->
    <!--  REMOVE THIS LINE IF YOU USE MyBatis3
    <logger name="org.springframework.jdbc.datasource.DataSourceTransactionManager
↪">
        <level value="debug" />
    </logger>
          REMOVE THIS LINE IF YOU USE MyBatis3  -->

    <logger name="jdbc.sqltiming">
        <level value="debug" />
    </logger>

    <!-- only for development -->
    <logger name="jdbc.resultsettable">
        <level value="debug" />
    </logger>

    <root level="warn"> <!-- (11) -->
        <appender-ref ref="STDOUT" /> <!-- (12) -->
        <appender-ref ref="APPLICATION_LOG_FILE" />
    </root>

</configuration>
```

Sr. No.	Description
(1)	Definition of appender is specified to output the log on console. It can be selected whether the output destination is standard output or standard error, however when the destination is not specified, it is considered as standard output.
(2)	The output format for log is specified. If the format is not specified, the message alone is output. Time and message level etc. are output according to the business requirements. Here, LTSV (Labeled Tab Separated Value) of “Label:Value<TAB>Label:Value<TAB>...” format is set.
(3)	Definition for appender is specified to output application log. The appender to be used can also be specified in <logger>, however, here it is referred to root (11) since application log is used by default. RollingFileAppender is commonly used at the time of application log output, however, FileAppender may also be used to implement log rotation using another functions such as logrotate.
(4)	A current file name (File name of log being output) is specified. It should be specified when it is necessary to specify a fixed file name. If <file>log file name</file> is not specified, it is output with the name in pattern (5).
(5)	Name of the file after rotation is specified. Usually, date or time format is widely used. Note that 24 hour clock is not set if HH is mistakenly set as hh.
(6)	The number of remaining files for which rotation is performed is specified.
(7)	A character code of log file is specified.
(8)	It is set so as to output the application log by default.

次のページに続く

Table. 7.3 – 前のページからの続き

Sr. No.	Description
(9)	The logger name is set so that logger under com.example.sample outputs the log above debug level.
(10)	A monitoring log is set. Refer to *Common Settings* of *Exception Handling*. **Warning: About additivity setting value** Specify `false`. If `true`(default value) is specified, the same log will be output by upper level logger (for example, root). Concretely speaking, the monitoring log will be output using three appenders(`MONITORING_LOG_FILE`, `STDOUT` and `APPLICATION_LOG_FILE`).
(11)	It is set such that logger without <logger> specification outputs the log of warn level or above.
(12)	It is set in such a way that ConsoleAppender, RollingFileAppender (application logs) are used by default.

Tip: About LTSV(Labeled Tab Separated Value)

LTSV is one of text data formats, and mainly used as the log format.

For log fomart, LTSV is easy to parse using some tools because it has following features.

- It's easy to split the field compared to other delimiters because tabs is used as field delimiters.
- Even if the field definition (changing the position of field or adding the field or removing the field) is changed, it does not affect to parsing because of including a label(name) in the field.

It is also one of features that there are that pasting on the Excel can format it with the least effort.

The following three items should be set in logback.xml.

Type	Overview
appender	"Location" and "Layout" to be used for output
root	Default "Appender" and the minimum "Log level" to be used for output
logger	"Which logger (package or class etc.)" is to be output at which minimum "log level"

In <appender> element, the "location" and the "layout" to be used for output are defined. It is not used at the time of the log output only by defining the appender. It is used for the first time when it is referred in <logger> element or <root> element. There are two attributes, namely, "name" and "class" and both are mandatory.

Attribute	Overview
name	Name of the appender. It is specified by appender-ref. Any name can be assigned as there is no restriction on assigning the name.
class	FQCN of appender implementation class.

The main appenders that are provided are shown below.

Appender	Overview
ConsoleAppender	Console output
FileAppender	File output
RollingFileAppender	File output (Rolling possible)
AsyncAppender	Asynchronous output. It is used for logging in processes with high performance requirement. (It is necessary to set the output destination in other Appender.)

Refer to Logback Official Manual -Appenders- for detailed Appender types.

Basic log output by calling API of SLF4J

Log is output by calling a method according to each log level of SLF4J logger(`org.slf4j.Logger`).

```
package com.example.sample.app.welcome;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

@Controller
public class HomeController {

    private static final Logger logger = LoggerFactory
            .getLogger(HomeController.class);   // (1)

    @RequestMapping(value = "/", method = { RequestMethod.GET,
            RequestMethod.POST })
    public String home(Model model) {
        logger.trace("This log is trace log."); // (2)
        logger.debug("This log is debug log."); // (3)
        logger.info("This log is info log.");   // (4)
        logger.warn("This log is warn log.");   // (5)
        logger.error("This log is error log."); // (6)
        return "welcome/home";
    }

}
```

Sr. No.	Description
(1)	`Logger` is generated from `org.slf4j.LoggerFactory`. If Class object is set as an argument of `getLogger`, the logger name acts as an FQCN of that class. In this example, the logger name is "com.example.sample.app.welcome.HomeController".
(2)	The log of TRACE level is output.
(3)	The log of DEBUG level is output.
(4)	The log of INFO level is output.
(5)	The log of WARN level is output.
(6)	The log of ERROR level is output.

Log output results are shown below. Log level of com.example.sample is DEBUG, hence TRACE log is not output.

```
date:2013-11-06 20:13:05    thread:tomcat-http--3 X-Track:
 ↪5844f073b7434b67a875cb85b131e686    level:DEBUG logger:com.example.sample.app.
 ↪welcome.HomeController    message:This log is debug log.
date:2013-11-06 20:13:05    thread:tomcat-http--3 X-Track:
 ↪5844f073b7434b67a875cb85b131e686    level:INFO  logger:com.example.sample.app.
 ↪welcome.HomeController    message:This log is info log.
date:2013-11-06 20:13:05    thread:tomcat-http--3 X-Track:
 ↪5844f073b7434b67a875cb85b131e686    level:WARN  logger:com.example.sample.app.
 ↪welcome.HomeController    message:This log is warn log.
date:2013-11-06 20:13:05    thread:tomcat-http--3 X-Track:
 ↪5844f073b7434b67a875cb85b131e686    level:ERROR logger:com.example.sample.app.
 ↪welcome.HomeController    message:This log is error log.
```

The description can be as given below when an argument is to be entered in placeholder of a log message.

```
int a = 1;
logger.debug("a={}", a);
String b = "bbb";
```

```
logger.debug("a={}, b={}", a, b);
```

The log given below is output.

```
date:2013-11-06 20:32:45    thread:tomcat-http--3   X-Track:
↪853aa701a401404a87342a574c69efbc    level:DEBUG logger:com.example.sample.app.
↪welcome.HomeController    message:a=1
date:2013-11-06 20:32:45    thread:tomcat-http--3   X-Track:
↪853aa701a401404a87342a574c69efbc    level:DEBUG logger:com.example.sample.app.
↪welcome.HomeController    message:a=1, b=bbb
```

Warning: Note that string concatenation such as `logger.debug("a=" + a + " ,b=" + b);` should not be carried out.

When the exception is to be caught, ERROR log (WARN log in some cases) is output as shown below. Error message and exception generated are passed to the log method.

```
public String home(Model model) {
    // omitted

    try {
        throwException();
    } catch (Exception e) {
        logger.error("Exception happened!", e);
        // omitted
    }
    // omitted
}

public void throwException() throws Exception {
    throw new Exception("Test Exception!");
}
```

Accordingly, stack trace of caused exception is output and the cause of the error can be easily analyzed.

```
date:2013-11-06 20:38:04    thread:tomcat-http--5   X-Track:
↪11d7dbdf64e44782822c5aea4fc4bb4f    level:ERROR logger:com.example.sample.app.
↪welcome.HomeController    message:Exception happend!
java.lang.Exception: Test Exception!
    at com.example.sample.app.welcome.HomeController.throwException(HomeController.
↪java:40) ~[HomeController.class:na]
    at com.example.sample.app.welcome.HomeController.home(HomeController.java:31) ~
↪[HomeController.class:na]
    at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method) ~[na:1.7.0_40]
    (omitted)
```

However, as shown below, when the exception that is caught is wrapped with other exception and is re-thrown at upper level, there is no need to output the log. This is because usually the error log is output at upper level.

```
try {
    throwException();
} catch (Exception e) {
    throw new SystemException("e.ex.fw.9001", e);
    // no need to log
}
```

Note: When cause exception is to be passed to a log method, a placeholder cannot be used. Only in this case, the message argument can be concatenated using a string.

```
try {
    throwException();
} catch (Exception e) {
    // NG => logger.error("Exception happend! [a={} , b={}]", e, a, b);
    logger.error("Exception happend! [a=" + a + " , b=" + b + "]", e);
    // omitted
}
```

Points to be noted for the description of log output

SLF4J Logger internally checks the log level and outputs actual log only for the required levels.

Therefore, the log level check as given below is basically not necessary.

```
if (logger.isDebugEnabled()) {
    logger.debug("This log is Debug.");
}

if (logger.isDebugEnabled()) {
    logger.debug("a={}", a);
}
```

However, the log level should be checked in the cases given below to prevent performance degradation.

1. When there are 3 or more arguments

   When arguments of log message are 3 or more, argument array should be passed in the API of SLF4J. Log level should be checked in order to to avoid the cost for generating an array and the array should be generated only when necessary.

```
if (logger.isDebugEnabled()) {
    logger.debug("a={}, b={}, c={}", new Object[] { a, b, c });
}
```

2. When it is necessary to call a method for creating an argument

   When it is necessary to call a method while creating an argument for the log message, the log level should be checked to avoid the method execution cost and the method should be executed only when necessary.

```
if (logger.isDebugEnabled()) {
    logger.debug("xxx={}", foo.getXxx());
}
```

7.1.3 How to extend

Log output specifications are individually defined based on monitoring products and requirements and a case which is individually implemented is assumed. Here, a couple of examples are described below.

1. Uniform management of log messages
2. Integration of output format of log messages

Uniform management of log messages

An implementation example which aims to improve maintenance through uniform management of log messages.

Log messages are managed uniformly by consolidating the log messages in another file like a property file and resolving messages at the time of log output.

A method is described as an implementation example wherein the messages corresponding to log ID are output in the property file to enable setting of log ID in the argument of log output method.

Note: A method wherein consolidation is done using enum of Java also exists as a method to manage Log ID and log message, however, a general method wherein the property file is used is introduced in this guideline.

In this implementation example

1. Logger wrapper class
2. Property file

are created.

Here, `LogIdBasedLogger`is considered as a Logger wrapper class and `log-messages.properties`is considered as a property file.

- *LogIdBasedLogger* (Logger wrapper class)

```
package com.example.sample.common.logger;

import java.text.MessageFormat;
import java.util.Arrays;
import java.util.Locale;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.context.NoSuchMessageException;
import org.springframework.context.support.ResourceBundleMessageSource;

public class LogIdBasedLogger {

    private static final String UNDEFINED_MESSAGE_FORMAT = "UNDEFINED-MESSAGE id:
↪{0} arg:{1}";   // (1)

    private static ResourceBundleMessageSource messageSource = new␣
↪ResourceBundleMessageSource();// (2)

    static {    // (3)
        messageSource.setDefaultEncoding("UTF-8");          // (4)
        messageSource.setBasenames("i18n/log-messages");    // (5)
    }

    private final Logger logger;

    private LogIdBasedLogger(Class<?> clazz) {
        logger = LoggerFactory.getLogger(clazz);            // (6)
    }

    public static LogIdBasedLogger getLogger(Class<?> clazz) {
        return new LogIdBasedLogger(clazz);
    }

    public boolean isDebugEnabled() {                       // (7)
        return logger.isDebugEnabled();
    }

    public void debug(String format, Object... args) {
        logger.debug(format, args);                         // (8)
    }

    public void info(String id, Object... args) {
```

```
        if (logger.isInfoEnabled()) {
            logger.info(createLogMessage(id, args));        // (9)
        }
    }

    public void warn(String id, Object... args) {
        if (logger.isWarnEnabled()) {
            logger.warn(createLogMessage(id, args));        // (9)
        }
    }

    public void error(String id, Object... args) {
        if (logger.isErrorEnabled()) {
            logger.error(createLogMessage(id, args));       // (9)
        }
    }

    public void trace(String id, Object... args) {
        if (logger.isTraceEnabled()) {
            logger.trace(createLogMessage(id, args));       // (9)
        }
    }

    public void warn(String id, Throwable t, Object... args) {
        if (logger.isWarnEnabled()) {
            logger.warn(createLogMessage(id, args), t);     // (9)
        }
    }

    public void error(String id, Throwable t, Object... args) {
        if (logger.isErrorEnabled()) {
            logger.error(createLogMessage(id, args), t);    // (9)
        }
    }

    private String createLogMessage(String id, Object... args) {
        return getMessage(id, args);
    }

    private String getMessage(String id, Object... args) {
        String message;
        try {
            message = messageSource.getMessage(id, args, Locale
                    .getDefault());
        } catch (NoSuchMessageException e) {                // (10)
            message = MessageFormat.format(UNDEFINED_MESSAGE_FORMAT, id, Arrays
                    .toString(args));
        }
        return message;
    }
```

```
}
```

Sr. No.	Description
(1)	A log message when the log ID is not defined. Here, a message same as `org.terasoluna.gfw.common.exception.ExceptionLogger`is used as an example.
(2)	An implementation example wherein the log message is fetched by `MessageSource`. `MessageSource`managed by message data is stored in `static`area to enhance general versatility. This implementation facilitates the use of Logger wrapper class at any time since dependence on accessibility to DI container is eliminated.
(3)	Generate `MessageSource`using static initializer. In this implementation, `log-messages.properties`placed in `i18n`is read.
(4)	Specify a character code to be used while parsing a property file. In this implementation, since property file is in UTF-8 encoding format, UTF-8 is specified. For details, refer *Display of messages set in properties*of *Message Management*.
(5)	Specify a property file by using `setBasenames` method considering internationalization. For details of `setBasenames`, refer Javadoc of setBasenames of ReloadableResourceBundleMessageSource class.
(6)	Use SLF4J in logger wrapper class as well. Logging library implementation is not used directly.
(7)	Determine whether log output of DEBUG level is allowed. For precautions for use, refer *Points to be noted for the description of log output*.
(8)	In this implementation example, log ID is not used in the log of DEBUG level. Log message of argument is output as it is.

次のページに続く

Table. 7.4 – 前のページからの続き

Sr. No.	Description
(9)	Output TRACE/INFO/WARN/ERROR level log by fetching the log message corresponding to log ID from the property file.
(10)	If log ID is not described in the property file while calling getMessage, an exception :`NoSuchMessageException`is generated. Hence, `NoSuchMessageException`is caught Aand a log message stating “log ID is not defined in the property file” is output.

- *log-messages.properties* (property file)

```
i.ab.cd.1001 = This message is Info-Level. {0}
w.ab.cd.2001 = This message is Warn-Level. {0}
e.ab.cd.3001 = This message is Error-Level. {0}
t.ab.cd.4001 = This message is Trace-Level. {0}
```

Note: In this guideline, since message for screen output and message for log output are managed separately, a new property file is created. However, both the messages can be output in a single file.

File unit should be determined in accordance with the nature of the application and how to manage a method of message.

Execution results are as below.

- Call sample

```
package com.example.sample.app.welcome;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

import com.example.sample.common.logger.LogIdBasedLogger;

@Controller
public class HomeController {

    private static final LogIdBasedLogger logger = LogIdBasedLogger
            .getLogger(HomeController.class);

    @RequestMapping(value = "/", method = { RequestMethod.GET,
            RequestMethod.POST })
    public String home(Model model) {
        logger.debug("debug log");
        logger.info("i.ab.cd.1001","replace_value_1");
        logger.warn("w.ab.cd.2001","replace_value_2");
        logger.error("e.ab.cd.3001","replace_value_3");
        logger.trace("t.ab.cd.4001","replace_value_4");
        logger.info("i.ab.cd.1002","replace_value_5");
        return "welcome/home";
    }
}
```

- Example of log output

```
date:2016-05-30 17:34:18.590  thread:http-bio-8080-exec-3  X-Track:
 ↪e2a65cd9160b48d6aaeb63fe6e751c6b  level:DEBUG  logger:com.example.sample.app.
 ↪welcome.HomeController   message:debug log
date:2016-05-30 17:34:18.590  thread:http-bio-8080-exec-3  X-Track:
 ↪e2a65cd9160b48d6aaeb63fe6e751c6b  level:INFO   logger:com.example.sample.app.
 ↪welcome.HomeController   message:This message is Info-Level. replace_value_1
date:2016-05-30 17:34:18.590  thread:http-bio-8080-exec-3  X-Track:
 ↪e2a65cd9160b48d6aaeb63fe6e751c6b  level:WARN   logger:com.example.sample.app.
 ↪welcome.HomeController   message:This message is Warn-Level. replace_value_2
date:2016-05-30 17:34:18.590  thread:http-bio-8080-exec-3  X-Track:
 ↪e2a65cd9160b48d6aaeb63fe6e751c6b  level:ERROR  logger:com.example.sample.app.
 ↪welcome.HomeController   message:This message is Error-Level. replace_value_3
date:2016-05-30 17:34:18.590  thread:http-bio-8080-exec-3  X-Track:
 ↪e2a65cd9160b48d6aaeb63fe6e751c6b  level:TRACE  logger:com.example.sample.app.
 ↪welcome.HomeController   message:This message is Trace-Level. replace_value_4
date:2016-05-30 17:34:18.590  thread:http-bio-8080-exec-3  X-Track:
 ↪e2a65cd9160b48d6aaeb63fe6e751c6b  level:INFO   logger:com.example.sample.app.
 ↪welcome.HomeController   message:UNDEFINED-MESSAGE id:i.ab.cd.1002 arg:[replace_
 ↪value_5]
```

Integration of output format of log messages

Output format of log messages differs according to log output format as given in the table below..

Hence, log output format must be combined with another format or both the formats must be integrated with individual format in the integration of log format.

In this guideline, an example wherein the format is defined in the log output by business logic and an example wherein both are integrated with the individual format ({Exception code (message ID) or log ID}], {message or log message}) are explained.

Sr. No.	Log output format	Corresponding log	Default format
(1)	Explicitly output log using business logic	Access log, external communication log etc	Nil
(2)	Framework detects the exception and implicitly outputs the log	Business error log, system error log etc	[{Exception code (Message ID)}] {Message}

Note: “Business error log” and “system error log” output at the time of generating an exception are output in the default format of the table given above in accordance with the exception handling system of *Common library*.

Integration with the format of the log output by framework after detecting an exception

An implementation example is shown wherein the log output by business logic is coordinated with the format of the log output by framework after detecting an exception.

In this guideline, it is implemented by adding a process to apply format, in Logger wrapper class (`LogIdBasedLogger`).

```
package com.example.sample.common.logger;

import java.text.MessageFormat; // (1)

// omitted

public class LogIdBasedLogger {

    private static final String LOG_MESSAGE_FORMAT = "[{0}] {1}"; // (2)

    // omitted

    private String createLogMessage(String id, String... args) {
        return MessageFormat.format(LOG_MESSAGE_FORMAT, id, getMessage(id,
                args)); // (1)
    }

    // omitted

}
```

Sr.No.	Description
(1)	Add a process wherein a log message is created based on log message format.
(2)	Define a format. Log ID replaces `{0}`and log message replaces `{1}`.

Execution results are as below.

```
date:2016-05-30 16:32:33.239  thread:http-bio-8080-exec-4  X-Track:
 ↪4f61314a51524ab3a41832b0ceae7119  level:DEBUG  logger:com.example.sample.app.
 ↪welcome.HomeController   message:debug log
date:2016-05-30 16:32:33.239  thread:http-bio-8080-exec-4  X-Track:
 ↪4f61314a51524ab3a41832b0ceae7119  level:INFO   logger:com.example.sample.app.
 ↪welcome.HomeController   message:[i.ab.cd.1001] This message is Info-Level.␣
 ↪replace_value_1
```

```
date:2016-05-30 16:32:33.239  thread:http-bio-8080-exec-4  X-Track:
↪4f61314a51524ab3a41832b0ceae7119  level:WARN  logger:com.example.sample.app.
↪welcome.HomeController  message:[w.ab.cd.2001] This message is Warn-Level.␣
↪replace_value_2
date:2016-05-30 16:32:33.239  thread:http-bio-8080-exec-4  X-Track:
↪4f61314a51524ab3a41832b0ceae7119  level:ERROR  logger:com.example.sample.app.
↪welcome.HomeController  message:[e.ab.cd.3001] This message is Error-Level.␣
↪replace_value_3
date:2016-05-30 17:34:18.590  thread:http-bio-8080-exec-3  X-Track:
↪4f61314a51524ab3a41832b0ceae7119  level:TRACE  logger:com.example.sample.app.
↪welcome.HomeController  message:[t.ab.cd.4001] This message is Trace-Level.␣
↪replace_value_4
date:2016-05-30 16:32:33.239  thread:http-bio-8080-exec-4  X-Track:
↪4f61314a51524ab3a41832b0ceae7119  level:INFO  logger:com.example.sample.app.
↪welcome.HomeController  message:[i.ab.cd.1002] UNDEFINED-MESSAGE id:i.ab.cd.
↪1002 arg:[replace_value_5]
```

Integration to a unique format

An implementation example wherein the log output by business logic and framework is integrated in a unique format ([{Exception code (message ID) or log ID}], {message or log message}).

Define a format in the log which is output by business logic

An example wherein log output by business logic is output in the format mentioned earlier.

In this guideline, it is implemented by adding a process to apply the format, to Logger wrapper class (`LogIdBasedLogger`).

```
package com.example.sample.common.logger;

import java.text.MessageFormat; // (1)

// omitted

public class LogIdBasedLogger {

    private static final String LOG_MESSAGE_FORMAT = "[{0}], {1}"; // (2)

    // omitted

    private String createLogMessage(String id, String... args) {
        return MessageFormat.format(LOG_MESSAGE_FORMAT, id, getMessage(id,
                args)); // (1)
    }
```

```
    // omitted

}
```

Sr. No.	Description
(1)	Add a process to create log message based on log message format.
(2)	Define a format. ID replaces `{0}`and log message replaces `{1}`.

Execution results are as below.

```
date:2016-05-30 16:32:33.239  thread:http-bio-8080-exec-4  X-Track:
 ↪4f61314a51524ab3a41832b0ceae7119  level:DEBUG  logger:com.example.sample.app.
 ↪welcome.HomeController   message:debug log
date:2016-05-30 16:32:33.239  thread:http-bio-8080-exec-4  X-Track:
 ↪4f61314a51524ab3a41832b0ceae7119  level:INFO   logger:com.example.sample.app.
 ↪welcome.HomeController   message:[i.ab.cd.1001], This message is Info-Level.␣
 ↪replace_value_1
date:2016-05-30 16:32:33.239  thread:http-bio-8080-exec-4  X-Track:
 ↪4f61314a51524ab3a41832b0ceae7119  level:WARN   logger:com.example.sample.app.
 ↪welcome.HomeController   message:[w.ab.cd.2001], This message is Warn-Level.␣
 ↪replace_value_2
date:2016-05-30 16:32:33.239  thread:http-bio-8080-exec-4  X-Track:
 ↪4f61314a51524ab3a41832b0ceae7119  level:ERROR  logger:com.example.sample.app.
 ↪welcome.HomeController   message:[e.ab.cd.3001], This message is Error-Level.␣
 ↪replace_value_3
date:2016-05-30 17:34:18.590  thread:http-bio-8080-exec-3  X-Track:
 ↪4f61314a51524ab3a41832b0ceae7119  level:TRACE  logger:com.example.sample.app.
 ↪welcome.HomeController   message:[t.ab.cd.4001], This message is Trace-Level.␣
 ↪replace_value_4
date:2016-05-30 16:32:33.239  thread:http-bio-8080-exec-4  X-Track:
 ↪4f61314a51524ab3a41832b0ceae7119  level:INFO   logger:com.example.sample.app.
 ↪welcome.HomeController   message:[i.ab.cd.1002], UNDEFINED-MESSAGE arg:[replace_
 ↪value_5]
```

Changing format of the log output by framework

An example wherein the log output by framework is output in format mentioned earlier.

Bean definition of `ExceptionLogger`of `applicationContext.xml`is changed to change the format of business error log and system error log.

An example of how to define `ExceptionLogger`is given below.

- **applicationContext.xml**

```
<!-- Exception Logger. -->
<bean id="exceptionLogger"
    class="org.terasoluna.gfw.common.exception.ExceptionLogger">
    <property name="exceptionCodeResolver" ref="exceptionCodeResolver" />
    <property name="logMessageFormat" value="[{0}], {1}" />    <!-- (1) -->
</bean>
```

Sr. No.	Description
(1)	Define a format in `logMessageFormat`. Exception code (message ID) replaces `{0}`and message replaces `{1}`.

Execution results are as below..

```
date:2013-09-19 21:03:06   thread:tomcat-http--3   X-Track:
↪c19eec546b054d54a13658f94292b24f    level:ERROR logger:o.t.gfw.common.exception.
↪ExceptionLogger         message:[e.ad.od.9012], not found item entity. item code␣
↪[10-123456].
...
// stackTarace omitted
```

7.1.4 Appendix

Using MDC

A cross-sectional log can be output by using MDC (Mapped Diagnostic Context).

Log traceability improves if same information (such as user name or unique request ID) is included in the log to be output in a request.

MDC internally consists of a ThreadLocal map and sets value for the key. The value set in log can be output till it is removed.

The value should be set at the beginning of the request and removed at the time of process termination.

Basic usage method

An example of using MDC is given below.

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.slf4j.MDC;

public class Main {

    private static final Logger logger = LoggerFactory.getLogger(Main.class);

    public static void main(String[] args) {
        String key = "MDC_SAMPLE";
        MDC.put(key, "sample"); // (1)
        try {
            logger.debug("debug log");
            logger.info("info log");
            logger.warn("warn log");
            logger.error("error log");
        } finally {
            MDC.remove(key); // (2)
        }
        logger.debug("mdc removed!");
    }
}
```

The value added to MDC can be output in log by defining the output format as `%X{key name}` format in `<pattern>` of logback.xml.

```
<appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
    <encoder>
        <pattern><![CDATA[date:%d{yyyy-MM-dd HH:mm:ss}\tthread:%thread\tmdcSample:
↪%X{MDC_SAMPLE}\tlevel:%-5level\t\tmessage:%msg%n]]></pattern>
    </encoder>
</appender>
```

Execution results are as follows:

```
date:2013-11-08 17:45:48    thread:main mdcSample:sample    level:DEBUG    ␣
↪message:debug log
date:2013-11-08 17:45:48    thread:main mdcSample:sample    level:INFO     ␣
↪message:info log
date:2013-11-08 17:45:48    thread:main mdcSample:sample    level:WARN     ␣
↪message:warn log
date:2013-11-08 17:45:48    thread:main mdcSample:sample    level:ERROR    ␣
↪message:error log
date:2013-11-08 17:45:48    thread:main mdcSample:  level:DEBUG     message:mdc␣
↪removed!
```

Note: If `MDC.clear()` is executed, all the added values are deleted.

Setting value in MDC using Filter

Common library provides `org.terasoluna.gfw.web.logging.mdc.AbstractMDCPutFilter` as a base class to add/delete values from MDC using filter.
Further, following are provided as implementation classes.

- `org.terasoluna.gfw.web.logging.mdc.XTrackMDCPutFilter` to set an unique ID for each request in MDC
- `org.terasoluna.gfw.security.web.logging.UserIdMDCPutFilter` to set authentication user name of Spring Security in MDC

If individual values are to be added to MDC using Filter, it is desirable to implement `AbstractMDCPutFilter`
based on implementation of `org.terasoluna.gfw.web.logging.mdc.XTrackMDCPutFilter`.

How to use MDCFilter

Definition of MDCFilter is added to the filter definition of web.xml.

```
<!-- omitted -->

<!-- (1) -->
<filter>
    <filter-name>MDCClearFilter</filter-name>
    <filter-class>org.terasoluna.gfw.web.logging.mdc.MDCClearFilter</filter-class>
</filter>

<filter-mapping>
    <filter-name>MDCClearFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>

<!-- (2) -->
<filter>
    <filter-name>XTrackMDCPutFilter</filter-name>
    <filter-class>org.terasoluna.gfw.web.logging.mdc.XTrackMDCPutFilter</filter-
↪class>
</filter>
```

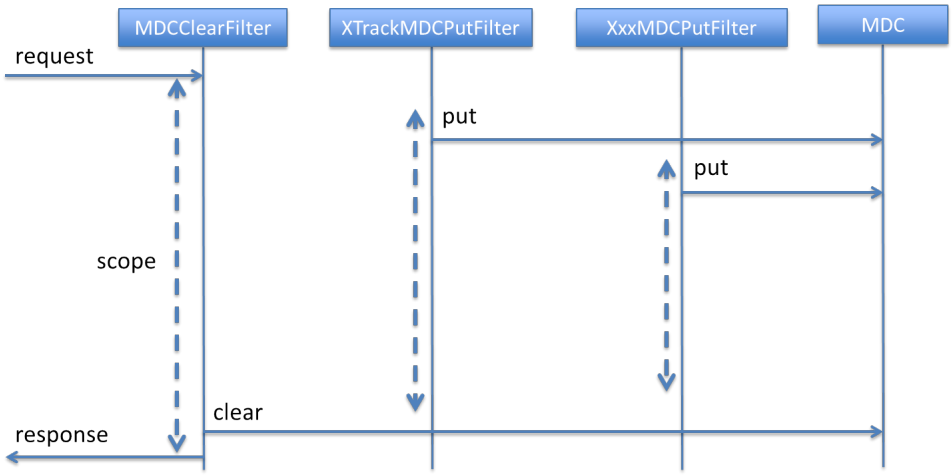
```
<filter-mapping>
    <filter-name>XTrackMDCPutFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>

<!-- (3) -->
<filter>
    <filter-name>UserIdMDCPutFilter</filter-name>
    <filter-class>org.terasoluna.gfw.security.web.logging.UserIdMDCPutFilter</
↪filter-class>
</filter>
<filter-mapping>
    <filter-name>UserIdMDCPutFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>

<!-- omitted -->
```

Sr. No.	Description
(1)	`MDCClearFilter` that clears the contents of MDC is set. Values in MDC added by various `MDCPutFilter` are deleted by this Filter.
(2)	`XTrackMDCPutFilter` is set. `XTrackMDCPutFilter` sets Request ID in key "X-Track".
(3)	`UserIdMDCPutFilter` is set. `UserIdMDCPutFilter` sets User ID in key "USER".

As shown in the sequence diagram below, `MDCClearFilter` should be defined prior to each `MDCPutFilter`to clear the contents of MDC for post-processing.

Request ID and User ID can be output to log by adding `%X{X-Track}` and `%X{USER}` in `<pattern>` of logback.xml.

```
<!-- omitted -->
<appender name="APPLICATION_LOG_FILE" class="ch.qos.logback.core.rolling.
↪RollingFileAppender">
    <file>${app.log.dir:-log}/projectName-application.log</file>
    <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
        <fileNamePattern>${app.log.dir:-log}/projectName-application-%d{yyyyMMdd}.
↪log</fileNamePattern>
        <maxHistory>7</maxHistory>
    </rollingPolicy>
```




```
    <encoder>
        <charset>UTF-8</charset>
        <pattern><![CDATA[date:%d{yyyy-MM-dd HH:mm:ss}\tthread:%thread\tUSER:%X
↪{USER}\tX-Track:%X{X-Track}\tlevel:%-5level\tlogger:%-48logger{48}\tmessage:%msg
↪%n]]></pattern>
    </encoder>
</appender>
<!-- omitted -->
```

Example of log output

```
date:2013-09-06 23:05:22  thread:tomcat-http--3   USER:   X-Track:
↪97988cc077f94f9d9d435f6f76027428    level:DEBUG logger:o.t.g.w.logging.
↪HttpSessionEventLoggingListener  message:SESSIONID
↪#D7AD1D42D3E77D61DB64E7C8C65CB488 sessionCreated : org.apache.catalina.session.
↪StandardSessionFacade@e51960
date:2013-09-06 23:05:22  thread:tomcat-http--3   USER:anonymousUser  X-Track:
↪97988cc077f94f9d9d435f6f76027428    logger:o.t.gfw.web.logging.
↪TraceLoggingInterceptor      message:[START CONTROLLER] HomeController.
↪home(Locale,Model)
date:2013-09-06 23:05:22  thread:tomcat-http--3   USER:anonymousUser  X-Track:
↪97988cc077f94f9d9d435f6f76027428    level:INFO  logger:c.terasoluna.logging.app.
↪welcome.HomeController  message:Welcome home! The client locale is ja.
date:2013-09-06 23:05:22  thread:tomcat-http--3   USER:anonymousUser  X-Track:
↪97988cc077f94f9d9d435f6f76027428    logger:o.t.gfw.web.logging.
↪TraceLoggingInterceptor      message:[END CONTROLLER  ] HomeController.
↪home(Locale,Model)-> view=home, model={serverTime=2013/09/06 23:05:22 JST}
date:2013-09-06 23:05:22  thread:tomcat-http--3   USER:anonymousUser  X-Track:
↪97988cc077f94f9d9d435f6f76027428    logger:o.t.gfw.web.logging.
↪TraceLoggingInterceptor      message:[HANDLING TIME   ] HomeController.
↪home(Locale,Model)-> 36,508,860 ns
```

Note: User information to be set in MDC by `UserIdMDCPutFilter` is created by Spring Security Filter. As mentioned earlier, if `UserIdMDCPutFilter` is defined in web.xml, user ID is output to log after the completion of a series of Spring Security processes. If user information is to be output to the log immediately after it is generated, the information should be incorporated in Spring Security Filter as shown below by deleting web.xml definition.

The definitions given below are added to spring-security.xml.

```
<sec:http>
    <!-- omitted -->
    <sec:custom-filter ref="userIdMDCPutFilter" after="ANONYMOUS_FILTER"/>
↪<!-- (1) -->
    <!-- omitted -->
</sec:http>

<!-- (2) -->
<bean id="userIdMDCPutFilter" class="org.terasoluna.gfw.security.web.
↪logging.UserIdMDCPutFilter">
</bean>
```

Sr. No.	Description
(1)	`UserIdMDCPutFilter` defined in Bean is added after “ANONYMOUS_FILTER”.
(2)	`UserIdMDCPutFilter` is defined.

In blank project, `UserIdMDCPutFilter` is defined in spring-security.xml.

Log output related functionalities provided by common library

HttpSessionEventLoggingListener

`org.terasoluna.gfw.web.logging.HttpSessionEventLoggingListener` is a listener class that outputs debug log at the time of generating, discarding, activating or deactivating session, and adding or deleting session attributes.

The following should be added to web.xml.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/javaee" xmlns:xsi="http://www.w3.org/
↪2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/
↪ns/javaee/web-app_3_0.xsd"
```

```
    version="3.0">
    <listener>
        <listener-class>org.terasoluna.gfw.web.logging.
↪HttpSessionEventLoggingListener</listener-class>
    </listener>

    <!-- omitted -->
</web-app>
```

In logback.xml, `org.terasoluna.gfw.web.logging.HttpSessionEventLoggingListener` is set at debug level as shown below.

```
<logger
    name="org.terasoluna.gfw.web.logging.HttpSessionEventLoggingListener"> <!--␣
↪(1) -->
    <level value="debug" />
</logger>
```

Debug log as shown below is output.

```
date:2013-09-06 16:41:33    thread:tomcat-http--3   USER:   X-Track:
↪c004ddb56a3642d5bc5f6b5d884e5db2        level:DEBUG     logger:o.t.g.w.logging.
↪HttpSessionEventLoggingListener  message:SESSIONID
↪#EDC3C240A7A1CCE87146A6BA1321AD0F sessionCreated : org.apache.catalina.session.
↪StandardSessionFacade@f00e0f
```

When the lifecycle of an object is managed using a Session such as `@SessionAttributes` etc., it is strongly recommended to use this listener to confirm whether the attributes added to the session are deleted as anticipated.

TraceLoggingInterceptor

`org.terasoluna.gfw.web.logging.TraceLoggingInterceptor` is the `HandlerInterceptor` to output start and termination of Controller process to the log. When the process is terminated, View name returned by the Controller, attributes added to Model and the time required for Controller process are also output.

`TraceLoggingInterceptor`is added in `<mvc:interceptors>` of spring-mvc.xml as shown below.

```
<mvc:interceptors>
    <!-- omitted -->
    <mvc:interceptor>
        <mvc:mapping path="/**" />
        <mvc:exclude-mapping path="/resources/**" />
        <bean
            class="org.terasoluna.gfw.web.logging.TraceLoggingInterceptor">
        </bean>
    </mvc:interceptor>
```

```
    <!-- omitted -->
</mvc:interceptors>
```

By default, WARN log is output if Controller process takes more than 3 seconds.

When the threshold value is to be changed, it is specified in nano seconds in `warnHandlingNanos` property.

The following settings should be performed if the threshold value is to be changed to 10 seconds (10 * 1000 * 1000 * 1000 nano seconds).

```
<mvc:interceptors>
    <!-- omitted -->
    <mvc:interceptor>
        <mvc:mapping path="/**" />
        <mvc:exclude-mapping path="/resources/**" />
        <bean
            class="org.terasoluna.gfw.web.logging.TraceLoggingInterceptor">
            <property name="warnHandlingNanos" value="#{10 * 1000 * 1000 * 1000}" /
↪>
        </bean>
    </mvc:interceptor>
    <!-- omitted -->
</mvc:interceptors>
```

In logback.xml, `org.terasoluna.gfw.web.logging.TraceLoggingInterceptor` is set at trace level as shown below.

```
<logger name="org.terasoluna.gfw.web.logging.TraceLoggingInterceptor"> <!-- (1) -->
    <level value="trace" />
</logger>
```

ExceptionLogger

`org.terasoluna.gfw.common.exception.ExceptionLogger` is provided as a logger when an exception occurs.

Refer to *How to use* of *Exception Handling* for the usage method.

7.2 Properties Management

7.2.1 Overview

This section explains how to manage properties.

Value that needs to be managed as properties can be classified into following two categories.

Sr. No.	Classification	Description	Example
1.	Environment dependent setting	The setting needs to be changed according to the environment on which the application is running. It depends on non-functional requirements such as system configuration.	• Database connection information (connection URL, connection user, password, etc) • Destination of the file storage (directory path etc) • more ...
2.	Application settings	The settings that can be customize the application behavior. It depends on functional requirements.	• Password valid days • Reservation period days • more ...

Note: In this guideline, it is recommended to manage these settings as properties (properties file).

If an application is mechanized such a way that acquires setting from the properties, there is no need to re-build the application even if there is any changes in these values. Therefore it is possible to release the tested application on production environment.

Tip: Values that are managed as properties can be acquired from JVM system properties (-D option) or OS environment variables. About access order, refer to "*How to use*".

Values that are managed as properties can be used at the following two locations.

- Bean definition file
- Java classes managed by DI container

7.2.2 How to use

About properties file definition

Properties file values in Java class and bean definition file can be accessed by defining
`<context:property-placeholder/>` tag in bean definition file.
`<context:property-placeholder/>` tag reads the group of specified properties files
and can fetch values for properties files key `xxx` specified in `${xxx}` format in `@Value` annotation or bean definition files.

> **Note:** When specified in `${xxx:defaultValue}`format and when key `xxx` is not set in properties file, `defaultValue` is used.

See the method below for defining a properties file

bean definition file

- applicationContext.xml
- spring-mvc.xml

```
<context:property-placeholder location="classpath*:META-INF/spring/*.
↪properties"/>  <!-- (1) -->
```

Sr. No.	Description
(1)	In location, set the resource location path. Multiple paths separated by comma can be specified in location attribute. By performing above settings, read properties file under META-INF/spring directory of class path. Once these settings are done, just add the properties file under META-INF/spring. For details on location value, see Reference.

Note: `<context:property-placeholder>` needs to be defined in both `applicationContext.xml` and `spring-mvc.xml`.

Properties are accessed in the following order by default.

1. System properties of active JVM
2. Environment variables
3. Application definition properties

As per default setting, properties file defined in application is searched and read after all environment related properties (JVM system properties and environment variables) are read.

Read sequence can be changed by setting local-override attribute of `<context:property-placeholder/>` tag to true.

By performing these settings, the properties defined in application are enabled with higher priority.

bean definition file

```
<context:property-placeholder
    location="classpath*:META-INF/spring/*.properties"
    local-override="true" /> <!-- (1) -->
```

Sr. No.	Description
(1)	Access properties in the following order when local-override attribute is set to true. 1. Application definition properties 2. System properties of active JVM 3. Environment variables

Note: Normally the above settings are sufficient. When multiple `<context:property-placeholder/>` tags are specified, read order can be defined by setting order attribute value.

bean definition file

```
<context:property-placeholder
    location="classpath:/META-INF/property/extendPropertySources.
↪properties"
    order="1" ignore-unresolvable="true" /> <!-- (1) -->
<context:property-placeholder
    location="classpath*:/META-INF/spring/*.properties"
    order="2" ignore-unresolvable="true" /> <!-- (2) -->
```

Sr. No.	Description
(1)	By setting the order attribute to a value which is less than (2), properties file corresponding to location attribute is read before (2). When a key overlapping with the key in properties file read in (2) exists, value fetched in (1) is given preference. By setting ignore-unresolvable attribute to true, error which occurs when key exists only in properties file of (2) can be prevented.
(2)	By setting the order attribute to value greater than (1), properties file corresponding to location attribute is read after (1). When a key overlapping with the key in properties file read in (1) exists, value fetched in (1) is set. By setting ignore-unresolvable attribute to true, error which occurs when key exists only in properties file of (1) can be prevented.

Using properties in bean definition file

See the example below of datasource configuration file.

In the following example, it is assumed that properties file definition (

`<context:property-placeholder/>`) is specified.

Basically, property value can be set by setting properties file key in bean definition file using `${}` placeholder.

Properties file

```
database.url=jdbc:postgresql://localhost:5432/shopping
database.password=postgres
database.username=postgres
database.driverClassName=org.postgresql.Driver
```

bean definition file

```
<bean id="dataSource"
    destroy-method="close"
    class="org.apache.commons.dbcp2.BasicDataSource">
    <property name="driverClassName"
              value="${database.driverClassName}"/>  <!-- (1) -->
    <property name="url" value="${database.url}"/>  <!-- (2) -->
    <property name="username" value="${database.username}"/>  <!-- (3) -->
    <property name="password" value="${database.password}"/>  <!-- (4) -->
    <!-- omitted -->
</bean>
```

Sr. No.	Description
(1)	By setting `${database.driverClassName}`, the value for read properties file key `database.driverClassName` gets substituted.
(2)	By setting `${database.url}`, the value for read properties file key `database.url` gets substituted.
(3)	By setting `${database.username}`, the value for read properties file key `database.username` gets substituted.
(4)	By setting `${database.password}`, the value for read properties file key `database.password` gets substituted.

As a result of reading the properties file key, the values are replaced as follows:

```
<bean id="dataSource"
    destroy-method="close"
    class="org.apache.commons.dbcp2.BasicDataSource">
    <property name="driverClassName" value="org.postgresql.Driver"/>
    <property name="url"
              value="jdbc:postgresql://localhost:5432/shopping"/>
    <property name="username" value="postgres"/>
    <property name="password" value="postgres"/>
    <!-- omitted -->
</bean>
```

Using properties in Java class

It is possible to use properties in Java class by specifying `@Value` annotation in the field wherein properties values are to be stored.

To use `@Value` annotation, the corresponding object needs to be stored in DI container of Spring.

In the following example, it is assumed that properties file definition (`<context:property-placeholder/>`) is specified.

External reference is possible by adding `@Value` annotation to variables and setting properties file key in value using `${}` placeholder.

Properties file

```
item.upload.title=list of update file
item.upload.dir=file:/tmp/upload
item.upload.maxUpdateFileNum=10
```

Java class

```
@Value("${item.upload.title}")  // (1)
private String uploadTitle;

@Value("${item.upload.dir}")  // (2)
private Resource uploadDir;

@Value("${item.upload.maxUpdateFileNum}")  // (3)
private int maxUpdateFileNum;

// Getters and setters omitted
```

Sr. No.	Description
(1)	By setting `${item.upload.title}` to `@Value` annotation value, the value for read properties file key `item.upload.title` gets substituted. `uploadTitle` is substituted by "list of update file" in String class.
(2)	By setting `${item.upload.dir}` to `@Value` annotation value, the value for read properties file key `item.upload.dir` gets substituted. `org.springframework.core.io.Resource` object created with initial value "/tmp/upload" is stored in `uploadDir`.
(3)	By setting `${item.upload.maxUpdateFileNum}` to `@Value` annotation value, the value for read properties file key `item.upload.maxUpdateFileNum` gets substituted. `maxUpdateFileNum` is substituted by 10.

Warning: There could be cases wherein properties values are to be used in static methods of Utility classes etc.; however properties value cannot be fetched using `@Value` annotation in classes for which bean definition is not done. In this case, it is recommended to create Helper class with `@Component` annotation and to fetch properties values using `@Value` annotation. (This class needs to be included in the component-scan scope.) Classes in which values from properties file is to be used, should not be made Utility classes.

7.2.3 How to extend

Extension of method for fetching properties values is explained below. This can be achieved by extending `org.springframework.context.support.PropertySourcesPlaceholderConfigurer` class.

The example below illustrates a case wherein encrypted properties file is used.

Decrypting encrypted values and using them

To strengthen security, properties file needs to be encrypted in some cases.

The example below illustrates decryption of encrypted properties values. (No specific encrypting and decrypting methods are mentioned.)

bean definition file

- applicationContext.xml
- spring-mvc.xml

```
<!-- (1) -->
<bean class="com.example.common.property.
↪EncryptedPropertySourcesPlaceholderConfigurer">
    <!-- (2) -->
    <property name="locations"
              value="classpath*:/META-INF/spring/*.properties" />
</bean>
```

Sr. No.	Description
(1)	Define the extended PropertySourcesPlaceholderConfigurer instead of `<context:property-placeholder/>`. `<context:property-placeholder/>` tag should be deleted.
(2)	Set “locations” in name attribute of property tag and specify the path of the properties file to be read, in value attribute. The method of specifying path of the properties file to be read is same as *About properties file definition*.

Java class

- Extended PropertySourcesPlaceholderConfigurer

```
public class EncryptedPropertySourcesPlaceholderConfigurer extends
    PropertySourcesPlaceholderConfigurer { // (1)
    @Override
    protected void doProcessProperties(
            ConfigurableListableBeanFactory beanFactoryToProcess,
            StringValueResolver valueResolver) { // (2)
        super.doProcessProperties(beanFactoryToProcess,
            new EncryptedValueResolver(valueResolver)); // (3)
```

```
    }
}
```

Sr. No.	Description
(1)	Inherited PropertySourcesPlaceholderConfigurer, should extend `org.springframework.context.support.PropertySourcesPlaceholderConfigurer` class.
(2)	Override `doProcessProperties` method of `org.springframework.context.support.PropertySourcesPlaceholderConfigurer` class.
(3)	Call `doProcessProperties` of parent class; however, use valueResolver(`EncryptedValueResolver`) `valueResolver` wherein `valueResolver` is implemented independently. In `EncryptedValueResolver` class, decrypt when encrypted values of properties file are fetched.

- EncryptedValueResolver.java

```
public class EncryptedValueResolver implements
                                    StringValueResolver { // (1)

    private final StringValueResolver valueResolver;

    EncryptedValueResolver(StringValueResolver stringValueResolver) { // (2)
        this.valueResolver = stringValueResolver;
    }

    @Override
    public String resolveStringValue(String strVal) { // (3)

        // Values obtained from the property file to the naming
        // as seen with the encryption target
        String value = valueResolver.resolveStringValue(strVal); // (4)
```

```
        // Target messages only, implement coding
        if (value.startsWith("Encrypted:")) { // (5)
            value =  value.substring(10); // (6)
            // omitted decryption
        }
        return value;
    }
}
```

Sr. No.	Description
(1)	Inherited `EncryptedValueResolver` should implement `org.springframework.util.StringValueResolver`.
(2)	When `EncryptedValueResolver` class is created in constructor, set `StringValueResolver` inherited from `EncryptedPropertySourcesPlaceholderConfigurer`.
(3)	Override `resolveStringValue` method of `org.springframework.util.StringValueResolver`. If the values fetched from properties file are encrypted, these must be decrypted in `resolveStringValue` method. The process mentioned in steps (5) and (6) is just an example, the process differs depending on type of implementation.
(4)	The value is being fetched by specifying key as an argument of `resolveStringValue` method of `StringValueResolver` set in constructor. This value is defined in properties file.
(5)	Check whether values of properties file are encrypted. The method to determine whether the values are encrypted differs depending on type of implementation. Here, the value can be considered encrypted if it starts with “Encrypted:”. If the values are encrypted, decrypt them in step (6) and if they are not encrypted, return them as is.
(6)	Encrypted values of properties file are being decrypted. (No specific decryption process is mentioned.) Decryption method differs depending on type of implementation.

- Helper to fetch properties

```
@Value("${encrypted.property.string}") // (1)
private String testString;

@Value("${encrypted.property.int}") // (2)
private int testInt;

@Value("${encrypted.property.integer}") // (3)
private Integer testInteger;

@Value("${encrypted.property.file}") // (4)
private File testFile;

// Getters and setters omitted
```

Sr. No.	Description
(1)	By setting `${encrypted.property.string}` to `@Value` annotation value, the value for read properties file key `encrypted.property.string` is decrypted and then substituted. Value decrypted in String class is substituted in `testString`.
(2)	By setting `${encrypted.property.int}` to `@Value` annotation value, the value for read properties file key `encrypted.property.int` is decrypted and then substituted. Value decrypted in integer type is substituted in `testInt`.
(3)	By setting `${encrypted.property.integer}` to `@Value` annotation value, the value for read properties file key `encrypted.property.integer` is decrypted and then substituted. Value decrypted in Integer class is substituted in `testInteger`.
(4)	By setting `${encrypted.property.file}` to `@Value` annotation value, the value for read properties file key `encrypted.property.file` is decrypted and then substituted. In `testFile`, File object is stored as initial value which is created using the decrypted value (auto conversion).

Properties file

The values encrypted as properties values are prefixed with "Encrypted:" to indicate that they are encrypted. Although one can view the contents of properties file, but cannot understand them as the values are encrypted.

```
encrypted.property.string=Encrypted:
↪ZlpbQRJRWlNAU1FGV0ASRVteXhJQVxJXXFFAS0JGV1Yc
encrypted.property.int=Encrypted:AwI=
encrypted.property.integer=Encrypted:AwICAgI=
encrypted.property.file=Encrypted:YkBdQldARkt/U1xTVVdfV1xGHFpGX14=
```

7.3 Date operations (JSR-310 Date and Time API)

7.3.1 Overview

This guideline recommends the use of JSR-310 Date and Time API which offers various date and time calculations,
as against `java.util.Date` and `java.util.Calendar`.

> **Note:** Since JSR-310 Date and Time API has been introduced from Java8, using Joda Time is recommended for the environments of versions Java8 and below. For how to use Joda Time, refer *Date Operations (Joda Time)*.

7.3.2 How to use

Date and Time API offers various classes depending on the application such as a class exclusively handling dates and a class exclusively handling time.
In this guideline, the focus is on `java.time.LocalDate` , `java.time.LocalTime` and `java.time.LocalDateTime`, however, since the prefix of all the methods offered by each class is same for primary date and time operations, it should be interpreted by substituting with the appropriate class name.
Classes and methods used primarily are as given below.

Primary class for handling date and time

Class name	Description	Primary factory methods
java.time.LocalDate java.time.LocalTime java.time.LocalDateTime	Class which handles date and time operation and does not possess information about timezone difference	`now` Generated by current date and time `of` Generated by any date and time `parse` Generated from date and time string `from` Generated from other objects with date and time information
java.time.OffsetTime java.time.OffsetDateTime java.time.ZonedDateTime	Class which handles date and time operation considering timezone difference	Same as above
java.time.chrono.JapaneseDate	Class to handle Japanese calendar operations	Same as above

Primary class for handling information about time period

Class name	Description	Primary factory methods
java.time.Period java.time.Duration	Class which handles date and time base, time base period	`between` Generated from the difference between two objects with date and time information `from` Generated from another object that contains amount of time `of` Generated in any time period

Class for handling format

Class name	Description	Primary factory method
java.time.format.DateTimeFormatter	Class which handles operations related to date and time format	`ofPattern` Generates a formatter in specified pattern

How to use each class and method is explained below.

Note: For the topics that are not covered in the guideline, refer Javadoc for details.

Note: Date and Time API class is immutable (date and time calculation result is considered as a new object, and changes do not occur in the objects for calculation).

Fetch date and time

Fetch by current date and time

`java.time.LocalTime` , `java.time.LocalDate` and `java.time.LocalDateTime` must be selectively used in accordance with the purpose for which it is to be used.Example is shown below.

1. Use `java.time.LocalTime` when only time is to be fetched.

```
LocalTime localTime =  LocalTime.now();
```

2. Use `java.time.LocalDate` when only date is to be fetched.

```
LocalDate localDate =  LocalDate.now();
```

3. Use `java.time.LocalDateTime` when both date and time are to be fetched.

```
LocalDateTime localDateTime = LocalDateTime.now();
```

Fetch by specifying year, month, day, hours, minutes and seconds

Specific date and time can be specified by using of method. Example is shown below.

1. Specify time and fetch `java.time.LocalTime`.

```
// 23:30:59
LocalTime localTime =  LocalTime.of(23, 30, 59);
```

2. Specify date and fetch `java.time.LocalDate`.

```
// 2015/12/25
LocalDate localDate =  LocalDate.of(2015, 12, 25);
```

3. Specify date and time and fetch `java.time.LocalDateTime`.

```
// 2015/12/25 23:30:59
LocalDateTime localDateTime = LocalDateTime.of(2015, 12, 25, 23, 30, 59);
```

Also, various dates and times can be fetched by using `java.time.temporal.TemporalAdjusters`.

```
// LeapYear(2012/2)
LocalDate localDate1 = LocalDate.of(2012, 2, 1);

// Last day of month(2012/2/29)
LocalDate localDate2 = localDate1.with(TemporalAdjusters.lastDayOfMonth());

// Next monday (2012/2/6)
LocalDate localDate3 = localDate1.with(TemporalAdjusters.next(DayOfWeek.MONDAY));
```

Note: Unlike the specifications of `java.util.Calendar`, calendar month starts from 1.

Fetch date and time when time zone is specified

When an international application is to be created, a design must be adopted considering the time zone.
`java.time.OffsetTime` , `java.time.OffsetDateTime` and `java.time.ZonedDateTime` must be used selectively in Date and Time API in accordance with the purpose for which it is to be used.
Example is given below.

1. Use `java.time.OffsetTime` when the time difference between time + UTC is to be fetched.

```
// Ex, 12:30:11.567+09:00
OffsetTime offsetTime =  OffsetTime.now();
```

2. Use `java.time.OffsetDateTime` when the time difference between date, time + UTC is to be fetched.

```
// Ex, 2015-12-25T12:30:11.567+09:00
OffsetDateTime offsetDateTime =  OffsetDateTime.now();
```

3. Use `java.time.ZonedDateTime` when the time difference and region for date, time + UTC is to be fetched.

```
// Ex, 2015-12-25T12:30:11.567+09:00[Asia/Tokyo]
ZonedDateTime zonedDateTime = ZonedDateTime.now();
```

Further, current date and time considering the time zone can be fetched in all these methods by specifying `java.time.ZoneId` which indicates time zone in the argument.

`java.time.ZoneId` example is shown below.

```
ZoneId zoneIdTokyo = ZoneId.of("Asia/Tokyo");
OffsetTime offsetTime =  OffsetTime.now(zoneIdTokyo);
```

Note that, `java.time.ZoneId` consists of a method to be defined by region name/area name format and a method defined by time difference from UTC.

```
ZoneId.of("Asia/Tokyo");
ZoneId.of("UTC+01:00");
```

Although purpose of using `java.time.OffsetDateTime` and `java.time.ZonedDateTime` is similar, the basic difference is as given below.

An appropriate class should be selected according to the characteristics of the system to be created.

Class name	Description
`java.time.OffsetDateTime`	Since it only consists of quantitative value (only time difference), system does not undergo any change even if there is a change in the concept of time for each area.
`java.time.ZonedDateTime`	Since it includes concept of region besides time difference, the system undergoes change when a change occurs in the concept of time for each region.(when daylight saving etc is included as a policy)

Time period

Fetch time period

`java.time.Period` is used while handling date based period and `java.time.Duration` is used while handling time based period.

Since day represented by `java.time.Duration` is of exactly 24 hours, expected results may not be obtained if daylight saving changes are not incorporated.

On the contrary, since `java.time.Period` represents 1 day including daylight saving, an error does not occur even in the system which handles daylight saving.

Example is given below.

```
LocalDate date1 = LocalDate.of(2010, 01, 15);
LocalDate date2 = LocalDate.of(2011, 03, 18);
LocalTime time1 = LocalTime.of(11, 50, 50);
LocalTime time2 = LocalTime.of(12, 52, 53);

// One year, two months and three days.
Period pd = Period.between(date1, date2);

// One hour, two minutes and three seconds.
Duration dn = Duration.between(time1, time2);
```

Note: A method can also be employed wherein the period is generated by specifying it by using `of` method. For details, refer Javadoc of Period, Duration.

Type conversion

Interoperability of each class of Date and Time API

`java.time.LocalTime` , `java.time.LocalDate` and `java.time.LocalDateTime` can easily be mutually converted. Example is given below.

1. Conversion from `java.time.LocalTime` to `java.time.LocalDateTime`.

```
// Ex. 12:10:30
LocalTime localTime =  LocalTime.now();

// 2015-12-25 12:10:30
LocalDateTime localDateTime = localTime.atDate(LocalDate.of(2015, 12, 25));
```

2. Conversion from `java.time.LocalDate` to `java.time.LocalDateTime`.

```
// Ex. 2012-12-25
LocalDate localDate =  LocalDate.now();

// 2015-12-25 12:10:30
LocalDateTime localDateTime = localDate.atTime(LocalTime.of(12, 10, 30));
```

3. Conversion from `java.time.LocalDateTime` to `java.time.LocalTime` and `java.time.LocalDate`.

```
// Ex. 2015-12-25 12:10:30
LocalDateTime localDateTime =  LocalDateTime.now();

// 12:10:30
LocalTime localTime =  localDateTime.toLocalTime();

// 2012-12-25
LocalDate localDate =  localDateTime.toLocalDate();
```

Similarly, `java.time.OffsetTime` , `java.time.OffsetDateTime` and `java.time.ZonedDateTime` can also be easily mutually converted. Example is given below.

1. Conversion from `java.time.OffsetTime` to `java.time.OffsetDateTime`.

```
// Ex, 12:30:11.567+09:00
OffsetTime offsetTime =  OffsetTime.now();
```

```
// 2015-12-25T12:30:11.567+09:00
OffsetDateTime OffsetDateTime = offsetTime.atDate(LocalDate.of(2015, 12, 25));
```

2. Conversion from `java.time.OffsetDateTime` to `java.time.ZonedDateTime`.

```
// Ex, 2015-12-25T12:30:11.567+09:00
OffsetDateTime offsetDateTime =  OffsetDateTime.now();

// 2015-12-25T12:30:11.567+09:00[Asia/Tokyo]
ZonedDateTime zonedDateTime = offsetDateTime.atZoneSameInstant(ZoneId.of("Asia/
↪Tokyo"));
```

3. Conversion from `java.time.ZonedDateTime` to `java.time.OffsetDateTime` and `java.time.OffsetTime`.

```
// Ex, 2015-12-25T12:30:11.567+09:00[Asia/Tokyo]
ZonedDateTime zonedDateTime =  ZonedDateTime.now();

// 2015-12-25T12:30:11.567+09:00
OffsetDateTime offsetDateTime =  zonedDateTime.toOffsetDateTime();

// 12:30:11.567+09:00
OffsetTime offsetTime =  zonedDateTime.toOffsetDateTime().toOffsetTime();
```

Also, `java.time.LocalTime` can be converted to `java.time.OffsetTime` by adding time difference information.

```
// Ex, 12:30:11.567
LocalTime localTime =  LocalTime.now();

// 12:30:11.567+09:00
OffsetTime offsetTime = localTime.atOffset(ZoneOffset.ofHours(9));
```

Besides, conversion to another class is also possible by adding missing information (date information is not adequate in case of conversion from `LocalTime` to `LocalDateTime`).

Conversion method begins with the prefix `at` or `to`. For details, refer Javadoc of each class .

Interoperability with java.util.Date

A method which directly converts `java.time.LocalDate` class to `java.util.Date` is not provided.

However, since a method which converts `java.time.Instant` offered by Date and Time API is added to `java.util.Date` from Java8 and subsequent versions, a conversion can be carried out through `java.time.Instant`.
Example is given below.

1. Conversion from `java.time.LocalDateTime` to `java.util.Date`.

```
LocalDateTime localDateTime = LocalDateTime.now();
Instant instant = localDateTime.toInstant(ZoneOffset.ofHours(9));
Date date = Date.from(instant);
```

2. Conversion from `java.util.Date` to `java.time.LocalDateTime`.

```
Date date = new Date();
Instant instant = date.toInstant();
LocalDateTime localDateTime = LocalDateTime.ofInstant(instant, ZoneId.
↪systemDefault());
```

> **Note:** Since `java.time.LocalTime` and `java.time.LocalDate` do not contain Instant values, it is necessary to convert once to `java.time.LocalDateTime`.

Interoperability with java.sqlpackage

An upgrade is added to `java.sql` package from Java8 version and a method for mutual conversion with `java.time` package is defined.
Example is given below.

1. Conversion from `java.sql.Date` to `java.time.LocalDate`.

```
java.sql.Date date =  new java.sql.Date(System.currentTimeMillis());
LocalDate localDate = date.toLocalDate();
```

2. Conversion from `java.time.LocalDate` to `java.sql.Date`.

```
LocalDate localDate = LocalDate.now();
java.sql.Date date =  java.sql.Date.valueOf(localDate);
```

3. Conversion from `java.sql.Time` to `java.time.LocalTime`.

```
java.sql.Time time =  new java.sql.Time(System.currentTimeMillis());
LocalTime localTime = time.toLocalTime();
```

4. Conversion from `java.time.LocalTime` to `java.sql.Time`.

```
LocalTime localTime = LocalTime.now();
java.sql.Time time =  java.sql.Time.valueOf(localTime);
```

5. Conversion from `java.sql.Timestamp` to `java.time.LocalDateTime`.

```
java.sql.Timestamp timestamp =  new java.sql.Timestamp(System.currentTimeMillis());
LocalDateTime localDateTime = timestamp.toLocalDateTime();
```

6. Conversion from `java.time.LocalDateTime` to `java.sql.Timestamp`.

```
LocalDateTime localDateTime = LocalDateTime.now();
java.sql.Timestamp timestamp =  java.sql.Timestamp.valueOf(localDateTime);
```

How to use org.terasoluna.gfw.common.date package

Currently, Date Factory for Date and Time API is not provided in the common library. (Refer: *System Date*)

However, `java.time.LocalDate` can be generated by using `org.terasoluna.gfw.common.date.ClassicDateFactory` and `java.sql.Date` as an interim measure.

It can be generated by converting from `java.time.LocalDate` even for the `java.time.LocalTime` and `java.time.LocalDateTime` classes.

Example is given below.

bean definition file ([projectname]-env.xml)

```
<bean id="dateFactory" class="org.terasoluna.gfw.common.date.
↪DefaultClassicDateFactory" />
```

Java class

```
@Inject
ClassicDateFactory dateFactory;

public DateFactorySample getSystemDate() {

 java.sql.Date date = dateFactory.newSqlDate();
 LocalDate localDate = date.toLocalDate();

 // omitted
}
```

Note: Date Factory corresponding to Date and Time API will be added later.

Format for the string

A method which uses `toString` method and a method which uses `java.time.fomat.DateTimeFormatter` can be used for converting the object containing date and time information to string.

When outputting any date and time string, conversion to various date and time strings can be done by using `java.time.fomat.DateTimeFormatter`.

`java.time.fomat.DateTimeFormatter` consists of a method which uses a formatter of predefined ISO pattern and a method which is used by defining a format of any pattern.

```
DateTimeFormatter formatter1 = DateTimeFormatter.BASIC_ISO_DATE;

DateTimeFormatter formatter2 = DateTimeFormatter.ofPattern("G uuuu/MM/dd E")
                                                .withLocale(Locale.JAPANESE)
                                                .withResolverStyle(ResolverStyle.STRICT);
```

In this case, `Locale` and `ResolverStyle` (strict) can be defined besides string format.

Since default value of `Locale` changes depending on the system, it should be set at the time of initialization.

Also, when `ResolverStyle` (strict) uses `ofPattern` method, `ResolverStyle.SMART` is set as default, however, in this guideline, it is recommended to specify `ResolverStyle.STRICT` for strict interpretation of

date to avoid occurrence of unexpected behaviour.（When ISO pattern formatter is to be used, `ResolverStyle.STRICT` is specified.)

Also, since format `yyyy` in Date and Time API represent year in the calendar, interpretation will be different according to the calendar. (Year will be 2015 according to Western calendar but will be 0027 according to Japanese calendar).

When western calendar is to be indicated, it is recommended to use `uuuu` format instead of `yyyy` format.For the defined format list, refer DateTimeFormatter .

Example is given below.

```
DateTimeFormatter formatter1 = DateTimeFormatter.BASIC_ISO_DATE;

DateTimeFormatter formatter2 = DateTimeFormatter.ofPattern("G uuuu/MM/dd E")
                                        .withLocale(Locale.JAPANESE)
                                        .withResolverStyle(ResolverStyle.STRICT);

LocalDate localDate1 = LocalDate.of(2015, 12, 25);

// "2015-12-25"
System.out.println(localDate1.toString());
// "20151225"
System.out.println(formatter1.format(localDate1));
// "Western calendar 2015/12/25 Friday"
System.out.println(formatter2.format(localDate1));
```

Also, when the strings are to be displayed on the screen,

dedicated JSP tags do not exist in Date and Time API unlike Joda Time.

Since `fmt:formatDate` tag of JSTL handles only `java.util.Date` and `java.util.TimeZone` objects,

formatted string is passed and displayed when the date and time information held by object of Date and Time API on JSP is to be displayed.

Controller class

```
@Controller
public class HomeController {

    @RequestMapping(value = "/", method = {RequestMethod.GET, RequestMethod.POST})
```

```
    public String home(Model model, Locale locale) {

        DateTimeFormatter dateFormatter = DateTimeFormatter.ofPattern("uuuu/MM/dd")
                                            .withLocale(locale)
                                            .withResolverStyle(ResolverStyle.
↪STRICT);

        LocalDate localDate1 = LocalDate.now();

        model.addAttribute("currentDate", localDate1.toString());
        model.addAttribute("formattedCurrentDateString", dateFormatter.
↪format(localDate1));

    // omitted

    }
}
```

jsp file

```
<p>currentDate =  ${f:h(currentDate)}.</p>
<p>formattedCurrentDateString =  ${f:h(formattedCurrentDateString)}.</p>
```

Output results example (html)

```
<p>currentDate =  2015-12-25.</p>
<p>formattedCurrentDateString =  2015/12/25.</p>
```

Path from the string

Similar to conversion to string, various date strings can be converted to Date and Time API class by using `java.time.fomat.DateTimeFormatter`.

Example is as shown below.

```
DateTimeFormatter formatter1 = DateTimeFormatter.ofPattern("uuuu/MM/dd")
                                            .withLocale(Locale.JAPANESE)
                                            .withResolverStyle(ResolverStyle.
↪STRICT);

DateTimeFormatter formatter2 = DateTimeFormatter.ofPattern("HH:mm:ss")
                                            .withLocale(Locale.JAPANESE)
                                            .withResolverStyle(ResolverStyle.
↪STRICT);

LocalDate localDate = LocalDate.parse("2015/12/25", formatter1);
LocalTime localTime = LocalDate.parse("14:09:20", formatter2);
```

Date operation

Date and time can be easily calculated and compared in Date and Time API.

Example is as shown below.

Date and time calculation

`plus` method and `minus` method are provided for calculating date and time.

1. Example for calculating time.

```
LocalTime localTime =  LocalTime.of(20, 30, 50);
LocalTime plusHoursTime = localTime.plusHours(2);
LocalTime plusMinutesTime = localTime.plusMinutes(10);
LocalTime minusSecondsTime = localTime.minusSeconds(15);
```

2. Example for calculating date.

```
LocalDate localDate =  LocalDate.of(2015, 12, 25);
LocalDate plusYearsDate = localDate.plusYears(10);
LocalDate minusMonthsTime = localDate.minusMonths(1);
LocalDate plusDaysTime = localDate.plusDays(3);
```

> **Note:** If a negative number is passed in the `plus` method, results similar to the results at the time of using `minus` method can be obtained. Same for `minus` method.

Date and time comparison

Time series for past, future and current period can be compared in Date and Time API.

Example is as shown below.

1. Example for comparison of time.

```
LocalTime morning =  LocalTime.of(7, 30, 00);
LocalTime daytime =  LocalTime.of(12, 00, 00);
LocalTime evening =  LocalTime.of(17, 30, 00);

daytime.isBefore(morning); // false
morning.isAfter(evening); // true
evening.equals(LocalTime.of(17, 30, 00)); // true

daytime.isBefore(daytime); // false
morning.isAfter(morning); // false
```

2. Example for comparison of date.

```
LocalDate may =  LocalDate.of(2015, 6, 1);
LocalDate june =  LocalDate.of(2015, 7, 1);
LocalDate july =  LocalDate.of(2015, 8, 1);

may.isBefore(june); // true
june.isAfter(july); // false
july.equals(may); // false

may.isBefore(may); // false
june.isAfter(june); // false
```

Note that, the class applicable to `Interval` of Joda Time currently does not exist in Date and Time API.

Determination of date and time

Example for determining date and time is shown below.

1. When a valid date and time string is to be determined, it can be determined based on the occurrence and non-occurrence of `java.time.format.DateTimeParseException`.

```
String strDateTime = "aabbcc";
DateTimeFormatter timeFormatter  = DateTimeFormatter.ofPattern("HHmmss")
                                .withLocale(Locale.JAPANESE)
                                .withResolverStyle(ResolverStyle.STRICT);;

DateTimeFormatter dateFormatter  = DateTimeFormatter.ofPattern("uuMMdd")
                                .withLocale(Locale.JAPANESE)
                                .withResolverStyle(ResolverStyle.STRICT);;
```

```
try {
    // DateTimeParseException
    LocalTime localTime = LocalTime.parse(strDateTime, timeFormatter);
}
catch (DateTimeParseException e) {
    System.out.println("Invalid time string !!");
}

try {
    // DateTimeParseException
    LocalDate localDate = LocalDate.parse(strDateTime, dateFormatter);
}
catch (DateTimeParseException e) {
    System.out.println("Invalid date string !!");
}
```

2. When a leap year is to be determined, `isLeapYear` method of `java.time.LocalDate` can be used.

```
LocalDate date1 = LocalDate.of(2012, 1, 1);
LocalDate date2 = LocalDate.of(2015, 1, 1);

date1.isLeapYear(); // true
date2.isLeapYear(); // false
```

Fetching year, month, day, hours, minutes, seconds

When the respective year, month, day, hours, minutes and seconds are to be fetched, use `get` method.

An example to fetch information related to date is shown below.

```
LocalDate localDate = LocalDate.of(2015, 2, 1);

// 2015
int year = localDate.getYear();

// 2
int month = localDate.getMonthValue();

// 1
int dayOfMonth = localDate.getDayOfMonth();

// 32 ( day of year )
int dayOfYear = localDate.getDayOfYear();
```

Japanese calendar （JapaneseDate）

A class called `java.time.chrono.JapaneseDate` is provided in Date and Time API to handle Japanese calendar.

> **Note:** `java.time.chrono.JapaneseDate` cannot be used before Meiji 6 (1873 of Western calendar) when the Gregorian calendar was introduced.

Fetching Japanese calendar

Similar to `java.time.LocalDate`, Japanese calendar can be fetched by using `now` method and `of` method.
Further, Japanese calendar can also be fetched by using `java.time.chrono.JapaneseEra` class.

Example is shown below.

```
JapaneseDate japaneseDate1 = JapaneseDate.now();
JapaneseDate japaneseDate2 = JapaneseDate.of(2015, 12, 25);
JapaneseDate japaneseDate3 = JapaneseDate.of(JapaneseEra.HEISEI, 27, 12, 25);
```

An exception occurs when a value prior to Meiji 6 is specified in the argument.

```
// DateTimeException
JapaneseDate japaneseDate = JapaneseDate.of(1500, 1, 1);
```

Format for the string

It is possible to convert to Japanese calendar date by using `java.time.fomat.DateTimeFormatter`.
While using, calendar is set to `java.time.chrono.JapaneseChronology` by using `DateTimeFormatter#withChronology` method.
Since various formats can be handled in Japanese calendar dates as well, output can be obtained corresponding to the requirements like zero-filling or space-filling etc.
An example wherein Japanese calendar is displayed using space filling is shown below.

```
DateTimeFormatter formatter = DateTimeFormatter.ofPattern("GppyYearppMMonthppdDay")
                                  .withLocale(Locale.JAPANESE)
                                  .withResolverStyle(ResolverStyle.STRICT)
                                  .withChronology(JapaneseChronology.INSTANCE);

JapaneseDate japaneseDate = JapaneseDate.of(1992, 1, 1);

// "Heisei YY4 MM1 DD1"
System.out.println(formatter.format(japaneseDate));
```

Path from the string

Japanese string can be converted to `java.time.chrono.JapaneseDate` by using `java.time.fomat.DateTimeFormatter`.

Example is shown below.

```
DateTimeFormatter formatter = DateTimeFormatter.ofPattern("GyYearMMMonthddDate")
                                  .withLocale(Locale.JAPANESE)
                                  .withResolverStyle(ResolverStyle.STRICT)
                                  .withChronology(JapaneseChronology.INSTANCE);

JapaneseDate japaneseDate1 = JapaneseDate.from(formatter.parse("Heisei 27YY12MM25DD
↪"));
JapaneseDate japaneseDate2 = JapaneseDate.from(formatter.parse("Meiji YY6MM01DD01
↪"));
```

Conversion of Western and Japanese calendar

Conversion between Western and Japanese calendars can be easily carried out from `java.time.LocalDate` by using from method.

```
LocalDate localDate = LocalDate.of(2015, 12, 25);
JapaneseDate jpDate = JapaneseDate.from(localDate);
```

7.4 Date Operations (Joda Time)

7.4.1 Overview

The API of `java.util.Date` , `java.util.Calendar` class is poorly built to perform complex date calculations.

This guideline recommends the usage of Joda Time which provides quality replacement for the Java Date and Time classes.

In Joda Time, date is expressed using `org.joda.time.DateTime` , `org.joda.time.LocalDate` , or `org.joda.time.LocalTime` object instead of `java.util.Date`.

`org.joda.time.DateTime` , `org.joda.time.LocalDate` , or `org.joda.time.LocalTime` object is immutable (the result of date related calculations, etc. is returned in a new object).

7.4.2 How to use

The usage of Joda Time, Joda Time JSP tags is explained below.

Fetching date

Fetching current time

As per the requirements, any of the following classes can be used.

`org.joda.time.DateTime` , `org.joda.time.LocalDate` and `org.joda.time.LocalTime`.
The usage method is shown below.

1. Use `org.joda.time.DateTime` to fetch time up to milliseconds.

```
DateTime dateTime = new DateTime();
```

2. Use `org.joda.time.LocalDate` when only date, which does not include time and TimeZone, is required.

```
LocalDate localDate = new LocalDate();
```

3. Use `org.joda.time.LocalTime` when only time, which does not include date and TimeZone, is required.

```
LocalTime localTime = new LocalTime();
```

4. Use `org.joda.time.DateTime.withTimeAtStartOfDay()` to fetch the current date with the time set to start of day.

```
DateTime dateTimeAtStartOfDay = new DateTime().withTimeAtStartOfDay();
```

Note: LocalDate and LocalTime do not have the TimeZone information.

Note: It is recommended that you use `org.terasoluna.gfw.common.date.jodatime.JodaTimeDateFactor` for fetching instance of DateTime, LocalDate and LocalTime at the time of fetching current time.

```
DateTime dateTime = dataFactory.newDateTime();
```

Refer to *System Date* for using `JodaTimeDateFactory`.

LocalDate and LocalTime can be generated in the following way.

```
LocalDate localDate = dataFactory.newDateTime().toLocalDate();
LocalTime localTime = dataFactory.newDateTime().toLocalTime();
```

Fetching current time by specifying the time zone

`org.joda.time.DateTimeZone` class indicates time zone.

This class is used to fetch the current time for the specified time zone. The usage method is shown below.

```
DateTime dateTime = new DateTime(DateTimeZone.forID("Asia/Tokyo"));
```

`org.terasoluna.gfw.common.date.jodatime.JodaTimeDateFactory`is used as follows:

```
// Fetching current system date using default TimeZone
DateTime dateTime = dataFactory.newDateTime();

// Changing to TimeZone of Tokyo
DateTime dateTimeTokyo = dateTime.withZone(DateTimeZone.forID("Asia/Tokyo"));
```

For the list of other available Time zone ID strings, refer to Available Time Zones.

Fetching the date and time by specifying Year Month Day Hour Minute and Second

Specific time can be specified in the constructor. An example is given below.

- Fetching DateTime by specifying time up to milliseconds

```
DateTime dateTime = new DateTime(year, month, day, hour, minute, second,␣
↪millisecond);
```

- Fetching LocalDate by specifying Year Month and Day

```
LocalDate localDate = new LocalDate(year, month, day);
```

- Fetching LocalDate by specifying Hour Minute and Second

```
LocalTime localTime = new LocalTime(hour, minutes, seconds, milliseconds);
```

Fetching Year Month Day individually

DateTime provides a method to fetch Year and Month. The example is shown below.

```
DateTime dateTime = new DateTime(2013, 1, 10, 2, 30, 22, 123);

int year = dateTime.getYear();  // (1)
int month = dateTime.getMonthOfYear();  // (2)
int day = dateTime.getDayOfMonth();  // (3)
int week = dateTime.getDayOfWeek();  // (4)
```

```
int hour = dateTime.getHourOfDay();  // (5)
int min = dateTime.getMinuteOfHour();  // (6)
int sec = dateTime.getSecondOfMinute();  // (7)
int millis = dateTime.getMillisOfSecond();  // (8)
```

Sr. No.	Description
(1)	Get Year. In this example, `2013` is returned.
(2)	Get Month. In this example, `1` is returned.
(3)	Get Day. In this example, `10` is returned.
(4)	Get Day of Week. In this example, `4` is returned. The mapping of returned values and days of week is as follows: [1:Monday, 2:Tuesday:, 3:Wednesday, 4:Thursday, 5:Friday, 6:Saturday, 7:Sunday]
(5)	Get Hour. In this example, `2` is returned.
(6)	Get Minute. In this example, `30` is returned.
(7)	Get Second. In this example, `22` is returned.
(8)	Get Millisecond. In this example, `123` is returned.

Note: getDayOfMonth() starts with 1, differing from the specifications of `java.util.Calendar`.

Type conversion

Interoperability with java.util.Date

In DateTime, type conversion with `java.util.Date` can be easily performed.

```
Date date = new Date();

DateTime dateTime = new DateTime(date);  // (1)

Date convertDate = dateTime.toDate();  // (2)
```

Sr. No.	Description
(1)	Convert `java.util.Date` to DateTime by passing `java.util.Date` as an argument to DateTime constructor.
(2)	Convert DateTime to `java.util.Date` using DateTime#toDate method.

String format

```
DateTime dateTime = new DateTime();

dateTime.toString("yyyy-MM-dd HH:mm:ss");  // (1)
```

Sr. No.	Description
(1)	String of “yyyy-MM-dd HH:mm:ss” format is fetched. For values that can be specified as arguments of toString, refer to Input and Output .

Parsing from string

```
LocalDate localDate = DateTimeFormat.forPattern("yyyy-MM-dd").parseLocalDate("2012-
↪08-09");  // (1)
```

Sr. No.	Description
(1)	Convert “yyyy-MM-dd” string format to LocalDate type. For values that can be specified as arguments of DateTimeFormat#forPattern, refer to Formatters.

Date operations

Date calculations

LocalDate provides methods to perform date calculations. Examples are shown below.

```
LocalDate localDate = new LocalDate(); // localDate is 2013-01-10
LocalDate yesterday = localDate.minusDays(1);  // (1)
LocalDate tomorrow = localDate.plusDays(1);  // (2)
LocalDate afterThreeMonth = localDate.plusMonths(3);  // (3)
LocalDate nextYear = localDate.plusYears(1);  // (4)
```

Sr. No.	Description
(1)	The value specified in argument of LocalDate#minusDays is subtracted from the date. In this example, it becomes `2013-01-09`.
(2)	The value specified in argument of LocalDate#plusDays is added to the date. In this example, it becomes `2013-01-11`.
(3)	The value specified in argument of LocalDate#plusMonths is added to the number of months. In this example, it becomes `2013-04-10`.
(4)	The value specified in argument of LocalDate#plusYears is added to the number of years. In this example, it becomes `2014-01-10`.

For methods other than those mentioned above, refer to LocalDate JavaDoc .

Fetching first and last day of the month

The method of fetching the first and the last day of the month by considering the current date and time as base, is shown below.

```
LocalDate localDate = new LocalDate(); // dateTime is 2013-01-10
Property dayOfMonth = localDate.dayOfMonth(); // (1)
LocalDate firstDayOfMonth = dayOfMonth.withMinimumValue(); // (2)
LocalDate lastDayOfMonth = dayOfMonth.withMaximumValue(); // (3)
```

Sr. No.	Description
(1)	Get Property object that holds the attribute values related to day of current month.
(2)	Get first day of the month by fetching the minimum value from Property object. In this example, it becomes `2013-01-01`.
(3)	Get last day of the month by fetching the maximum value from Property object. In this example, it becomes `2013-01-31`.

Fetching the first and the last day of the week

The method of fetching the first and the last day of the week by considering the current date and time as base, is shown below.

```
LocalDate localDate = new LocalDate(); // dateTime is 2013-01-10
Property dayOfWeek = localDate.dayOfWeek(); // (1)
LocalDate firstDayOfWeek = dayOfWeek.withMinimumValue(); // (2)
LocalDate lastDayOfWeek = dayOfWeek.withMaximumValue(); // (3)
```

Sr. No.	Description
(1)	Get Property object that holds attribute values related to the day of current week.
(2)	Get first day of the week (Monday) by fetching the minimum value from Property object. In this example, it becomes `2013-01-07`.
(3)	Get last day of the week (Sunday) by fetching the maximum value from Property object. In this example, it becomes `2013-01-13`.

Comparison of date time

By comparing the date and time, it can be determined whether it is a past or future date.

```
DateTime dt1 = new DateTime();
DateTime dt2 = dt1.plusHours(1);
DateTime dt3 = dt1.minusHours(1);


System.out.println(dt1.isAfter(dt1)); // false
System.out.println(dt1.isAfter(dt2)); // false
System.out.println(dt1.isAfter(dt3)); // true

System.out.println(dt1.isBefore(dt1)); // false
System.out.println(dt1.isBefore(dt2)); // true
System.out.println(dt1.isBefore(dt3)); // false

System.out.println(dt1.isEqual(dt1)); // true
System.out.println(dt1.isEqual(dt2)); // false
System.out.println(dt1.isEqual(dt3)); // false
```

Sr. No.	Description
(1)	`isAfter` method returns `true` when the target date and time is later than the date and time of argument.
(2)	`isBefore` method returns `true` when the target date and time is prior to the date and time of argument.
(3)	`isEqual` method returns `true` when the target date and time is same as the date and time of argument.

Fetching the duration

Joda-Time provides several classes related to duration. The following 2 classes are explained here.

- `org.joda.time.Interval`
- `org.joda.time.Period`

Interval

Class indicating the interval between two instances (DateTime）.

The following 4 are checked using the Interval class.

- Checking whether the interval includes the specified date or interval.
- Checking whether the 2 intervals are consecutive.
- Fetching the difference between 2 intervals in an interval
- Fetching the overlapping interval between 2 intervals

For implementation, refer to the following example.

```
DateTime start1 = new DateTime(2013,8,14,0,0,0);
DateTime end1 = new DateTime(2013,8,16,0,0,0);

DateTime start2 = new DateTime(2013,8,16,0,0,0);
DateTime end2 = new DateTime(2013,8,18,0,0,0);

DateTime anyDate = new DateTime(2013, 8, 15, 0, 0, 0);

Interval interval1 = new Interval(start1, end1);
Interval interval2 = new Interval(start2, end2);

interval1.contains(anyDate);  // (1)

interval1.abuts(interval2);  // (2)

DateTime start3 = new DateTime(2013,8,18,0,0,0);
DateTime end3 = new DateTime(2013,8,20,0,0,0);
Interval interval3 = new Interval(start3, end3);

interval1.gap(interval3);  // (3)

DateTime start4 = new DateTime(2013,8,15,0,0,0);
DateTime end4 = new DateTime(2013,8,17,0,0,0);
Interval interval4 = new Interval(start4, end4);

interval1.overlap(interval4);  // (4)
```

Sr. No.	Description
(1)	Check whether the interval includes the specified date and interval, using Interval#contains method. If the interval includes the specified date and interval, return “true”. If not, return “false”.
(2)	Check whether the 2 intervals are consecutive, using Interval#abuts method. If the 2 intervals are consecutive, return “true”. If not, return “false”.
(3)	Fetch the difference between 2 intervals in an interval, using Interval#gap method. In this example, the interval between “2013-08-16~2013-08-18” is fetched. When there is no difference between intervals, return null.
(4)	Fetch the overlapping interval between 2 intervals, using Interval#overlap method. In this example, the interval between “2013-08-15~2013-08-16” is fetched. When there is no overlapping interval, return null.

Intervals can be compared by converting into Period.

- Convert the intervals into Period when comparing them from abstract perspectives such as Month or Day.

```
// Convert to Period
interval1.toPeriod();
```

Period

Period is a class indicates duration in terms of Year, Month and Week.

For example, when Period of “1 month” is added to Instant（DateTime）indicating “March 1”, DateTime will be “April 1”.

The result of adding the Period of “1 month” to “March 1” and “April 1” is as shown below.

- Number of days is “31” when a Period of “1 month” is added to “March 1”.
- Number of days is “30” when a Period of “1 month” is added to “April 1”.

The result of adding a Period of “1 month” differs depending on the target DateTime.

Two different types of implementations have been provided for Period.

- Single field Period (Example : Type having values of single unit such as “1 Day” or “1 month”)
- Any field Period (Example : Type indicating the period and having values of multiple units such as “1 month 2 days 4 hours”)

For details, refer to Period.

JSP Tag Library

fmt:formatDate tag of JSTL handles objects of java.util.Date and java.util.TimeZone.

Use Joda tag library to handle DateTime, LocalDateTime, LocalDate, LocalTime and DateTimeZone objects of Joda-time.

The functionalities of JSP Tag Library are almost same as those of JSTL, hence it can be used easily if the person has knowledge of JSTL.

How to set

The following taglib definition is required to use the tag library.

```
<%@ taglib uri="http://www.joda.org/joda/time/tags" prefix="joda"%>
```

joda:format tag

joda:format tag is used to format the objects of DateTime, LocalDateTime, LocalDate and LocalTime.

```
<% pageContext.setAttribute("now", new org.joda.time.DateTime()); %>

<span>Using pattern="yyyyMMdd" to format the current system date</span><br/>
<joda:format value="${now}" pattern="yyyyMMdd" />
<br/>
<span>Using style="SM" to format the current system date</span><br/>
<joda:format value="${now}" style="SM" />
```

Output result

Using pattern="yyyyMMdd" to format the current system date
20131025
Using style="SM" to format the current system date
10/25/13 1:02:32 PM

The list of attributes of joda:format tag is as follows:

Table.7.6 **Attribute information**

No.	Attributes	Description
1.	value	Set the instance of ReadableInstant or ReadablePartial.
2.	var	Variable name
3.	scope	Scope of variables
4.	locale	Locale information
5.	style	Style information for doing formatting (2 digits. Set the style for date and time. Values that can be entered are S=Short, M=Medium, L=Long, F=Full, -=None)
6.	pattern	Pattern for doing formatting (yyyyMMdd etc.). For patterns that can be entered, refer to Input and Output.
7.	dateTimeZone	Time zone

For other Joda-Time tags, refer to Joda Time JSP tags User guide.

Note: When the date and time is displayed by specifying style attributes, the displayed contents differ depending on browser locale. Locale of the format displayed in the above style attribute is “en”.

Example (display of calendar)

Using Spring MVC, sample for displaying a month wise calendar, is shown below.

Process name	URL	Handler method
Display of current month's calendar	/calendar	today
Display of specified month's calendar	/calendar/month?year=yyyy&month=m	month

The controller is implemented as follows:

```
@Controller
@RequestMapping("calendar")
public class CalendarController {

    @RequestMapping
    public String today(Model model) {
        LocalDate today = new LocalDate();
        int year = today.getYear();
        int month = today.getMonthOfYear();
        return month(year, month, model);
    }

    @RequestMapping(value = "month")
    public String month(@RequestParam("year") int year,
            @RequestParam("month") int month, Model model) {
        LocalDate firstDayOfMonth = new LocalDate(year, month, 1);
        LocalDate lastDayOfMonth = firstDayOfMonth.dayOfMonth()
                .withMaximumValue();

        LocalDate firstDayOfCalendar = firstDayOfMonth.dayOfWeek()
                .withMinimumValue();
        LocalDate lastDayOfCalendar = lastDayOfMonth.dayOfWeek()
                .withMaximumValue();

        List<List<LocalDate>> calendar = new ArrayList<List<LocalDate>>();
        List<LocalDate> weekList = null;
        for (int i = 0; i < 100; i++) {
            LocalDate d = firstDayOfCalendar.plusDays(i);
            if (d.isAfter(lastDayOfCalendar)) {
                break;
            }

            if (weekList == null) {
                weekList = new ArrayList<LocalDate>();
                calendar.add(weekList);
            }

            if (d.isBefore(firstDayOfMonth) || d.isAfter(lastDayOfMonth)) {
                // skip if the day is not in this month
                weekList.add(null);
            } else {
```

```
                weekList.add(d);
            }

            int week = d.getDayOfWeek();
            if (week == DateTimeConstants.SUNDAY) {
                weekList = null;
            }
        }

        LocalDate nextMonth = firstDayOfMonth.plusMonths(1);
        LocalDate prevMonth = firstDayOfMonth.minusMonths(1);
        CalendarOutput output = new CalendarOutput();
        output.setCalendar(calendar);
        output.setFirstDayOfMonth(firstDayOfMonth);
        output.setYearOfNextMonth(nextMonth.getYear());
        output.setMonthOfNextMonth(nextMonth.getMonthOfYear());
        output.setYearOfPrevMonth(prevMonth.getYear());
        output.setMonthOfPrevMonth(prevMonth.getMonthOfYear());

        model.addAttribute("output", output);

        return "calendar";
    }
}
```

The `CalendarOutput` class mentioned below is JavaBean having the consolidated information to be output on the screen.

```
public class CalendarOutput {
    private List<List<LocalDate>> calendar;

    private LocalDate firstDayOfMonth;

    private int yearOfNextMonth;

    private int monthOfNextMonth;

    private int yearOfPrevMonth;

    private int monthOfPrevMonth;

    // omitted getter/setter
}
```

> **Warning:** For the sake of simplicity, this sample code includes all the logic in the handler method of Controller, but in real scenario, this logic should be delegated to Helper classes to improve maintainability.

In JSP(calendar.jsp), it is output as follows:

```
<p>
    <a
        href="${pageContext.request.contextPath}/calendar/month?year=${f:
↪h(output.yearOfPrevMonth)}&month=${f:h(output.monthOfPrevMonth)}">&larr;
        Prev</a> <a
        href="${pageContext.request.contextPath}/calendar/month?year=${f:
↪h(output.yearOfNextMonth)}&month=${f:h(output.monthOfNextMonth)}">Next
        &rarr;</a> <br>
    <joda:format value="${output.firstDayOfMonth}"
        pattern="yyyy-M" />
</p>
<table>
    <tr>
        <th>Mon.</th>
        <th>Tue.</th>
        <th>Wed.</th>
        <th>Thu.</th>
        <th>Fri.</th>
        <th>Sat.</th>
        <th>Sun.</th>
    </tr>
    <c:forEach var="week" items="${output.calendar}">
        <tr>
            <c:forEach var="day" items="${week}">
                <td><c:choose>
                        <c:when test="${day != null}">
                            <joda:format value="${day}"
                                pattern="d" />
                        </c:when>
                        <c:otherwise> </c:otherwise>
                    </c:choose></td>
            </c:forEach>
        </tr>
    </c:forEach>
</table>
```

Access {contextPath}/calendar to display the calendar below (showing result of November 2012).



Access {contextPath}/calendar/month?year=2012&month=12 to display the calendar below.



7.4.3 Appendix

Japanese calendar operation before Java8

The class called `java.time.chrono.JapaneseDate` is offered in Java8 for Japanese calendar operation however it is possible to deal with the Japanese calendar by `java.util.Calendar` class in older Java version.

Specifically, it is necessary to specify the following `java.util.Locale` in the `java.util.Calendar` class and `java.text.DateFormat` class.

```
Locale locale = new Locale("ja", "JP", "JP");
```

Below, it shows an example of the Japanese calendar display using the `Calendar` class.

```
Locale locale = new Locale("ja", "JP", "JP");
Calendar cal = Calendar.getInstance(locale); // Ex, 2015-06-05
String format1 = "Gy.MM.dd";
String format2 = "GGGGyy/MM/dd";

DateFormat df1 = new SimpleDateFormat(format1, locale);
DateFormat df2 = new SimpleDateFormat(format2, locale);

df1.format(cal.getTime()); // "H27.06.05"
df2.format(cal.getTime()); // "平成 27/06/05"
```

Similarly, it can also perform parsing the string.

```
Locale locale = new Locale("ja", "JP", "JP");
String format1 = "Gy.MM.dd";
String format2 = "GGGGyy/MM/dd";

DateFormat df1 = new SimpleDateFormat(format1, locale);
DateFormat df2 = new SimpleDateFormat(format2, locale);

Calendar cal1 = Calendar.getInstance(locale);
Calendar cal2 = Calendar.getInstance(locale);

cal1.setTime(df1.parse("H27.06.05"));
cal2.setTime(df2.parse("平成 27/06/05"));
```

Note:

The `java.util.JapaneseImperialCalendar` object corresponding to the Japanese calendar is created by specifying the `new Locale("ja","JP","JP")` into the `getInstance` method.

If you specify the other, `java.util.GregorianCalendar` object gets created therefore it should be noted.

7.5 System Date

7.5.1 Overview

In application development, testing may need to be carried out at any random date and time and not necessarily at system time of the server. Even in Production environment, there could be cases wherein recovery is performed by shifting the date to earlier date.

Therefore, it is desirable to have a setting that can fetch any date and time at development and operation sides instead of system time of the server.

About the components that are offered by common library

The common library providing a component (In the following, API is referred as “Date Factory”) for obtaining the system time.

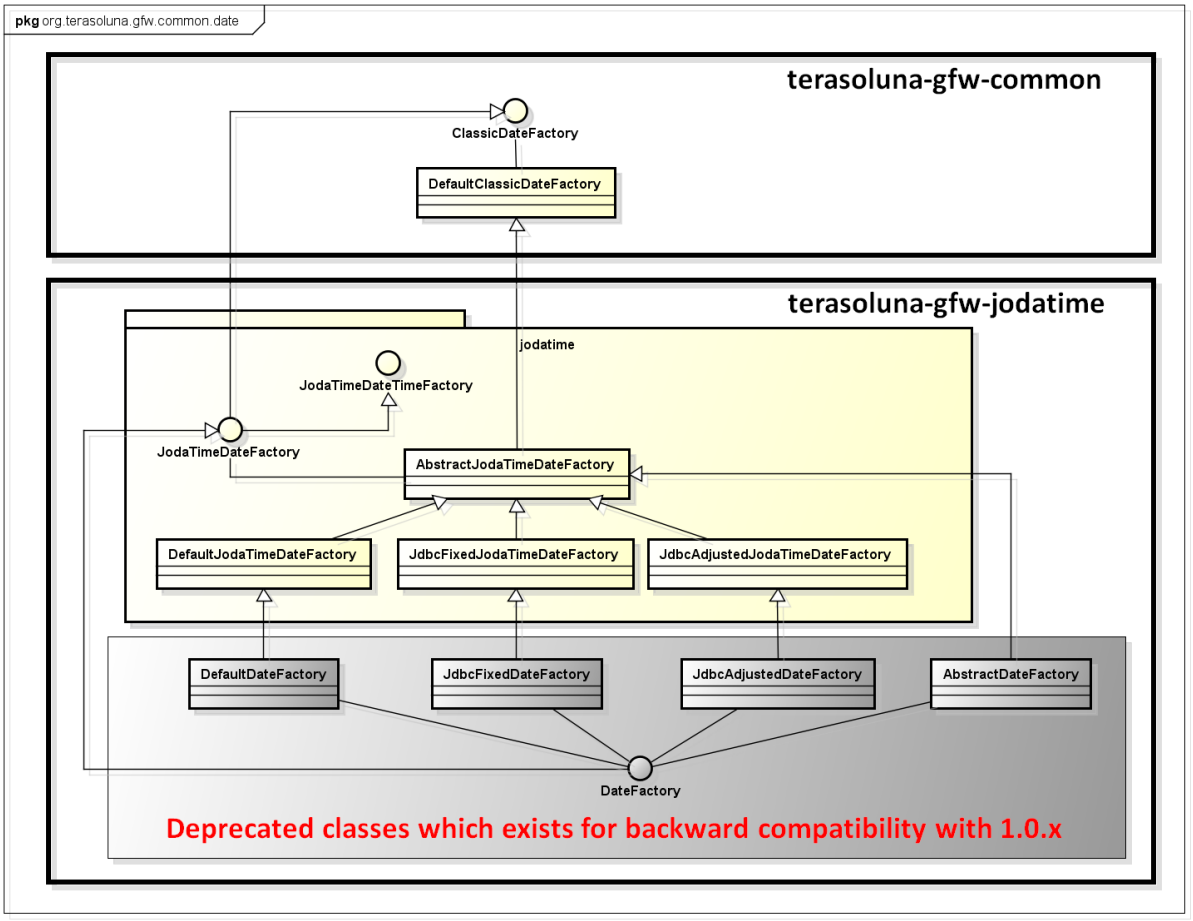
Component provided by common library is divided into two artifacts terasoluna-gfw-common and terasoluna-gfw-jodatime,

- terasoluna-gfw-common is a Date Factory that uses only the Java standard API
- terasoluna-gfw-jodatime is a Date Factory that uses Joda Time API

Following is the class diagram of components that provided by the common library.

terasoluna-gfw-common

Below interface provided as a component of terasoluna-gfw-common.



Interface	Description
org.terasoluna.gfw.common.date. ClassicDateFactory	Interface to obtain an instance of the following classes as the system time provided by Java. • `java.util.Date` • `java.sql.Timestamp` • `java.sql.Date` • `java.sql.Time` The common library provides the following classes as an implementation class of the interface. • `org.terasoluna.gfw.common.date.DefaultClassicDateFacto`

terasoluna-gfw-jodatime

Below interface provided as a component of terasoluna-gfw-jodatime.

Interface	Description
org.terasoluna.gfw.common.date.jodatime.JodaTimeDateTimeFactory	Interface to obtain an instance of the following classes as the system time provided by Joda Time. • `org.joda.time.DateTime`
org.terasoluna.gfw.common.date.jodatime.JodaTimeDateFactory	Interface that inherits from `ClassicDateFactory` and `JodaTimeDateTimeFactory`. The common library provides the following classes as an implementation class of the interface. • `org.terasoluna.gfw.common.date.jodatime.DefaultJodaTi` • `org.terasoluna.gfw.common.date.jodatime.JdbcFixedJoda` • `org.terasoluna.gfw.common.date.jodatime.JdbcAdjustedJ` **In this guideline, it is recommended that you use the implementation class corresponding to this interface.**
org.terasoluna.gfw.common.date.DateFactory	Interface that inherits from `JodaTimeDateFactory`(Deprecated). This interface is an interface provided for backward compatibility with `DateFactory` which is offered by terasoluna-gfw-common 1.0.x. The common library provides the following classes as an implementation class of the interface. • `org.terasoluna.gfw.common.date.DefaultDateFactory`(Depre • `org.terasoluna.gfw.common.date.JdbcFixedDateFactory`(De • `org.terasoluna.gfw.common.date.JdbcAdjustedDateFactory` **Since these interfaces and corresponding implementation classes are deprecated API, it is prohibited to use in new applications development.**

Note: For Joda Time, refer *Date Operations (Joda Time)*.

7.5.2 How to use

The implementation class of Date Factory interface is defined in the bean definition file and an instance of the Date Factory is used by injection in Java class.

Depending on the intended use, select from the following implementation classes.

Class name	Overview	Remarks
org.terasoluna.gfw.common.date.jodatime.DefaultJodaTimeDateFactory	Returns system time of application server.	Time cannot be changed as the value is equivalent to that fetched by new `new DateTime()`.
org.terasoluna.gfw.common.date.jodatime.JdbcFixedJodaTimeDateFactory	Returns the fixed time registered in DB.	It is assumed to be used in Integration Test environment that requires a completely fixed time. It is not used in Performance Test environment and Production environment. In order to use this class, table is required for managing a fixed time.
org.terasoluna.gfw.common.date.jodatime.JdbcAdjustedJodaTimeDateFactory	Returns the time fetched by adding difference (milliseconds) between the time registered in DB and system time of application server.	It is assumed to be used in Integration Test environment and System Test environment but It can also be used in Production environment by setting the difference value as 0. In order to use this class, table is required for managing a difference value.

Note: It is recommended to define the bean definition file that sets implementation class in [projectName]-env.xml so that it can be changed according to the environment. Using Date Factory, the date and time can be changed just by changing the settings of bean definition file, without having to change the source code. Example of Bean definition file is described later.

Tip: If you want to test in JUnit by changing the date and time, it is also possible to set a random time by replacing the implementation class of the interface to mock class. For replacement method refer [*Unit Test*].

pom.xml setting

Add terasoluna-gfw-jodatime dependency.

In case of multi-project configuration, add in the `pom.xml`(`projectName-domain/pom.xml`) of the domain project.

If project is created from blank project, dependencies of terasoluna-gfw-jodatime is pre-configured.

```
<dependencies>

    <!-- (1) -->
    <dependency>
        <groupId>org.terasoluna.gfw</groupId>
        <artifactId>terasoluna-gfw-jodatime-dependencies</artifactId>
        <type>pom</type>
    </dependency>

</dependencies>
```

Sr. No.	Description
1.	Add a terasoluna-gfw-jodatime-dependencies into dependencies. The dependencies of Joda Time related to Date Factory and Joda Time libraries are defined for terasoluna-gfw-jodatime-dependencies.

Note: In the above setting example, since it is assumed that the dependent library version is managed by the parent project terasoluna-gfw-parent, specifying the version in pom.xml is not necessary.

Returning system time of server

Use `org.terasoluna.gfw.common.date.jodatime.DefaultJodaTimeDateFactory`.

Bean definition file([projectname]-env.xml)

```
<bean id="dateFactory" class="org.terasoluna.gfw.common.date.jodatime.
↪DefaultJodaTimeDateFactory" />  <!-- (1) -->
```

Sr. No.	Description
(1)	Define `DefaultJodaTimeDateFactory` class in bean.

Java class

```
@Inject
JodaTimeDateFactory dateFactory;  // (2)

public TourInfoSearchCriteria setUpTourInfoSearchCriteria() {

    DateTime dateTime = dateFactory.newDateTime();  // (3)

    // omitted
}
```

Sr. No.	Description
(2)	Inject Date Factory in the class to be used.
(3)	Call the method that returns the class instance of the date to be used In above example `org.joda.time.DateTime` type instance is fetched.

Returning the fixed time fetched from DB

Use `org.terasoluna.gfw.common.date.jodatime.JdbcFixedJodaTimeDateFactory`.

Bean definition file

```
<bean id="dateFactory" class="org.terasoluna.gfw.common.date.jodatime.
↪JdbcFixedJodaTimeDateFactory" >  <!-- (1) -->
    <property name="dataSource" ref="dataSource" />  <!-- (2) -->
    <property name="currentTimestampQuery" value="SELECT now FROM system_date" />
↪<!-- (3) -->
</bean>
```

Sr. No.	Description
(1)	Define `JdbcFixedJodaTimeDateFactory` in bean.
(2)	Specifies the datasource (`javax.sql.DataSource`) in the `dataSource` property in which table to manage a fixed time is present.
(3)	Set the SQL in `currentTimestampQuery` property for obtaining a fixed time.

Example of Table settings

Records need to be added by creating a table as shown below.

```
CREATE TABLE system_date(now timestamp NOT NULL);
INSERT INTO system_date(now) VALUES (current_date);
```

Record number	now
1	2013-01-01 01:01:01.000

Java class

```
@Inject
JodaTimeDateFactory dateFactory;

@RequestMapping(value="datetime", method = RequestMethod.GET)
public String listConfirm(Model model) {

    for (int i=0; i < 3; i++) {
        model.addAttribute("jdbcFixedDateFactory" + i, dateFactory.newDateTime());
↪// (4)
        model.addAttribute("DateTime" + i, new DateTime()); // (5)
    }

    return "date/dateTimeDisplay";
}
```

Sr. No.	Description
(4)	Pass the system time retrieved from Date Factory to the screen. When confirming the results, the output is the fixed value set in DB.
(5)	Pass the result of `new DateTime()` for confirmation. When confirming the results, each time the output is different value (System time of the application server).

Execution result



SQL log

```
16. SELECT now FROM system_date {executed in 0 msec}
17. SELECT now FROM system_date {executed in 1 msec}
18. SELECT now FROM system_date {executed in 0 msec}
```

Access log is output to DB if Date Factory is called. In order to output SQL log, `Log4jdbcProxyDataSource` described in *Database Access (Common)* is used.

Returning time obtained by adding the difference registered in DB to the server system time

Use `org.terasoluna.gfw.common.date.jodatime.JdbcAdjustedJodaTimeDateFactory`.

bean definition file

```
<bean id="dateFactory" class="org.terasoluna.gfw.common.date.jodatime.
↪JdbcAdjustedJodaTimeDateFactory" > <!-- (1) -->
  <property name="dataSource" ref="dataSource" /> <!-- (2) -->
  <property name="adjustedValueQuery" value="SELECT diff * 60 * 1000 FROM␣
↪operation_date" /> <!-- (3) -->
</bean>
```

Sr. No.	Description
(1)	Define `JdbcAdjustedJodaTimeDateFactory` in bean.
(2)	Specifies the datasource (`javax.sql.DataSource`) in the `dataSource` property in which table to manage a difference value is present.
(3)	Set the SQL in `adjustedValueQuery` property for obtaining a difference value. The above SQL is the SQL of the difference values in “minutes” unit.

Example of table settings

Records need to be added by creating a table as shown below.

```
CREATE TABLE operation_date(diff bigint NOT NULL);
INSERT INTO operation_date(diff) VALUES (-1440);
```

Record number	diff
1	-1440

In this example, the difference is in “minutes”. (DB data is specified as -1440 minutes = previous day)

By converting the retrieved result into milliseconds (integer value), the unit for DB value can be set to any one of the units namely, hours, minutes, seconds or milliseconds.

Note: Above SQL is for PostgreSQL. For Oracle, it is better to use *NUMBER(19)* instead of *BIGINT*.

Tip: If you want to make difference value unit other than the "minutes", the following SQL can be specified in the *adjustedValueQuery* property.

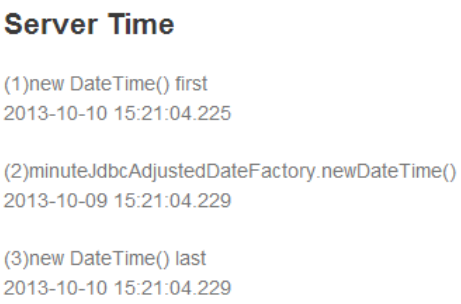
Difference value unit	SQL
milliseconds	SELECT diff FROM operation_date
seconds	SELECT diff * 1000 FROM operation_date
hours	SELECT diff * 60 * 60 * 1000 FROM operation_date
days	SELECT diff * 24 * 60 * 60 * 1000 FROM operation_date

Java class

```
@Inject
JodaTimeDateFactory dateFactory;

@RequestMapping(value="datetime", method = RequestMethod.GET)
public String listConfirm(Model model) {

    model.addAttribute("firstExpectedDate", new DateTime());  // (4)
    model.addAttribute("serverTime", dateFactory.newDateTime());  // (5)
    model.addAttribute("lastExpectedDate", new DateTime());  // (6)

    return "date/dateTimeDisplay";
}
```

Sr. No.	Description
(4)	Pass the time that retrieved before calling the Date Factory method to the screen for confirmation.
(5)	Pass the system time retrieved from Date Factory to the screen. When confirming the results, the output is the time that is 1440 minutes subtracted from the execution time.
(6)	Pass the time that retrieved after calling the Date Factory method to the screen for confirmation.

Execution result



SQL log

```
17. SELECT diff * 60 * 1000 FROM operation_date {executed in 1 msec}
```

Access log is output to DB if Date Factory is called.

Caching and reloading the difference

When the difference value is set to 0 and used in production environment, performance deteriorates as the difference is fetched each time from DB. Therefore, in *JdbcAdjustedJodaTimeDateFactory*, it is possible to cache the difference values obtained by the SQL. Once the value fetched at booting is cached, table is not accessed for each request.

bean definition file

```
<bean id="dateFactory" class="org.terasoluna.gfw.common.date.jodatime.
↪JdbcAdjustedJodaTimeDateFactory" >
  <property name="dataSource" ref="dataSource" />
  <property name="adjustedValueQuery" value="SELECT diff * 60 * 1000 FROM␣
↪operation_date" />
  <property name="useCache" value="true" /> <!-- (1) -->
</bean>
```

Sr. No.	Description
(1)	When it is *true*, the difference value fetched from table is cached. By default it is *false* so the value is not cached. When it is *false*, SQL is executed each time when the method of Date Factory is called.

When the difference value is to be changed after setting cache, cache value can be reloaded by executing *JdbcAdjustedJodaTimeDateFactory.reload()* method after changing the table value.

Java class

```
@Controller
@RequestMapping(value = "reload")
public class ReloadAdjustedValueController {

    @Inject
    JdbcAdjustedJodaTimeDateFactory dateFactory;

    // omitted

    @RequestMapping(method = RequestMethod.GET)
    public String reload() {

        long adjustedValue = dateFactory.reload(); // (2)
```

```
        // omitted
    }
}
```

Sr. No.	Description
(2)	By executing *reload* method of the *JdbcAdjustedJodaTimeDateFactory*, difference can be reloaded from table.

7.5.3 Testing

When carrying out testing, it may be necessary to change to another date and time instead of the current date and time.

Environment	Date Factory to be used	Test details
Unit Test	DefaultJodaTimeDateFactory	Mock for DataFactory is created for date related testing
Integration Test	DefaultJodaTimeDateFactory	Testing not relating to date
	JdbcFixedJodaTimeDateFactory	When testing is carried out by having a fixed date and time
	JdbcAdjustedJodaTimeDateFactory	When linked with an external system and testing is done for multiple days considering the date flow of a testing for a single day
System Test	JdbcAdjustedJodaTimeDateFactory	When testing is carried out by specifying the testing date or for a future date
Production	DefaultJodaTimeDateFactory	When there is no possibility of change in actual time
	JdbcAdjustedJodaTimeDateFactory	**When the possibility to change the time is to be retained in an operation.** **Normally the difference is set as 0. It is provided only if required.** **Always,** *useCache* **should be set to 'true'.**

Unit Test

In Unit Test, sometimes it needs to be verified whether the time is registered and the registered time has been updated as expected.

In such cases, if the server time is registered as it is during the process, it becomes difficult to perform regression test in JUnit, as the value differs with each test execution. Here, by using Date Factory, the time to be registered can be fixed to any value.

Use mock to match the time in milliseconds. An example wherein fixed date is returned by setting a value in Date Factory, is shown below. In this example, mockito is used for mock.

Java class

```
import org.terasoluna.gfw.common.date.jodatime.JodaTimeDateFactory;

// omitted

@Inject
StaffRepository staffRepository;

@Inject
JodaTimeDateFactory dateFactory;

@Override
public Staff staffUpdateTel(String staffId, String tel) {

    // ex staffId=0001
    Staff staff = staffRepository.findOne(staffId);

    // ex tel = "0123456789"
    staff.setTel(tel);

    // set ChangeMillis
    staff.setChangeMillis(dateFactory.newDateTime()); // (1)

    staffRepository.save(staff);

    return staff;
}

// omitted
```

JUnit source

```
import static org.junit.Assert.*;
import static org.hamcrest.CoreMatchers.*;
import static org.mockito.Mockito.*;

import org.joda.time.DateTime;
import org.junit.Before;
import org.junit.Test;
import org.terasoluna.gfw.common.date.jodatime.JodaTimeDateFactory;

public class StaffServiceTest {

    StaffService service;

    StaffRepository repository;
```

```
    JodaTimeDateFactory dateFactory;

    DateTime now;

    @Before
    public void setUp() {
        service = new StaffService();
        dateFactory = mock(JodaTimeDateFactory.class);
        repository = mock(StaffRepository.class);
        now = new DateTime();
        service.dateFactory = dateFactory;
        service.staffRepository = repository;
        when(dateFactory.newDateTime()).thenReturn(now); // (2)
    }

    @After
    public void tearDown() throws Exception {
    }

    @Test
    public void testStaffUpdateTel() {

        Staff setDataStaff = new Staff();
        when(repository.findOne("0001")).thenReturn(setDataStaff);

        // execute
        Staff staff = service.staffUpdateTel("0001", "0123456789");

        //assert
        assertThat(staff.getChangeMillis(), is(now)); // (3)

    }
}
```

Sr. No.	Description
(1)	Value specified in (2) of mock is fetched and set.
(2)	Set the date and time to the return value of DateFactory in mock.
(3)	**success** is returned since it is same as the fixed value that has been set.

Example wherein process changes with date

The example below illustrates a Service class which is implemented with the specification of "Reserved tour cannot be cancelled if the cancellation is sought less than 7 days before the departure day".

Java class

```
import org.terasoluna.gfw.common.date.jodatime.JodaTimeDateFactory;

// omitted

@Inject
JodaTimeDateFactory dateFactory;

// omitted

@Override
public void cancel(String reserveNo) throws BusinessException {
    // omitted

    LocalDate today = dateFactory.newDateTime().toLocalDate(); // (1)
    LocalDate cancelLimit = tourInfo.getDepDay().minusDays(7); // (2)

    if (today.isAfter(cancelLimit)) { // (3)
        // omitted (4)
    }

    // omitted
}
```

Sr. No.	Description
(1)	Fetch current date and time. For `LocalDate`, refer to *Date Operations (Joda Time)*.
(2)	Calculate the last date up to which the tour can be cancelled.
(3)	Check if today's date is later than the last date for cancellation.
(4)	`BusinessException` is thrown if the date exceeds the last date for cancellation.

JUnit source

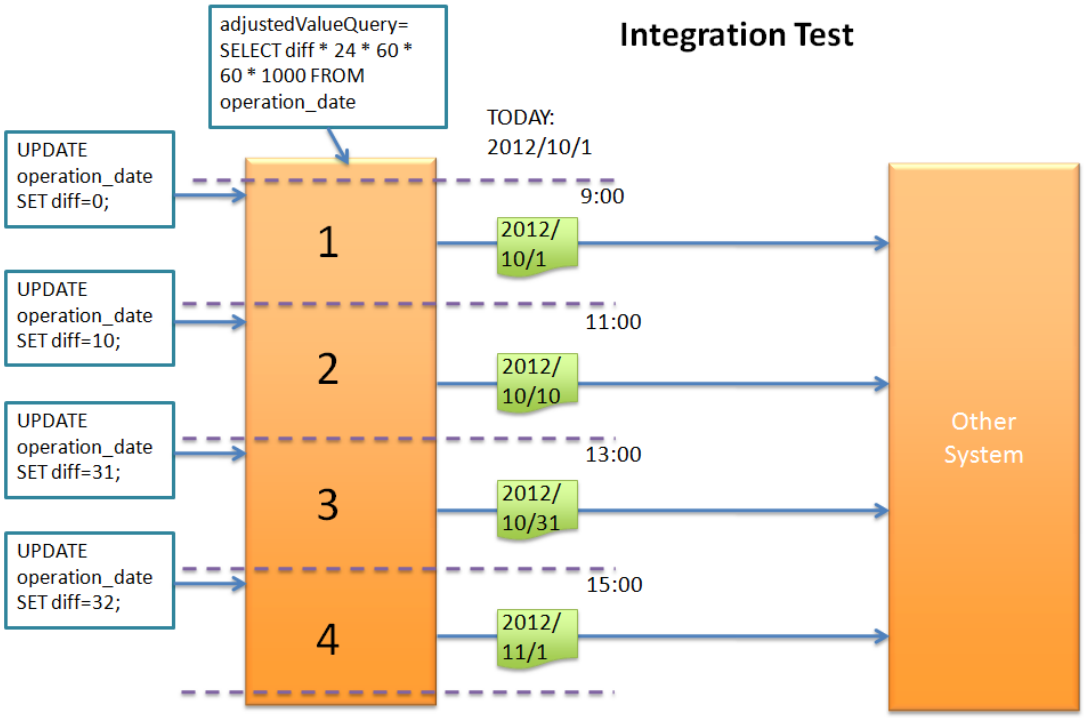
```
@Before
public void setUp() {
    service = new ReserveServiceImpl();

    // omitted

    Reserve reserveResult = new Reserve();
    reserveResult.setDepDay(new LocalDate(2012, 10, 10)); // (5)
    when(reserveRepository.findOne((String) anyObject())).thenReturn(
            reserveResult);
    dateFactory = mock(JodaTimeDateFactory.class);
    service.dateFactory = dateFactory;
}

@Test
public void testCancel01() {

  // omitted

  now = new DateTime(2012, 10, 1, 0, 0, 0, 0);
  when(dateFactory.newDateTime()).thenReturn(now); // (6)

  // run
  service.cancel(reserveNo); // (7)
```

```
  // omitted
}

@Test(expected = BusinessException.class)
public void testCancel02() {

  // omitted

  now = new DateTime(2012, 10, 9, 0, 0, 0, 0);
  when(dateFactory.newDateTime()).thenReturn(now); // (8)

  try {
      // run
      service.cancel(reserveNo); // (9)
      fail("Illegal Route");
  } catch (BusinessException e) {
      // assert message if required
      throw e;
  }
}
```

Sr. No.	Description
(5)	Set the departure date to 2012/10/10 in the tour reservation information to be fetched from Repository class.
(6)	Set the Return value of dateFactory.newDateTime() to 2012/10/1.
(7)	Execute Cancel. Cancellation is successful as the date is prior to the last date for cancellation.
(8)	Return value of dateFactory.newDateTime() should be 2012/10/9.
(9)	Execute Cancel. Cancellation fails as the date falls after the last date for cancellation.

Integration Test

In Integration Test, there may be cases wherein data of several days (for example: files) is created and transferred in a single day, for communicating with the system.



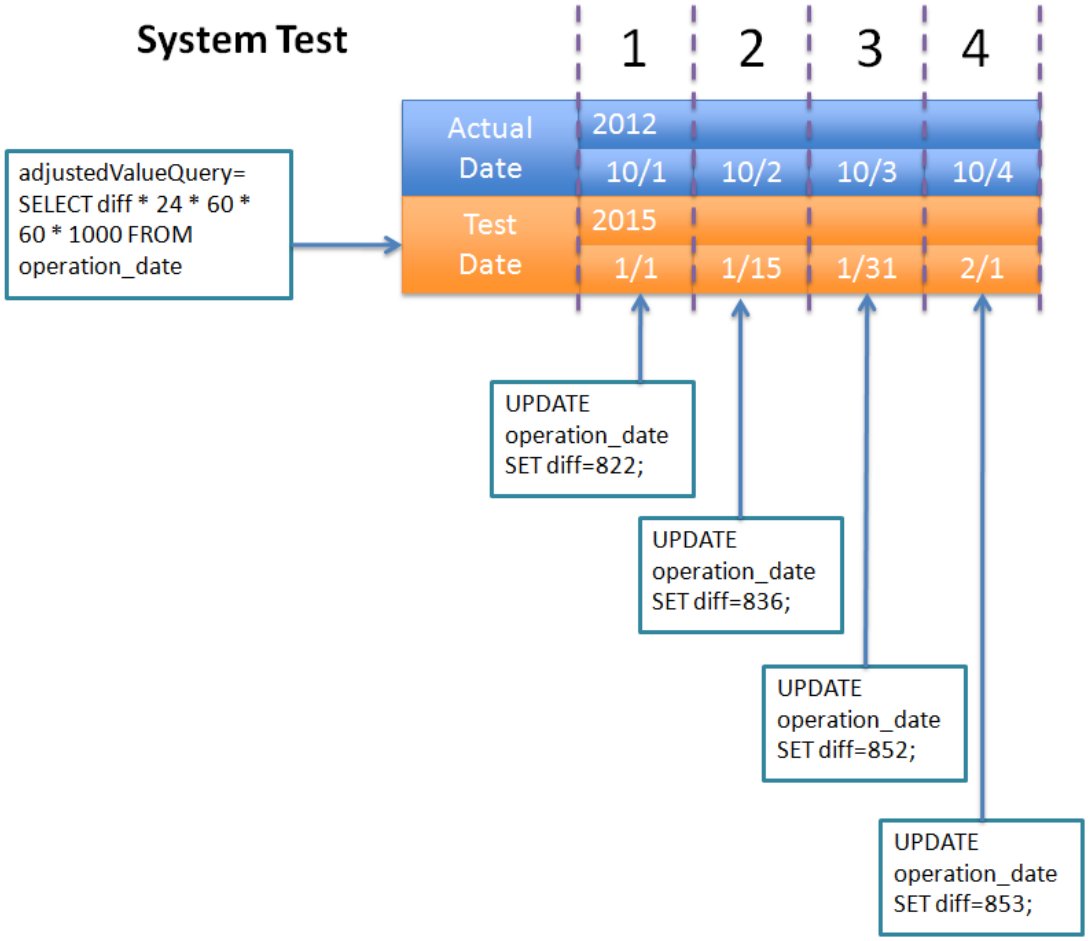
When the actual date is 2012/10/1, Use *JdbcAdjustedJodaTimeDateFactory* and set the SQL to calculate the difference with test execution date.

Sr. No.	Description
1	Set the difference between 9:00-11:00 as “0 days” and return value of Date Factory as 2012/10/1.
2	Set the difference between 11:00-13:00 as “0 days” and return value of Date Factory as 2012/10/10.
3	Set the difference between 13:00-15:00 as “30 days” and return value of Date Factory as 2012/10/31.
4	Set the difference between 15:00-17:00 as “31 days” and return value of Date Factory as 2012/11/1.

Date can be changed only by changing the table value.

System Test

In System Test, testing may be carried out by creating test scenarios assuming the operation date.



Use *JdbcAdjustedJodaTimeDateFactory* and set SQL that calculates the date difference. Create a mapping table for actual date and operation date like 1, 2, 3 and 4 as shown in the figure. Testing can be carried out on the desired date, only by changing the difference value in the table.

Production

By setting the difference value to '0' using *JdbcAdjustedJodaTimeDateFactory*, the return value of Date Factory can be set to the date same as the actual date without changing the source. Even the bean definition file need not

be changed from System Test onwards. Further, even if the need to change date and time arises, return value of Date Factory can be changed by changing the table value.

Warning: When using in Production environment, verify that the difference value in the table used in Production environment is 0.

Configuration example

- 窶「When using the table for the first time in Production environment,
  - INSERT INTO operation_date (diff) VALUES (0);
- 窶「When test execution is completed in Production environment
  - UPDATE operation_date SET diff=0;

Always, *useCache* **should be set to 'true'.**

When there is no change in time, it is recommended to change the configuration file to *DefaultJodaTimeDateFactory*.

7.6 String Processing

7.6.1 Overview

There are very few operations in string standard API of Java which specialise in Japanese language.
A process must be devised independently for conversion of full width katakana / half width katakana and determination of a string consisting of only half width katakana.

Also, note that although in Java, all the strings are represented in Unicode
the special characters like 吉 are represented in unicode by char type 2 (32 bits) which is referred to as surrogate pair.
Even while handling these characters, an implementation which takes into account various types of characters are necessary to prevent occurrence of unexpected behavior.

The guideline assumes a case wherein Japanese language is processed, and
includes a typical string operation example and offering a Japanese language operation API by a common library.

7.6.2 How to use

Trim

`String#trim` method can be used as well while carrying out half width blank trim operation, however, while carrying out complex trim operations like only leading and trailing trim operation, trim operation of any string etc. `org.springframework.util.StringUtils` provided by Spring should be preferably used.

Example is given below.

```
String str = "  Hello World!!";

StringUtils.trimWhitespace(str); // => "Hello World!!"

StringUtils.trimLeadingCharacter(str, ' '); // => "Hello World!!"
```

```
StringUtils.trimTrailingCharacter(str, '!'); // => "  Hello World"
```

Note: There is no change in the behaviour even if surrogate pair string is specified in the first argument of `StringUtils#trimLeadingCharacter` and `StringUtils#trimTrailingCharacter`. Also note that, since the second argument is of char type, surrogate pair cannot be specified.

Padding, Suppress

While carrying out padding (string padding) and suppress (string takeout) operations, a method provided by `String` class can be used.

Example is given below.

```
int num = 1;

String paddingStr = String.format("%03d", num); // => "001"
String suppressStr = paddingStr.replaceFirst("^0+(?!$)", ""); // => "1"
```

Warning: If a surrogate pair is included while carrying out padding of apparent length, appropriate results cannot be obtained since surrogate pair cannot be taken into account by `String#format`. In order to achieve the padding by using surrogate pair, it is necessary to count number of characters considered as surrogate pair described later, calculate appropriate number of characters that should be padded and join the strings.

Processing of a string considered as a surrogate pair

Fetching string length

When the length of the string considered as surrogate pair is to be fetched, it is not possible to simply use `String#length` method.
Since surrogate pair is represented by 32 bits (char type 2), the count tends to exceed than the apparent number of characters.

In the example below, 5 is assigned to variable `len`.

```
String str = "吉田太郎";
int len = str.length(); // => 5
```

Accordingly, a method `String#codePointCount` is defined wherein length of the string considered as a surrogate pair is fetched from Java SE 5.

String length can be fetched by specifying start index and end index of the target string in the argument of `String#codePointCount`.

Example is given below.

```
String str = "吉田太郎";
int lenOfChar = str.length(); // => 5
int lenOfCodePoint = str.codePointCount(0, lenOfChar); // => 4
```

Further, Unicode consists of “concatenation”.

Since there is no appearance difference between `\u304c` indicating [が] and `\u304b\u3099` indicating [か] plus [voiced sound mark], however [か] plus [voiced sound mark] are likely to be counted as two characters.

When number of characters are to be counted including the combining characters as well as described above, counting is done after normalization of text using `java.text.Normalizer`.

A method which returns length of the string after considering combining characters and surrogate pair is given below.

```
public int getStrLength(String str) {
  String normalizedStr  = Normalizer.normalize(str, Normalizer.Form.NFC);
  int length = normalizedStr.codePointCount(0, normalizedStr.length());

  return length;
}
```

Fetch string in the specified range

When a string of specified range is to be fetched, unintended results may be obtained if only `String#substring` is used.

```
String str = "吉田太郎";
int startIndex = 0;
int endIndex = 2;

String subStr = str.substring(startIndex, endIndex);

System.out.println(subStr); // => "吉田"
```

In the example above, when you try to fetch "吉田" by taking out 2 characters from 0th character (beginning), only "吉" could be fetched since the surrogate pair is represented by 32 bits (char type 2).

In such a case, `String#substring` method must be used by searching start and end positions considering the surrogate pair, by using `String#offsetByCodePoints`.

An example wherein 2 characters are taken from the beginning (surname part) is shown below.

```
String str = "吉田太郎";
int startIndex = 0;
int endIndex = 2;

int startIndexSurrogate = str.offsetByCodePoints(0, startIndex); // => 0
int endIndexSurrogate = str.offsetByCodePoints(0, endIndex); // => 3

String subStrSurrogate = str.substring(startIndexSurrogate, endIndexSurrogate); //␣
↪=> "吉田"
```

String split

`String#split` method handles the surrogate pair as a default.

Example is given below.

```
String str = "吉田太郎";

str.split(" "); // => {"吉田", "太郎"}
```

Note:

Surrogate pair can also be specified in the argument of `String#split` as a delimiter.

Note:

Please note that behaviour while passing a blank character in `String#split` changes in Java SE 7 environment, and Java SE 8. Refer Pattern#split Javadoc

```
String str = "ABC";
String[] elems = str.split("");

// Java SE 7 => {, A, B, C}
// Java SE 8 => {A, B, C}
```

Full width, half width string conversion

Full width and half width character conversion is carried out by using API of `org.terasoluna.gfw.common.fullhalf.FullHalfConverter` class provided by common library.

`FullHalfConverter` class adopts a style wherein pair definition of full width and half width characters for conversion (`org.terasoluna.gfw.common.fullhalf.FullHalfPair`) is registered in advance. `FullHalfConverter` object for which default pair definition is registered, is provided as a `INSTANCE` constant of `org.terasoluna.gfw.common.fullhalf.DefaultFullHalf` class in the common library. For default pair definition, refer DefaultFullHalf source .

Note: When change requirements are not met in the default pair definition provided by common library, `FullHalfConverter` object registering a unique pair definition should be created. For basic methods for creation, refer *Creating FullHalfConverter class for which a unique full width and half width character pair definition is registered* .

How to apply common library

It is necessary to add following common library as dependency library in case of *Full width, half width string conversion* is used.

```
<dependencies>
    <dependency>
        <groupId>org.terasoluna.gfw</groupId>
        <artifactId>terasoluna-gfw-string</artifactId>
    </dependency>
</dependencies>
```

Note: In the setting example above, the dependent library version is assumed to be managed by the parent project. Therefore, the `<version>`element is not specified.

Conversion to full width string

`toFullwidth` method of `FullHalfConverter` is used while converting a half width character to full width character.

```
String fullwidth = DefaultFullHalf.INSTANCE.toFullwidth("ｱﾞ!A8ｶﾞｻﾞ");     // (1)
```

Sr. No.	Description
(1)	Pass the string which contains half width characters to argument of `toFullwidth` method and convert to full width string. In this example, it is converted to "アゞ！Ａ８ガザ". Also, note that the characters for which a pair is not defined ("ザ" in this example) are returned without any change.

Conversion to half width string

`toHalfwidth` method of `FullHalfConverter` is used while converting a full width character to half width character.

```
String halfwidth = DefaultFullHalf.INSTANCE.toHalfwidth("Ａ！アガｻ");     // (1)
```

Sr. No.	Description
(1)	Pass the string which contains full width characters to argument of `toHalfwidth` method and convert to half width characters. In this example, it is converted to "A!ｱｶﾞｻ". Also, note that the characters for which a pair is not defined ("サ" of this example) are returned without any change.

Note: `FullHalfConverter` cannot convert combining characters that represent a single character using 2 or more characters (Example: “"シ"(\u30b7) + voiced sound mark(\u3099)”) to half width character (Example: "ｼﾞ"). When combining characters are to be converted to half width characters, `FullHalfConverter` must be used after converting the same to integrated characters （Example:"ジ"(\u30b8)）by carrying out text normalization.

`java.text.Normalizer` is used while carrying out text normalization. Note that, when combining characters are to be converted to integrated characters, NFC or NFKC is used as a normalization format.

Implementation example wherein NFD (analyse by canonical equivalence) is used as a normalization format

```
String str1 = Normalizer.normalize("モジ", Normalizer.Form.NFD); // str1 = "モ
シ + Voiced sound mark(\u3099)"
String str2 = Normalizer.normalize("ﾓｼﾞ", Normalizer.Form.NFD);  // str2 = "ﾓｼﾞ"
```

Implementation example wherein NFC (analyse by canonical equivalence, and integrate again) is used as a normalization format

```
String mojiStr = "モシ\u3099";                                    // "モシ +␣
↪Voiced sound mark(\u3099)"
String str1 = Normalizer.normalize(mojiStr, Normalizer.Form.NFC); // str1 = "モ
ジ (\u30b8) "
String str2 = Normalizer.normalize("ﾓｼﾞ", Normalizer.Form.NFC);   // str2 = "ﾓｼﾞ
↪"
```

Implementation example wherein NFKD (analyse by compatibility equivalent) is used as a normalization format

```
String str1 = Normalizer.normalize("モジ", Normalizer.Form.NFKD); // str1 = "モ
シ + Voiced sound mark(\u3099)"
String str2 = Normalizer.normalize("ﾓｼﾞ", Normalizer.Form.NFKD);  // str2 = "モ
シ + Voiced sound mark(\u3099)"
```

Implementation example wherein NFKC (analyse by compatibility equivalent and integrate again) is used as a normalization format

```
String mojiStr = "モシ\u3099";                                    // "モシ +␣
↪Voiced sound mark(\u3099)"
String str1 = Normalizer.normalize(mojiStr, Normalizer.Form.NFKC); // str1 =
↪"モジ (\u30b8) "
String str2 = Normalizer.normalize("ﾓｼﾞ", Normalizer.Form.NFKC) ;  // str2 =
↪"モジ"
```

For details, refer Normalizer JavaDoc.

Creating FullHalfConverter class for which a unique full width and half width character pair definition is registered

`FullHalfConverter` for which a unique full width and half width character pair definition is registered can also be used without using `DefaultFullHalf`.

How to use `FullHalfConverter` for which a unique full width character and half width character pair definition is registered, is shown below.

Implementation example of a class that provides FullHalfConverter for which a unique pair definition is registered

```
public class CustomFullHalf {

    private static final int FULL_HALF_CODE_DIFF = 0xFEE0;

    public static final FullHalfConverter INSTANCE;

    static {
        // (1)
        FullHalfPairsBuilder builder = new FullHalfPairsBuilder();

        // (2)
        builder.pair("ー", "-");

        // (3)
        for (char c = '!'; c <= '~'; c++) {
            String fullwidth = String.valueOf((char) (c + FULL_HALF_CODE_DIFF));
            builder.pair(fullwidth, String.valueOf(c));
        }

        // (4)
        builder.pair("。", "｡").pair("「", "｢").pair("」", "｣").pair("、", "､")
                .pair("・", "･").pair("ァ", "ｧ").pair("ィ", "ｨ").pair("ゥ", "ｩ")
                .pair("ェ", "ｪ").pair("ォ", "ｫ").pair("ャ", "ｬ").pair("ュ", "ｭ")
                .pair("ョ", "ｮ").pair("ッ", "ｯ").pair("ア", "ｱ").pair("イ", "ｲ")
                .pair("ウ", "ｳ").pair("エ", "ｴ").pair("オ", "ｵ").pair("カ", "ｶ")
                .pair("キ", "ｷ").pair("ク", "ｸ").pair("ケ", "ｹ").pair("コ", "ｺ")
                .pair("サ", "ｻ").pair("シ", "ｼ").pair("ス", "ｽ").pair("セ", "ｾ")
                .pair("ソ", "ｿ").pair("タ", "ﾀ").pair("チ", "ﾁ").pair("ツ", "ﾂ")
                .pair("テ", "ﾃ").pair("ト", "ﾄ").pair("ナ", "ﾅ").pair("ニ", "ﾆ")
                .pair("ヌ", "ﾇ").pair("ネ", "ﾈ").pair("ノ", "ﾉ").pair("ハ", "ﾊ")
                .pair("ヒ", "ﾋ").pair("フ", "ﾌ").pair("ヘ", "ﾍ").pair("ホ", "ﾎ")
                .pair("マ", "ﾏ").pair("ミ", "ﾐ").pair("ム", "ﾑ").pair("メ", "ﾒ")
                .pair("モ", "ﾓ").pair("ヤ", "ﾔ").pair("ユ", "ﾕ").pair("ヨ", "ﾖ")
                .pair("ラ", "ﾗ").pair("リ", "ﾘ").pair("ル", "ﾙ").pair("レ", "ﾚ")
                .pair("ロ", "ﾛ").pair("ワ", "ﾜ").pair("ヲ", "ｦ").pair("ン", "ﾝ")
                .pair("ガ", "ｶﾞ").pair("ギ", "ｷﾞ").pair("グ", "ｸﾞ")
```

```
                .pair("ゲ", "ｹﾞ").pair("ゴ", "ｺﾞ").pair("ザ", "ｻﾞ")
                .pair("ジ", "ｼﾞ").pair("ズ", "ｽﾞ").pair("ゼ", "ｾﾞ")
                .pair("ゾ", "ｿﾞ").pair("ダ", "ﾀﾞ").pair("ヂ", "ﾁﾞ")
                .pair("ヅ", "ﾂﾞ").pair("デ", "ﾃﾞ").pair("ド", "ﾄﾞ")
                .pair("バ", "ﾊﾞ").pair("ビ", "ﾋﾞ").pair("ブ", "ﾌﾞ")
                .pair("ベ", "ﾍﾞ").pair("ボ", "ﾎﾞ").pair("パ", "ﾊﾟ")
                .pair("ピ", "ﾋﾟ").pair("プ", "ﾌﾟ").pair("ペ", "ﾍﾟ")
                .pair("ポ", "ﾎﾟ").pair("ヴ", "ｳﾞ").pair("\u30f7", "ﾜﾞ")
                .pair("\u30fa", "ｦﾞ").pair("ﾞ ", "ﾞ").pair("ﾟ ", "ﾟ").pair("  ", " ");

        // (5)
        INSTANCE = new FullHalfConverter(builder.build());
    }
}
```

Sr. No.	Description
(1)	Use `org.terasoluna.gfw.common.fullhalf.FullHalfPairsBuilder` and create `org.terasoluna.gfw.common.fullhalf.FullHalfPairs` which represents a set of pair definition of full width and half width characters.
(2)	Half width character corresponding to `"ー"`of full width character set to `"-"(\uFF70)` in `DefaultFullHalf` is changed to `"-"(\u002D)` in this example. Further, although `"-"(\u002D)` is also included in the process target given below (3), the pair definition defined earlier is given the precedence.
(3)	In this example, a pair is defined for code values of full width of unicode from `"！"` to `"～"` and of half width of unicode from `"!"` to `"~"` using a loop process which use the characteristic “code value sequence is same”.
(4)	Since code value sequence for the characters other than given in (3) does not match for full width characters and half width characters, define a pair individually for respective characters.
(5)	Use `FullHalfPairs` created by `FullHalfPairsBuilder` and create `FullHalfConverter`.

Note: For the values that can be specified in the argument of `FullHalfPairsBuilder#pair` method, refer FullHalfPair constructor JavaDoc

How to use FullHalfConverter for which a unique pair definition is registered

```
String halfwidth = CustomFullHalf.INSTANCE.toHalfwidth("ハローワールド！"); // (1)
```

Sr. No.	Description
(1)	Use `toHalfwidth` method of `FullHalfConverter` object for which a unique pair definition is registered and convert the string containing full width characters to half width string. In this example, it is converted to "ﾊﾛｰﾜｰﾙﾄﾞ！". ("ｰ" is `\u002D`)

Code point set check (character type check)

A code point set function provided by common library should be used for checking character type.

Here, how to implement a character type check by using a code point set function is explained.

- *Creating code point set*
- *Set operation of the code point sets*
- *String check by using code point set*
- *String check linked with Bean Validation*
- *Creating new code point set class*

How to apply common library

It is necessary to add *Code point set class provided by common library* as dependency library in case of *Code point set check (character type check)* is used.

Creating code point set

`org.terasoluna.gfw.common.codepoints.CodePoints` is a class that represents a code point set.

How to create `CodePoints` instance is shown below.

When an instance is created by calling a factory method (cache)

A method wherein an instance is created from a code point set class (`Class<? extends CodePoints>`) and the created instance is then cached, is explained below.

This method is recommended for cache process since it is not necessary to create multiple specific code point sets.

```
CodePoints codePoints = CodePoints.of(ASCIIPrintableChars.class);  // (1)
```

Sr. No.	Description
(1)	Pass code point set class in `CodePoints#of` method (factory method) and fetch an instance. In this example, an instance of code point set class (`org.terasoluna.gfw.common.codepoints.catalog.ASCIIPrintableChars`) of Ascii printable characters is fetched.

Note: Code point set class exists multiple times in the module, same as `CodePoints` class. Although other modules which provide code point set also exist, these modules must be added to their own projects when required. For details, refer *Code point set class provided by common library*.

Further, a new code point set class can also be created. For details, refer *Creating new code point set class*.

When an instance is created by calling a constructor of code point set class

A method wherein an instance is created from code point set class.

When this method is used, it is recommended to use the process which need not be cached (argument of set operation) since the created instance is not cached.

```
CodePoints codePoints = new ASCIIPrintableChars();  // (1)
```

Sr. No.	Description
(1)	Call constructor by using `new` operator and generate an instance of code point set class. In this example, an instance of code point set class (`ASCIIPrintableChars`) of Ascii printable characters is generated.

When an instance is created by calling constructor of CodePoints

A method wherein an instance is created from `CodePoints` is shown below.
When this method is used, it is recommended to use the process which need not be cached (argument of set operation) since the created instance is not cached.

- When the codepoint (`int`) is to be passed by using a variable length argument

```
CodePoints codePoints = new CodePoints(0x0061 /* a */, 0x0062 /* b */);  //␣
↪(1)
```

Sr. No.	Description
(1)	Generate an instance by passing `int` code point in `CodePoints` constructor. In this example, an instance of code point set for characters `"a"` and `"b"` is generated.

- When the `Set` of code point (`int`) is to be passed

```
Set<Integet> set = new HashSet<>();
set.add(0x0061 /* a */);
```

```
set.add(0x0062 /* b */);
CodePoints codePoints = new CodePoints(set);  // (1)
```

Sr. No.	Description
(1)	Add code point of `int` to `Set` and generate an instance by passing the `Set` in constructor of `CodePoints`. In this example, an instance of code point set for characters `"a"` and `"b"` is generated.

- When code point set string is to be passed by using variable length argument

```
CodePoints codePoints = new CodePoints("ab");          // (1)
```

```
CodePoints codePoints = new CodePoints("a", "b");  // (2)
```

Sr. No.	Description
(1)	Generate an instance by passing code point set string in constructor of `CodePoints`. In this example, an instance of code point set for characters `"a"` and `"b"` is generated.
(2)	Code point set string can also be passed by dividing it in the arguments. Result is same as (1).

Set operation of the code point sets

A new code point set instance can be created by performing set operation for code point set.

Further, note that status of source code point set does not change due to set operation.

A method wherein an instance of code point set is created by using set operation is given below.

When an instance of code point set is created by using union set method

```
CodePoints abCp = new CodePoints(0x0061 /* a */, 0x0062 /* b */);
CodePoints cdCp = new CodePoints(0x0063 /* c */, 0x0064 /* d */);

CodePoints abcdCp = abCp.union(cdCp);    // (1)
```

Sr. No.	Description
(1)	Calculate union of two code point sets by using `CodePoints#union` method and create an instance of new code point set. In this example, union of "code point set included in string `"ab"`" and "code point set included in string `"cd"`" is calculated and an instance of new code point set (code point set included in string `"abcd"`) is generated.

When an instance of code point set is created by using difference set method

```
CodePoints abcdCp = new CodePoints(0x0061 /* a */, 0x0062 /* b */,
        0x0063 /* c */, 0x0064 /* d */);
CodePoints cdCp = new CodePoints(0x0063 /* c */, 0x0064 /* d */);

CodePoints abCp = abcdCp.subtract(cdCp);    // (1)
```

Sr. No.	Description
(1)	Calculate difference set of two code point sets by using `CodePoints#subtract` method and create an instance of new code point set. In this example, difference set of "code point set included in string `"abcd"`" and "code point set included in string `"cd"`" is calculated and an instance of new code point set (code point set included in string `"ab"`) is created.

When an instance of new code point set is to be created by intersection set

```
CodePoints abcdCp = new CodePoints(0x0061 /* a */, 0x0062 /* b */,
        0x0063 /* c */, 0x0064 /* d */);
CodePoints cdeCp = new CodePoints(0x0063 /* c */, 0x0064 /* d */, 0x0064 /* e */);
```

```
CodePoints cdCp = abcdCp.intersect(cdeCp);     // (1)
```

Sr. No.	Description
(1)	Calculate intersection set of two code point sets by using `CodePoints#intersect` method and create an instance of new code point set. In this example, calculate intersection set of "code point set included in string `"abcd"`" and "code point set included in string `"cde"`" is calculated and an instance of new code point set (code point set included in string `"cd"`) is created.

String check by using code point set

A string can be checked by using a method provided in `CodePoints`.

How to use a method which is used while checking the string is given below.

containsAll method

Determine whether the entire string for checking is included in the code point set.

```
CodePoints jisX208KanaCp = CodePoints.of(JIS_X_0208_Katakana.class);

boolean result;
result = jisX208KanaCp.containsAll("カ");      // true
result = jisX208KanaCp.containsAll("カナ");    // true
result = jisX208KanaCp.containsAll("カナa");   // false
```

firstExcludedContPoint method

Return the first code point which is not included in the code point set, from the string targeted for checking. Further, return `CodePoints#NOT_FOUND` when the entire string for checking is included in code point set.

```
CodePoints jisX208KanaCp = CodePoints.of(JIS_X_0208_Katakana.class);

int result;
result = jisX208KanaCp.firstExcludedCodePoint("カナ a");  // 0x0061 (a)
result = jisX208KanaCp.firstExcludedCodePoint("カ a ナ");  // 0x0061 (a)
result = jisX208KanaCp.firstExcludedCodePoint("カナ");    // CodePoints#NOT_FOUND
```

allExcludedCodePoints method

Return `Set` of the code point which is not included in the code point set, from the string targeted for checking.

```
CodePoints jisX208KanaCp = CodePoints.of(JIS_X_0208_Katakana.class);

Set<Integer> result;
result = jisX208KanaCp.allExcludedCodePoints("カナ a");  // [0x0061 (a)]
result = jisX208KanaCp.allExcludedCodePoints("カ a ナ b"); // [0x0061 (a), 0x0062 (b)]
result = jisX208KanaCp.allExcludedCodePoints("カナ");    // []
```

String check linked with Bean Validation

It can be checked whether the entire string targeted for checking is included in the specified code point set by specifying code point set class in `@org.terasoluna.gfw.common.codepoints.ConsistOf` annotation.

How to use is shown below.

When there is only one code point set used for checking

```
@ConsisOf(JIS_X_0208_Hiragana.class)    // (1)
private String firstName;
```

Sr. No.	Description
(1)	Check whether the string specified in the targeted field is entirely “Hiragana of JIS X 0208”.

When there are multiple code point sets used for checking

```
@ConsisOf({JIS_X_0208_Hiragana.class, JIS_X_0208_Katakana.class})    // (1)
private String firstName;
```

Sr. No.	Description
(1)	Check whether the string specified in the targeted field is entirely “Hiragana of JIS X 0208” or “Katakana of JIS X 0208”.

Note: If string of length N is checked by code point sets M, a checking process that contains N x M is employed. When the string is large, it is likely to cause performance degradation. Hence, a new code point set class that acts as a union set of code point set used for checking is created, that class alone should be specified.

Creating new code point set class

When a new code point set class is to be created, specify code point using constructor by inheriting `CodePoints` class.

A method wherein a new code point set class is created is given below.

When a new code point set class is created by specifying code point

How to create code point set which is formed by “only numbers”

```
public class NumberChars extends CodePoints {
    public NumberCodePoints() {
        super(0x0030 /* 0 */, 0x0031 /* 1 */, 0x0032 /* 2 */, 0x0033 /* 3 */,
                0x0034 /* 4 */, 0x0035 /* 5 */, 0x0036 /* 6 */,
                0x0037 /* 7 */, 0x0038 /* 8 */, 0x0039 /* 9 */);
    }
}
```

When a new code point set class is created by using a set operation method of code point set class

How to create a code point set using a union set consisting of "Hiragana" and "Katakana"

```
public class FullwidthHiraganaKatakana extends CodePoints {
    public FullwidthHiraganaKatakana() {
        super(new X_JIS_0208_Hiragana().union(new X_JIS_0208_Katakana()));
    }
}
```

How to create a code point set using difference set consisting of "half width katakana excluding symbols (。「」、・)"

```
public class HalfwidthKatakana extends CodePoints {
    public HalfwidthKatakana() {
        CodePoints symbolCp = new CodePoints(0xFF61 /* 。 */, 0xFF62 /* 「 */,
                0xFF63 /* 」 */, 0xFF64 /* 、 */, 0xFF65 /* ・ */);

        super(new JIS_X_0201_Katakana().subtract(symbolCp));
    }
}
```

Note: When the code point set class used in set operation (`X_JIS_0208_Hiragana` or `X_JIS_0208_Katakana`etc in this example) is not to be used individually, it must be ensured that code point is not needlessly cached, by using `new` operator and calling constructor. If it is cached by using `CodePoints#of`method, code point set used only during set operation calculation remains in the heap resulting in load on the memory. On the other hand, if it is used individually, it should be cached using `CodePoints#of` method.

Code point set class provided by common library

Code point class provided by common library (`org.terasoluna.gfw.common.codepoints.catalog` package class) and artifact information to be incorporated while using are given below.

Sr. No.	Class name	Description	Artifact information

(1)	`ASCIIControlChars`	Ascii control characters set. (`0x0000-0x001F`、`0x007F`)	`<dependency>`
			`<groupId>org.terasoluna.gfw</gr`
			`<artifactId>terasoluna-gfw-code`
			`</dependency>`

(2)	`ASCIIPrintableChars`	Ascii printable characters set. (`0x0020-0x007E`)	(Same as above)
(3)	`CRLF`	Linefeed code set. `0x000A`(LINE FEED) and `0x000D`(CARRIAGE RETURN)。	(Same as above)

(4)	`JIS_X_0201_Katakana`	JIS X 0201 katakana set. Symbols (。「」、・) included as well.	`<dependency>`
			`<groupId>org.terasoluna.gfw.cod`
			`<artifactId>terasoluna-gfw-code`
			`</dependency>`

(5)	`JIS_X_0201_LatinLetters`	JIS X 0201 Latin characters set.	(Same as above)

(6)	`JIS_X_0208_SpecialChars`	Row 2 of JIS X 0208: Special characters set.	`<dependency>`
			`<groupId>org.terasoluna.gfw.cod`
			`<artifactId>terasoluna-gfw-code`
			`</dependency>`

(7)	`JIS_X_0208_LatinLetters`	Row 3 of JIS X 0208: Alphanumeric set.	(Same as above)

次のページに続く

Table. 7.8 – 前のページからの続き

Sr. No.	Class name	Description	Artifact information
(8)	`JIS_X_0208_Hiragana` set.	Row 4 of JIS X 0208: Hiragana	(Same as above)
(9)	`JIS_X_0208_Katakana` set.	Row 5 of JIS X 0208: Katakana	(Same as above)
(10)	`JIS_X_0208_GreekLetters` set.	Row 6 of JIS X 0208: Greek letters	(Same as above)
(11)	`JIS_X_0208_CyrillicLetters` set.	Row 7 of JIS X 0208: Cyrillic letters	(Same as above)
(12)	`JIS_X_0208_BoxDrawingChar`	Row 8 of JIS X 0208: Box drawing characters.	(Same as above)

(13)	`JIS_X_0208_Kanji`	Kanji 6355 characters specified in JIS X 208.
		First and second level kanjis.
			`<dependency>`
			`<groupId>org.terasoluna.gfw.cod`
			`<artifactId>terasoluna-gfw-code`
			`</dependency>`

(14)	`JIS_X_0213_Kanji`	Kanji 10050 characters specified in JIS X 0213:2004.
		First, second, third and fourth level kanjis.
			`<dependency>`
			`<groupId>org.terasoluna.gfw.cod`
			`<artifactId>terasoluna-gfw-code`
			`</dependency>`

Note:

In the above setting example, since it is assumed that the dependent library version is managed by the parent project terasoluna-gfw-parent , specifying the version in pom.xml is not necessary.

`JIS_X_0208_SpecialChars`codepoint set class is a special character set corresponding to JIS chinese characters (JIS X 0208)-section 01-02. Double byte dash (-) of JIS chinese characters is EM DASH and the corresponding UCS(ISO/IEC 10646-1, JIS X 0221, Unicode) codepoints usually correspond to `U+2014`. However, in the conversion table offered by Unicode consortium , characters supported by Unicode are HORINZONTAL BAR (U+2015) instead of EM DASH.. Since general conversion rules that are being used and Unicode conversion table vary, problems may occur during actual use if codepoint set is defined as per Unicode conversion table. Therefore, codepoint set is defined in 、`JIS_X_0208_SpecialChars` codepoint set class by converting HORINZONTAL BAR (`U+2015`) to EM DASH (`U+2014`).

7.7 Bean Mapping (Dozer)

7.7.1 Overview

Bean mapping is a process to copy the field values of one Bean to another Bean.

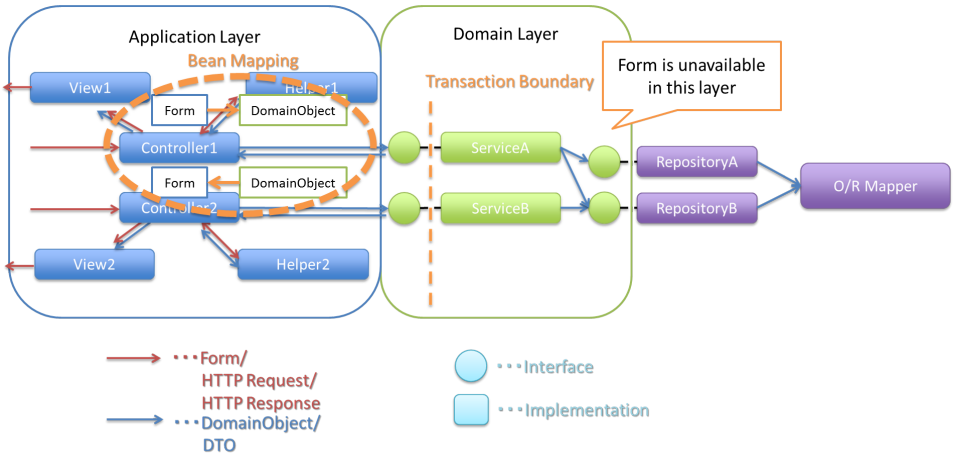
There are many cases where bean mapping is necessary while passing the data between different layers of the application (application layer and domain layer).

For example, `AccountForm` object of application layer is converted to `Account` object of domain layer.

Since the domain layer should not depend on the application layer, AccountForm object cannot be used as it is in the domain layer.

Hence, `AccountForm` object is Bean mapped to `Account` object and `Account` object is used in the domain layer.

Thus, linear dependency relation can be maintained between application layer and domain layer.

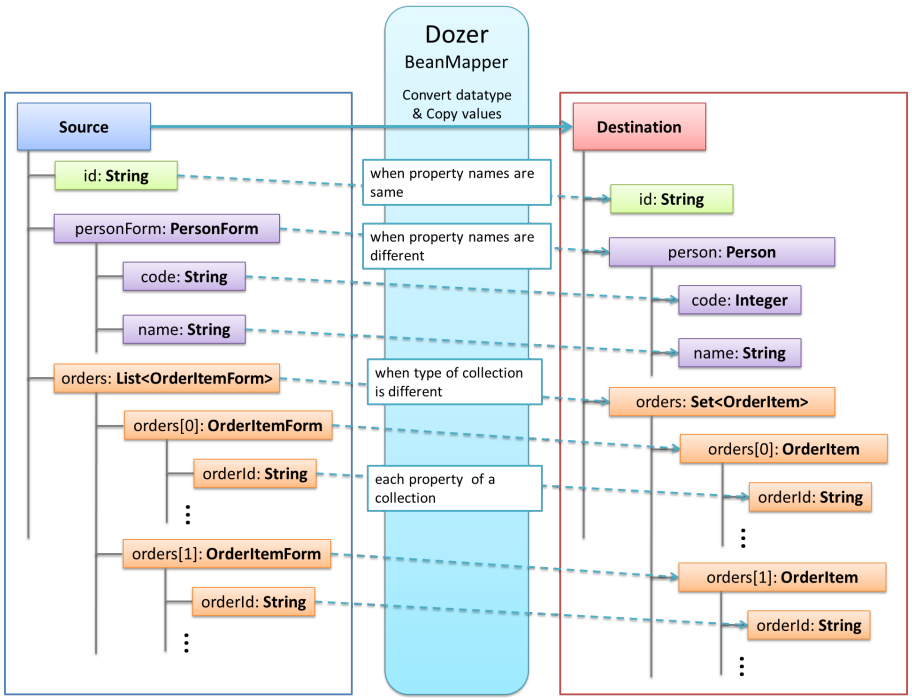


These objects can be mapped by calling getter/setter of Bean and passing the data.

However, since a complex process results in deterioration of the program readability, in this guideline, it is recommended to use Dozer a Bean mapping library available in OSS.

By using Dozer, different types of copies for copy source class and copy destination class and copy of nested

Beans can be easily performed as shown in the following figure.



Code examples with/without using Dozer are given below.

- Example of a complex process that results in the program readability deterioration

```
User user = userService.findById(userId);

XxxOutput output = new XxxOutput();

output.setUserId(user.getUserId());
output.setFirstName(user.getFirstName());
output.setLastName(user.getLastName());
output.setTitle(user.getTitle());
output.setBirthDay(user.getBirthDay());
output.setGender(user.getGender());
output.setStatus(user.getStatus());
```

- Example when Dozer is used

```
User user = userService.findById(userId);

XxxOutput output = beanMapper.map(user, XxxOutput.class);
```

How to use Dozer is explained below.

> **Warning:** Classes added by JSR-310 Date and Time API cannot be used in default state due to generation of exceptions, and hence a custom converter must be created.

7.7.2 How to use

Dozer is a mapping function library of JavaBean. Values are copied recursively (nested structure) from conversion source Bean to conversion destination Bean.

Bean definition for using Dozer

Create an instance of `org.dozer.Mapper` when Dozer is to be used independently as follows.

```
Mapper mapper = new DozerBeanMapper();
```

Creating instance of Mapper each time deteriorates efficiency, hence `org.dozer.spring.DozerBeanMapperFactoryBean` provided by Dozer should be used.

Define `org.dozer.spring.DozerBeanMapperFactoryBean`, a Factory class that creates Mapper in Bean definition file (applicationContext.xml).

```
<bean class="org.dozer.spring.DozerBeanMapperFactoryBean">
    <property name="mappingFiles"
        value="classpath*:/META-INF/dozer/**/*-mapping.xml" /><!-- (1) -->
</bean>
```

Sr. No.	Description
(1)	Specify mapping definition XML files in mappingFiles. `org.dozer.spring.DozerBeanMapperFactoryBean` maintains `org.dozer.Mapper` as an interface. Therefore, specify `Mapper` for `@Inject`. In this example, all the (any value)-mapping.xmls in any folder of /META-INF/dozer under the class path are read. The contents of these XML files are explained later.

`Mapper` should be injected in the class to perform Bean mapping.

```
@Inject
Mapper beanMapper;
```

Mapping when the field name and the type between Beans is same

Dozer can perform mapping by default without creating mapping definition XML file if the field name between the Beans is same.

Bean definition of conversion source

```
public class Source {
    private int id;
    private String name;
    // omitted setter/getter
}
```

Bean definition of conversion destination

```
public class Destination {
    private int id;
    private String name;
    // omitted setter/getter
}
```

Perform Bean mapping using `map` method of `Mapper` as given below. After executing the method given below, a new Destination object is created and each field value of source is copied to the created Destination object.

```
Source source = new Source();
source.setId(1);
source.setName("SourceName");
Destination destination = beanMapper.map(source, Destination.class); // (1)
System.out.println(destination.getId());
System.out.println(destination.getName());
```

Sr. No.	Description
(1)	Pass the object to be copied as the first argument and the Bean class where it is to be copied as the second argument.

The output of the above code is as given below. The value of object to be copied is set in the created object.

```
1
SourceName
```

To copy the field of `source` object to already existing `destination` object, perform the following

```
Source source = new Source();
source.setId(1);
source.setName("SourceName");
Destination destination = new Destination();
destination.setId(2);
destination.setName("DestinationName");
beanMapper.map(source, destination); // (1)
System.out.println(destination.getId());
System.out.println(destination.getName());
```

Sr. No.	Description
(1)	Pass the object to be copied as the first argument and the object where it is to be copied as the second argument.

The output of the above code is as given below. The value of object to be copied is reflected in the destination where it is to be copied.

```
1
SourceName
```

Note: The value that does not exist in `Source` class does not change before and after copying to the field of `Destination` class.

Bean definition of conversion source

```
public class Source {
    private int id;
    private String name;
    // omitted setter/getter
}
```

Bean definition of conversion destination

```
public class Destination {
    private int id;
    private String name;
    private String title;
    // omitted setter/getter
}
```

Example of Mapping

```
Source source = new Source();
source.setId(1);
source.setName("SourceName");
Destination destination = new Destination();
destination.setId(2);
destination.setName("DestinationName");
destination.setTitle("DestinationTitle");
beanMapper.map(source, destination);
System.out.println(destination.getId());
System.out.println(destination.getName());
System.out.println(destination.getTitle());
```

The output of the above code is as given below. Since there is no `title` field in the `Source` class, the value of `title` field of `Destination` object remains the same as the field value before copying.

```
1
SourceName
DestinationTitle
```

Mapping when the field name is the same and type between Beans is different

When field type of Bean is different in copy source and copy destination, mapping of the type where type conversion is supported can be done automatically.

The conversion given below is an example where it is possible to convert without a mapping definition XML file.

Example: String -> BigDecimal

Bean definition of conversion source

```
public class Source {
    private String amount;
    // omitted setter/getter
}
```

Bean definition of conversion destination

```
public class Destination {
    private BigDecimal amount;
    // omitted setter/getter
}
```

Example of Mapping

```
Source source = new Source();
source.setAmount("123.45");
```

```
Destination destination = beanMapper.map(source, Destination.class);
System.out.println(destination.getAmount());
```

The output of the above code is as given below. The value can be copied even when the type is different.

```
123.45
```

Refer to Manual for the supported type conversions.

Mapping when the field name between Beans is different

When field name of copy source is different from the copy destination, creating mapping definition XML file and defining the field for Bean mapping enables conversion.

Bean definition of conversion source

```
public class Source {
    private int id;
    private String name;
    // omitted setter/getter
}
```

Bean definition of conversion destination

```
public class Destination {
    private int destinationId;
    private String destinationName;
    // omitted setter/getter
}
```

To define *Bean definition for using Dozer*, create mapping definition XML file called (any value)-mapping.xml in the src/main/resources/META-INF/dozer folder.

```
<?xml version="1.0" encoding="UTF-8"?>
<mappings xmlns="http://dozer.sourceforge.net" xmlns:xsi="http://www.w3.org/2001/
↪XMLSchema-instance"
    xsi:schemaLocation="http://dozer.sourceforge.net
          http://dozer.sourceforge.net/schema/beanmapping.xsd">

    <mapping>
      <class-a>com.xx.xx.Source</class-a><!-- (1) -->
      <class-b>com.xx.xx.Destination</class-b><!-- (2) -->
      <field>
        <a>id</a><!-- (3) -->
        <b>destinationId</b><!-- (4) -->
      </field>
      <field>
        <a>name</a>
```

```
        <b>destinationName</b>
      </field>
    </mapping>

</mappings>
```

Sr. No.	Description
(1)	Specify fully qualified class name (FQCN) of copy source Bean in <class-a> tag.
(2)	Specify fully qualified class name (FQCN) of copy destination Bean in <class-b> tag.
(3)	Specify field name for mapping of copy source Bean in <a> tag of <field> tag.
(4)	Specify field name for mapping of copy destination Bean corresponding to (3) in <b> tag of <field> tag.

Example of Mapping

```
Source source = new Source();
source.setId(1);
source.setName("SourceName");
Destination destination = beanMapper.map(source, Destination.class); // (1)
System.out.println(destination.getDestinationId());
System.out.println(destination.getDestinationName());
```

Sr. No.	Description
(1)	Pass object to be copied as the first argument and the Bean class where it is to be copied as the second argument. (no difference with basic mapping.)

The output of the above code is as given below.

```
1
SourceName
```

Mapping definition XML file existing under META-INF/dozer of class path is read in `mappingFiles` property in accordance with the setting of *Bean definition for using Dozer*. File name should be (any value)-mapping.xml. The settings are applied if mapping between `Source` class and `Destination` class is defined in any file.

Note: It is recommended to create a mapping definition XML file in each Controller and name the file as -mapping.xml (value obtained by removing Controller from Controller name). For example, mapping definition XML file of TodoController is created in src/main/resources/META-INF/dozer/todo-mapping.xml.

One-way/Two-way mapping

The mapping defined in mapping XML is two-way mapping by default. In the above example, mapping is done from `Source` object to `Destination` object however, mapping can also be done from `Destination` object to `Source` object.

To specify only one-way mapping, set `"one-way"` in the `type` attribute of `<mapping>` tag in the mapping field definition.

```
<?xml version="1.0" encoding="UTF-8"?>
<mappings xmlns="http://dozer.sourceforge.net" xmlns:xsi="http://www.w3.org/2001/
↪XMLSchema-instance"
    xsi:schemaLocation="http://dozer.sourceforge.net
          http://dozer.sourceforge.net/schema/beanmapping.xsd">
        <!-- omitted -->
        <mapping type="one-way">
              <class-a>com.xx.xx.Source</class-a>
              <class-b>com.xx.xx.Destination</class-b>
                <field>
                  <a>id</a>
                  <b>destinationId</b>
                </field>
                <field>
                  <a>name</a>
                  <b>destinationName</b>
                </field>
        </mapping>
        <!-- omitted -->
</mappings>
```

Bean definition of conversion source

```
public class Source {
    private int id;
    private String name;
    // omitted setter/getter
}
```

Bean definition of conversion destination

```
public class Destination {
    private int destinationId;
    private String destinationName;
    // omitted setter/getter
}
```

Example of Mapping

```
Source source = new Source();
source.setId(1);
source.setName("SourceName");
Destination destination = beanMapper.map(source, Destination.class);
System.out.println(destination.getDestinationId());
System.out.println(destination.getDestinationName());
```

The output of the above code is as given below.

```
1
SourceName
```

If one-way is specified, an error does not occur even if the mapping is done in the reverse direction. The copy process is ignored. This is because `Source` field corresponding to `Destination` field does not exist if mapping is not defined.

```
Destination destination = new Destination();
destination.setDestinationId(2);
destination.setDestinationName("DestinationName");

Source source = new Source();
source.setId(1);
source.setName("SourceName");

beanMapper.map(destination, source);

System.out.println(source.getId());
System.out.println(source.getName());
```

The output of the above code is as given below.

```
1
SourceName
```

Mapping of nested fields

It is possible to map the field of Bean to be copied to the field with Nested attributes of the Bean where it is to be copied. (In Dozer terminology, it is called Deep Mapping .)

Bean definition of conversion source

```
public class EmployeeForm {
    private int id;
    private String name;
    private String deptId;
    // omitted setter/getter
}
```

Bean definition of conversion destination

```
public class Employee {
    private Integer id;
    private String name;
    private Department department;
    // omitted setter/getter
}
```

```
public class Department {
    private String deptId;
    // omitted setter/getter and other fields
}
```

Example: Define as below to map the `deptId` with `EmployeeForm` object to `deptId` of `Department`with `Employee` object.

```
<?xml version="1.0" encoding="UTF-8"?>
<mappings xmlns="http://dozer.sourceforge.net" xmlns:xsi="http://www.w3.org/2001/
↪XMLSchema-instance"
    xsi:schemaLocation="http://dozer.sourceforge.net
          http://dozer.sourceforge.net/schema/beanmapping.xsd">
    <!-- omitted -->
    <mapping map-empty-string="false" map-null="false">
        <class-a>com.xx.aa.EmployeeForm</class-a>
        <class-b>com.xx.bb.Employee</class-b>
        <field>
              <a>deptId</a>
              <b>department.deptId</b><!-- (1) -->
        </field>
    </mapping>
    <!-- omitted -->
</mappings>
```

Sr. No.	Description
(1)	Specify the field of `Employee` object for `deptId` of `Employee` form.

Example of Mapping

```
EmployeeForm source = new EmployeeForm();
source.setId(1);
source.setName("John");
source.setDeptId("D01");

Employee destination = beanMapper.map(source, Employee.class);
System.out.println(destination.getId());
System.out.println(destination.getName());
System.out.println(destination.getDepartment().getDeptId());
```

The output of the above code is as given below.

```
1
John
D01
```

In the above case, a new instance of `Employee`, a class of conversion destination is created. The newly created `Department` instance is set in the `department` field of `Employee` and `deptId` of the `EmployeeForm` is copied.

When the `Department` object is already set in the `department` field of `Employee`, a new instance is not created but the `deptId` of `EmployeeForm` is copied to the `deptId` field of the existing `Department` object.

```
EmployeeForm source = new EmployeeForm();
source.setId(1);
source.setName("John");
source.setDeptId("D01");

Employee destination = new Employee();
Department department = new Department();
destination.setDepartment(department);

beanMapper.map(source, destination);
System.out.println(department.getDeptId());
System.out.println(destination.getDepartment() == department);
```

The output of the above code is as given below.

```
D01
true
```

Collection mapping

Dozer supports two-way auto-mapping of the following Collection types. When field name is same, mapping definition XML file is not required.

- `java.util.List` <=> `java.util.List`
- `java.util.List` <=> Array
- Array <=> Array
- `java.util.Set` <=> `java.util.Set`
- `java.util.Set` <=> Array
- `java.util.Set` <=> `java.util.List`

Bean mapping with a collection of the following classes is considered.

```
package com.example.dozer;

public class Email {
    private String email;

    public Email() {
    }

    public Email(String email) {
        this.email = email;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    @Override
    public String toString() {
        return email;
    }

    // generated by Eclipse
    @Override
    public int hashCode() {
        final int prime = 31;
        int result = 1;
        result = prime * result + ((email == null) ? 0 : email.hashCode());
        return result;
    }
```

```
    // generated by Eclipse
    @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (obj == null)
            return false;
        if (getClass() != obj.getClass())
            return false;
        Email other = (Email) obj;
        if (email == null) {
            if (other.email != null)
                return false;
        } else if (!email.equals(other.email))
            return false;
        return true;
    }

}
```

Conversion source Bean

```
package com.example.dozer;

import java.util.List;

public class AccountForm {
    private List<Email> emails;

    public void setEmails(List<Email> emails) {
        this.emails = emails;
    }

    public List<Email> getEmails() {
        return emails;
    }
}
```

Conversion destination Bean

```
package com.example.dozer;

import java.util.List;

public class Account {
    private List<Email> emails;

    public void setEmails(List<Email> emails) {
        this.emails = emails;
```

```
    }

    public List<Email> getEmails() {
        return emails;
    }
}
```

Example of Mapping

```
AccountForm accountForm = new AccountForm();

List<Email> emailsSrc = new ArrayList<Email>();

emailsSrc.add(new Email("a@example.com"));
emailsSrc.add(new Email("b@example.com"));
emailsSrc.add(new Email("c@example.com"));

accountForm.setEmails(emailsSrc);

Account account = beanMapper.map(accountForm, Account.class);

System.out.println(account.getEmails());
```

The output of the above code is as given below.

```
[a@example.com, b@example.com, c@example.com]
```

There is no particular change in the explanation given so far.

As shown in the example below, **it is necessary to exercise caution when the element is already added to the Collection field of copy destination Bean.**

```
AccountForm accountForm = new AccountForm();
Account account = new Account();

List<Email> emailsSrc = new ArrayList<Email>();
List<Email> emailsDest = new ArrayList<Email>();

emailsSrc.add(new Email("a@example.com"));
emailsSrc.add(new Email("b@example.com"));
emailsSrc.add(new Email("c@example.com"));

emailsDest.add(new Email("a@example.com"));
emailsDest.add(new Email("d@example.com"));
emailsDest.add(new Email("e@example.com"));

accountForm.setEmails(emailsSrc);
account.setEmails(emailsDest);
```

```
beanMapper.map(accountForm, account);

System.out.println(account.getEmails());
```

The output of the above code is as given below.

```
[a@example.com, d@example.com, e@example.com, a@example.com, b@example.com,␣
↪c@example.com]
```

All the elements of Collection of copy source Bean are added to the Collection of copy destination Bean. Two `Email` objects with `a@exmample.com` are "Equal", but are added easily.

("Equal" here signifies `true` when compared by `Email.equals` and has the same value as `Email.hashCode`.)

The above behavior is called as **cumulative** in Dozer terminology and is a default behavior at the time of Collection mapping.

This behavior can be changed in mapping definition XML file.

```
<?xml version="1.0" encoding="UTF-8"?>
<mappings xmlns="http://dozer.sourceforge.net" xmlns:xsi="http://www.w3.org/2001/
↪XMLSchema-instance"
    xsi:schemaLocation="http://dozer.sourceforge.net
          http://dozer.sourceforge.net/schema/beanmapping.xsd">
    <!-- omitted -->
    <mapping>
        <class-a>com.example.dozer.AccountForm</class-a>
        <class-b>com.example.dozer.Account</class-b>
        <field relationship-type="non-cumulative"><!-- (1) -->
            <a>emails</a>
            <b>emails</b>
        </field>
    </mapping>
    <!-- omitted -->
</mappings>
```

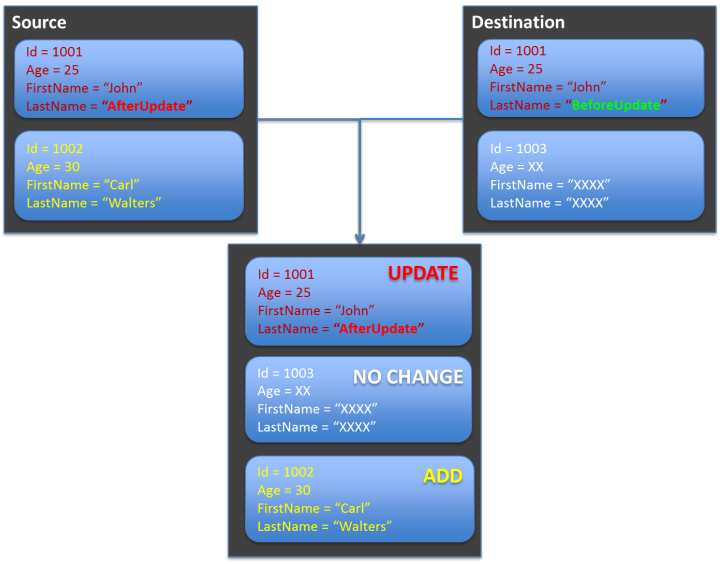
Sr. No.	Description
(1)	Specify `non-cumulative` in `relationship-type` attribute of `<field>` tag. Default value is `cumulative`. To specify `non-cumulative` in all the fields of Bean to be mapped, specify `non-cumulative` in the `relationship-type` attribute of `<mapping>` tag.

The output of the above code is as given below based on this setting.

```
[a@example.com, d@example.com, e@example.com, b@example.com, c@example.com]
```

Duplication of equivalent objects is eliminated.

Note: It is necessary to exercise caution for updating conversion source object with conversion destination object. In the above example, `a@exmample.com` of `AccountForm` is stored in the copy destination.



It can be implemented using the mapping definition XML file settings even to remove the fields existing only in the copy destination collection.

```
<?xml version="1.0" encoding="UTF-8"?>
<mappings xmlns="http://dozer.sourceforge.net" xmlns:xsi="http://www.w3.org/2001/
↪XMLSchema-instance"
    xsi:schemaLocation="http://dozer.sourceforge.net
          http://dozer.sourceforge.net/schema/beanmapping.xsd">
    <!-- omitted -->
    <mapping>
        <class-a>com.example.dozer.AccountForm</class-a>
        <class-b>com.example.dozer.Account</class-b>
        <field relationship-type="non-cumulative" remove-orphans="true" ><!-- (1) -
↪->
            <a>emails</a>
            <b>emails</b>
        </field>
    </mapping>
    <!-- omitted -->
</mappings>
```

Sr. No.	Description
(1)	Set `true` in the `remove-orphans` attribute of `<field>` tag. Default value is `false`.

The output of the above code is as given below based on this setting.

```
[a@example.com, b@example.com, c@example.com]
```

Only the objects to be copied are reflected in the collection where they are to be copied.

The same result is obtained even with the settings given below.

```
<?xml version="1.0" encoding="UTF-8"?>
<mappings xmlns="http://dozer.sourceforge.net" xmlns:xsi="http://www.w3.org/2001/
↪XMLSchema-instance"
    xsi:schemaLocation="http://dozer.sourceforge.net
          http://dozer.sourceforge.net/schema/beanmapping.xsd">
    <!-- omitted -->
    <mapping>
        <class-a>com.example.dozer.AccountForm</class-a>
        <class-b>com.example.dozer.Account</class-b>
        <field copy-by-reference="true"><!-- (1) -->
            <a>emails</a>
            <b>emails</b>
        </field>
    </mapping>
    <!-- omitted -->
</mappings>
```

Sr. No.	Description
(1)	Set `true` in the `copy-by-reference` attribute of `<field>` tag. Default value is `false`.

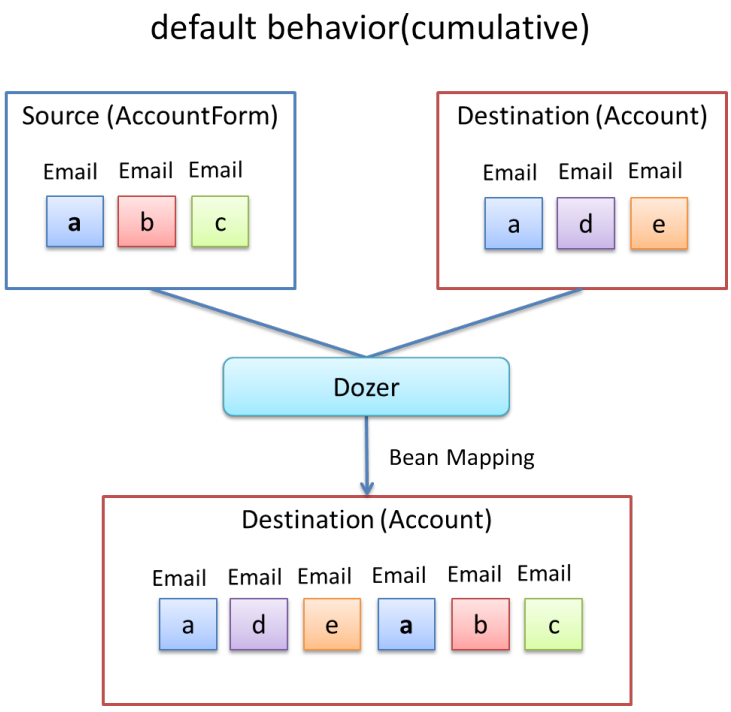
The behavior explained so far is given in the figure.

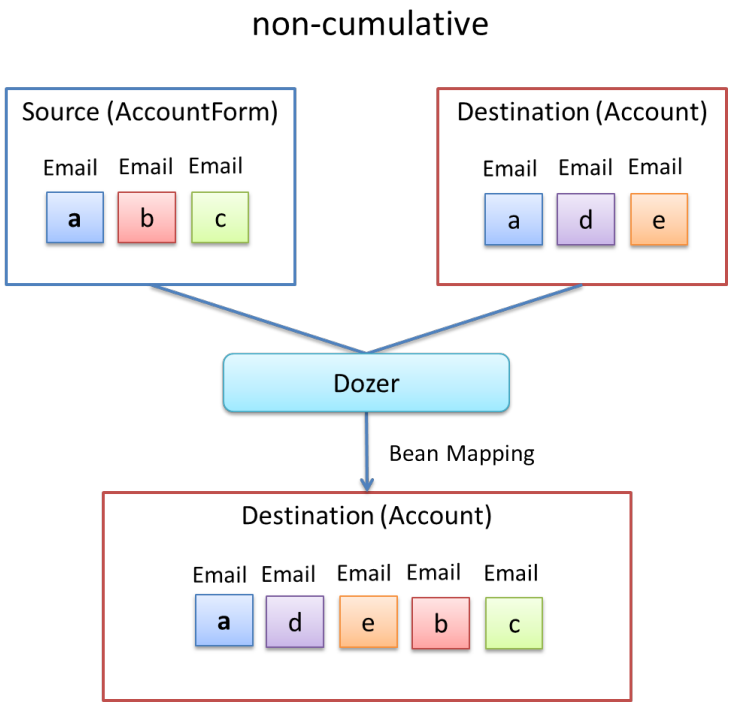
- Default behavior (cumulative)

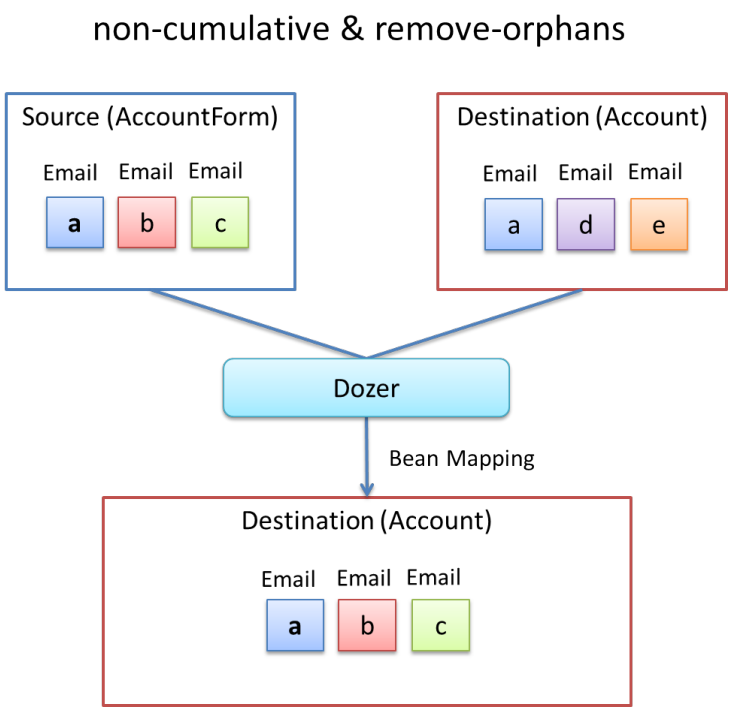
- non-cumulative

- non-cumulative and remove-orphans=true

  copy-by-reference is also of this pattern.


default behavior(cumulative)
Source (AccountForm)
Email Email Email
a b c
Destination (Account)
Email Email Email
a d e
Dozer
Bean Mapping
Destination (Account)
Email Email Email Email Email Email
a d e a b c

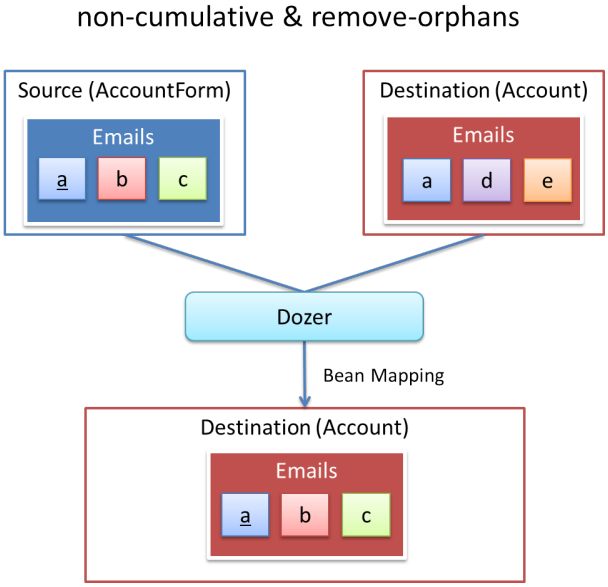

non-cumulative
Source (AccountForm)
Email Email Email
a b c
Destination (Account)
Email Email Email
a d e
Dozer
Bean Mapping
Destination (Account)
Email Email Email Email Email
a d e b c



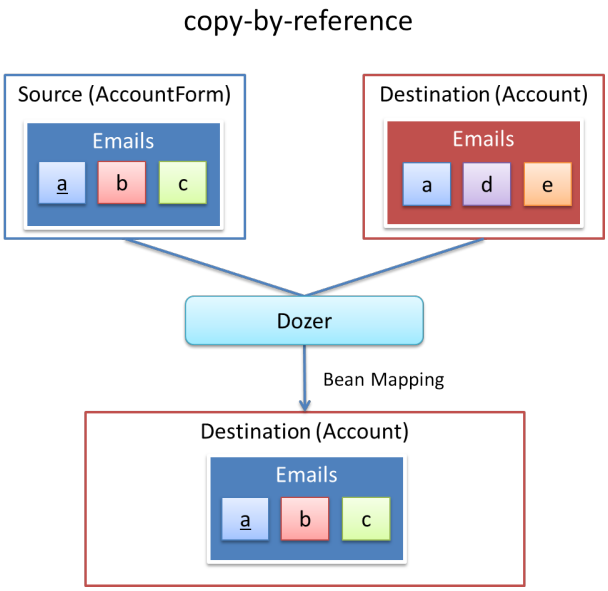
Note: The difference in "non-cumulative and remove-orphans=true" pattern and "copy-by-reference" pattern is whether the container of Collection after Bean conversion is copy destination or copy source.

In case of "non-cumulative and remove-orphans=true" pattern, the container of Collection after Bean conversion is copy destination whereas in case of "copy-by-reference" , the container is copy source. It is explained in the figure given below.

- non-cumulative and remove-orphans=true



- copy-by-reference



It is necessary to exercise caution when the copy destination has one-to many and many-to-many relationship in the entity of JPA (Hibernate). An unexpected issue may occur when copy destination entity is under EntityManager control. For example, an SQL of DELETE ALL + INSERT ALL may be issued when the container of collection is changed and an SQL to UPDATE (DELETE or INSERT when the number of elements differ) the changes may be issued when copied with "non-cumulative and remove-orphans=true". Any pattern can be used depending on the requirements.

Warning: When the Bean to be mapped has `String` collection, a bug wherein the behavior is not as expected is generated.

```
StringListSrc src = new StringListSrc;
StringListDest dest = new StringListDest();

List<String> stringsSrc = new ArrayList<String>();
List<String> stringsDest = new ArrayList<String>();

stringsSrc.add("a");
stringsSrc.add("b");
stringsSrc.add("c");

stringsDest.add("a");
stringsDest.add("d");
stringsDest.add("e");

src.setStrings(stringsSrc);
dest.setStrings(stringsDest);

beanMapper.map(src, dest);

System.out.println(dest.getStrings());
```

If code given above is executed in the non-cumulative and remove-orphans=true settings, following is expected to be output.

```
[a, b, c]
```

However, following is output.

```
[b, c]
```

and **duplicate String is removed**.

When the code is executed in the copy-by-reference="true" settings, following is output.

```
[a, b, c]
```

Tip: In Dozer, the mapping can be performed even between the lists which do not use Generics. At this time, data type of the object included in conversion source and conversion destination can be specified as HINT. Refer to Dozer Official Manual -Collection and Array Mapping(Using Hints for Collection Mapping)- for details.

Todo

It is checked that the mapping between Beans that uses Collection<T> fails.

Example :

```
public class ListNestedBean<T> {
   private List<T> nest;
   // omitted other declarations
}
```

Execution result :

```
java.lang.ClassCastException: sun.reflect.generics.reflectiveObjects.
↪TypeVariableImpl cannot be cast to java.lang.Class
```

7.7.3 How to extend

Creating custom convertor

Mapping of the data type not supported by Dozer can be performed through a custom converter.

- Example : `java.lang.String` <=> `org.joda.time.DateTime`

Custom converter is a class that implements `org.dozer.CustomConverter` provided by Dozer.

Custom converter can be specified in the following 3 patterns.

- Global Configuration
- Class level
- Field level

Global Configuration is recommended to perform conversion using the same logic in the entire application.

It is convenient to inherit `org.dozer.DozerConverter` to implement a custom converter.

```
package com.example.yourproject.common.bean.converter;

import org.dozer.DozerConverter;
import org.joda.time.DateTime;
import org.joda.time.format.DateTimeFormat;
import org.joda.time.format.DateTimeFormatter;
import org.springframework.util.StringUtils;

public class StringToJodaDateTimeConverter extends
                                        DozerConverter<String, DateTime> { // (1)
    public StringToJodaDateTimeConverter() {
        super(String.class, DateTime.class); // (2)
    }

    @Override
    public DateTime convertTo(String source, DateTime destination) {// (3)
        if (!StringUtils.hasLength(source)) {
            return null;
        }
        DateTimeFormatter formatter = DateTimeFormat
                .forPattern("yyyy-MM-dd HH:mm:ss");
        DateTime dt = formatter.parseDateTime(source);
        return dt;
    }

    @Override
    public String convertFrom(DateTime source, String destination) {// (4)
        if (source == null) {
            return null;
        }
        return source.toString("yyyy-MM-dd HH:mm:ss");
    }

}
```

Sr. No.	Description
(1)	Inherit `org.dozer.DozerConverter`.
(2)	Set 2 target classes in the constructor.
(3)	Describe the logic to convert from `String` to `DateTime`. In this example, Locale is used by default.
(4)	Describe the logic to convert from `DateTime` to `String`. In this example, Locale is used by default.

The created custom converter should be defined for mapping.

dozer-configration-mapping.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<mappings xmlns="http://dozer.sourceforge.net" xmlns:xsi="http://www.w3.org/2001/
↪XMLSchema-instance"
    xsi:schemaLocation="http://dozer.sourceforge.net
          http://dozer.sourceforge.net/schema/beanmapping.xsd">

    <configuration>
        <custom-converters><!-- (1) -->
            <!-- these are always bi-directional -->
            <converter
                type="com.example.yourproject.common.bean.converter.
↪StringToJodaDateTimeConverter"><!-- (2) -->
                <class-a>java.lang.String</class-a><!-- (3) -->
                <class-b>org.joda.time.DateTime</class-b><!-- (4) -->
            </converter>
        </custom-converters>
    </configuration>
    <!-- omitted -->
</mappings>
```

Sr. No.	Description
(1)	Define `custom-converters` to which all the custom converters belong.
(2)	Define converter for performing individual conversion. Specify fully qualified class name (FQCN) of implementation class in the converter type.
(3)	Fully qualified class name (FQCN) of conversion source Bean
(4)	Fully qualified class name (FQCN) of conversion destination Bean

When `java.lang.String` <=> `org.joda.time.DateTime` conversion needs to be performed in the entire application by using the mapping given above, the mapping is performed by calling custom converter instead of standard mapping.

Example :

Bean definition of conversion source

```
public class Source {
    private int id;
    private String date;
    // omitted setter/getter
}
```

Bean definition of conversion destination

```
public class Destination {
    private int id;
    private DateTime date;
    // omitted setter/getter
}
```

Mapping (two-way)

```
Source source = new Source();
source.setId(1);
source.setDate("2012-08-10 23:12:12");

DateTimeFormatter formatter = DateTimeFormat.forPattern("yyyy-MM-dd HH:mm:ss");
DateTime dt = formatter.parseDateTime(source.getDate());
```

```
// Source to Destination Bean Mapping (String to org.joda.time.DateTime)
Destination destination = dozerBeanMapper.map(source, Destination.class);
assertThat(destination.getId(), is(1));
assertThat(destination.getDate(),is(dt));

// Destination to Source Bean Mapping (org.joda.time.DateTime to String)
dozerBeanMapper.map(destination, source);

assertThat(source.getId(), is(1));
assertThat(source.getDate(),is("2012-08-10 23:12:12"));
```

Refer to Dozer Official Manual -Custom Converters- for the details of custom converter.

Note: The conversion from `String` to the standard date/time object such as `java.utl.Date` is explained in "*Mapping from string to date/time object*".

7.7.4 Appendix

The options that can be specified in the mapping definition XML file are explained.

All options can be confirmed in Dozer Official Manual -Custom Mappings Via Dozer XML Files-.

Field exclusion settings (field-exclude)

The fields that are not to be copied can be excluded at the time of Bean conversion.

Bean conversion is as given below.

Bean definition of conversion source

```
public class Source {
    private int id;
    private String name;
    private String title;
    // omitted setter/getter
}
```

Bean definition of copy destination

```
public class Destination {
    private int id;
    private String name;
    private String title;
```

```
    // omitted setter/getter
}
```

Define as follows to exclude any field of copy source Bean from mapping.

Carry out the settings of field exclusion in the mapping definition XML file as given below.

```
<?xml version="1.0" encoding="UTF-8"?>
<mappings xmlns="http://dozer.sourceforge.net" xmlns:xsi="http://www.w3.org/2001/
↪XMLSchema-instance"
    xsi:schemaLocation="http://dozer.sourceforge.net
          http://dozer.sourceforge.net/schema/beanmapping.xsd">
    <!-- omitted -->
    <mapping>
        <class-a>com.xx.xx.Source</class-a>
        <class-b>com.xx.xx.Destination</class-b>
        <field-exclude><!-- (1) -->
            <a>title</a>
            <b>title</b>
        </field-exclude>
    </mapping>
    <!-- omitted -->
</mappings>
```

Sr. No.	Description
(1)	Set the field you want to exclude in the <field-exclude> element. In this example, if map method is executed after specification, the title value of destination is not overwritten while copying Destination object from Source object.

```
Source source = new Source();
source.setId(1);
source.setName("SourceName");
source.setTitle("SourceTitle");

Destination destination = new Destination();
destination.setId(2);
destination.setName("DestinationName");
destination.setTitle("DestinationTitle");
beanMapper.map(source, destination);
System.out.println(destination.getId());
System.out.println(destination.getName());
System.out.println(destination.getTitle());
```

The output of the above code is as given below.

```
1
SourceName
DestinationTitle
```

The title value of destination after mapping is same as in the previous state.

Specifying mapping (map-id)

The mapping indicated in *Field exclusion settings (field-exclude)* is applied while performing Bean conversion in the entire application. To specify the applicable range of mapping, define map-id as given below.

```
<?xml version="1.0" encoding="UTF-8"?>
<mappings xmlns="http://dozer.sourceforge.net" xmlns:xsi="http://www.w3.org/2001/
↪XMLSchema-instance"
    xsi:schemaLocation="http://dozer.sourceforge.net
          http://dozer.sourceforge.net/schema/beanmapping.xsd">
    <!-- omitted -->
    <mapping map-id="mapidTitleFieldExclude">
        <class-a>com.xx.xx.Source</class-a>
        <class-b>com.xx.xx.Destination</class-b>
        <field-exclude>
            <a>title</a>
            <b>title</b>
        </field-exclude>
    </mapping>
    <!-- omitted -->
</mappings>
```

When the above settings are carried out, title can be excluded from copying by passing map-id (mapidTitleField-Exclude) to `map` method. When map-id is not specified, all the fields are copied without applying the settings.

The example of passing map-id to `map` method is shown below.

```
Source source = new Source();
source.setId(1);
source.setName("SourceName");
source.setTitle("SourceTitle");

Destination destination1 = new Destination();
destination1.setId(2);
destination1.setName("DestinationName");
destination1.setTitle("DestinationTitle");
beanMapper.map(source, destination1); // (1)
System.out.println(destination1.getId());
```

```
System.out.println(destination1.getName());
System.out.println(destination1.getTitle());

Destination destination2 = new Destination();
destination2.setId(2);
destination2.setName("DestinationName");
destination2.setTitle("DestinationTitle");
beanMapper.map(source, destination2, "mapidTitleFieldExclude"); // (2)
System.out.println(destination2.getId());
System.out.println(destination2.getName());
System.out.println(destination2.getTitle());
```

Sr. No.	Description
(1)	Normal mapping.
(2)	Pass map-id as the third argument and apply the specific mapping rules.

The output of the above code is as given below.

```
1
SourceName
SourceTitle

1
SourceName
DestinationTitle
```

Tip: map-id can be specified not only by mapping items but also by defining the field. Refer to Dozer Official Manual -Context Based Mapping- for details.

Note: The same form object can be used for both creating new/updating the existing Web applications. Form object is copied (mapped) to domain object, however a field may exist where the object is not to be copied depending on the operations. In this case, use `<field-exclude>`.

- Example: userId is included in the New form whereas it is not included in the Update form.

In this case, null is set in the userId at the time of update if the same form object is used. If form object is copied as it is by fetching copy destination object from DB, userId of copy destination becomes null. In order to avoid this, provide map-id for update and carry out the settings to exclude field of userId at the time of update.

Settings to exclude null/empty field of copy source (map-null, map-empty)

`null` or empty field of copy source Bean can be excluded from mapping. Set in the mapping definition XML file as given below.

```
<?xml version="1.0" encoding="UTF-8"?>
<mappings xmlns="http://dozer.sourceforge.net" xmlns:xsi="http://www.w3.org/2001/
↪XMLSchema-instance"
    xsi:schemaLocation="http://dozer.sourceforge.net
          http://dozer.sourceforge.net/schema/beanmapping.xsd">
    <!-- omitted -->
    <mapping map-null="false" map-empty-string="false"><!-- (1) -->
        <class-a>com.xx.xx.Source</class-a>
        <class-b>com.xx.xx.Destination</class-b>
    </mapping>
    <!-- omitted -->
</mappings>
```

Sr. No.	Description
(1)	To exclude the `null` field of copy source Bean from mapping, set `false` in the `map-null` attribute. Default value is `true`. To exclude the empty field of copy source Bean from mapping, set `false` in the `map-empty-string` attribute. Default value is `true`.

Bean definition of conversion source

```
public class Source {
    private int id;
```

```
    private String name;
    private String title;
    // omitted setter/getter
}
```

Bean definition of conversion destination

```
public class Destination {
    private int id;
    private String name;
    private String title;
    // omitted setter/getter
}
```

Mapping example

```
Source source = new Source();
source.setId(1);
source.setName(null);
source.setTitle("");

Destination destination = new Destination();
destination.setId(2);
destination.setName("DestinationName");
destination.setTitle("DestinationTitle");
beanMapper.map(source, destination);
System.out.println(destination.getId());
System.out.println(destination.getName());
System.out.println(destination.getTitle());
```

The output of the above code is as given below.

```
1
DestinationName
DestinationTitle
```

`name` and `title` field of copy source Bean are `null` or empty, and are excluded from mapping.

Mapping from string to date/time object

The copy source string field can be mapped to the copy destination date/time field.

Following 6 types of conversions are supported.

Date/time

- `java.lang.String` **<=>** `java.util.Date`
- `java.lang.String` **<=>** `java.util.Calendar`

- `java.lang.String` <=> `java.util.GregorianCalendar`
- `java.lang.String` <=> `java.sql.Timestamp`

Date only

- `java.lang.String` <=> `java.sql.Date`

Time only

- `java.lang.String` <=> `java.sql.Time`

Perform date/time conversion as follows.

For example, the conversion to `java.util.Date` is explained.

Conversions for `java.util.Calendar`,`java.util.GregorianCalendar`,`java.sql.Timestamp` can also be performed by the same method.

```
<?xml version="1.0" encoding="UTF-8"?>
<mappings xmlns="http://dozer.sourceforge.net" xmlns:xsi="http://www.w3.org/2001/
↪XMLSchema-instance"
    xsi:schemaLocation="http://dozer.sourceforge.net
          http://dozer.sourceforge.net/schema/beanmapping.xsd">
    <!-- omitted -->
    <mapping>
        <class-a>com.xx.xx.Source</class-a>
        <class-b>com.xx.xx.Destination</class-b>
        <field>
            <a date-format="yyyy-MM-dd HH:mm:ss:SS">date</a><!-- (1) -->
            <b>date</b>
        </field>
    </mapping>
    <!-- omitted -->
</mappings>
```

Sr. No.	Description
(1)	Specify field name and date format of copy source.

Bean definition of conversion source

```
public class Source {
    private String date;
    // omitted setter/getter
}
```

Bean definition of conversion destination

```
public class Destination {
    private Date date;
    // omitted setter/getter
}
```

Mapping

```
Source source = new Source();
source.setDate("2013-10-10 11:11:11.111");
Destination destination = beanMapper.map(source, Destination.class);
assert(destination.getDate().equals(new SimpleDateFormat("yyyy-MM-dd HH:mm:ss.SSS
↪").parse("2013-10-10 11:11:11.111")));
```

There are many cases where it is preferable to set a common date format in the project rather than setting an individual date for each mapping definition.

In such a case, it is recommended to set in the Global configuration file of Dozer.

The date format set in the mapping of the entire application is applied.

```
<?xml version="1.0" encoding="UTF-8"?>
<mappings xmlns="http://dozer.sourceforge.net" xmlns:xsi="http://www.w3.org/2001/
↪XMLSchema-instance"
    xsi:schemaLocation="http://dozer.sourceforge.net
          http://dozer.sourceforge.net/schema/beanmapping.xsd">
    <!-- omitted -->
    <configuration>
        <date-format>yyyy-MM-dd HH:mm:ss.SSS</date-format>
        <!-- omitted other configuration -->
    </configuration>
    <!-- omitted -->
</mappings>
```

There are no restrictions for the file name however,
src/main/resources/META-INF/dozer/dozer-configration-mapping.xml is recommended.

Global Configuration that impacts the entire application in this configuration file can be used within the scope of settings for dozer-configration-mapping.xml.

Refer to the Dozer Official Manual -Global Configuration- for the details of items that can be configured.

Mapping error

If mapping process fails during mapping, `org.dozer.MappingException` (runtime exception) is thrown.

The typical examples of `MappingException` being thrown are given below.

- map-id that does not exist in the `map` method is passed.
- map-id existing in `map` method is passed however, the source/target type passed to map process differs from the definition specified in map-id.
- When conversion is not supported by Dozer and custom converter for conversion does not exist as well.

Since these are normal program bugs, the sections called by `map` method should be modified appropriately.

8

Messaging

8.1 Sending E-mail (SMTP)

8.1.1 Overview

This chapter explains how to send an E-mail using SMTP.

In this guideline, it is assumed that a component for email coordination offered by API and Spring Framework of JavaMail is used.

Note: The description covers only the part related to sending an email. The processing method related to sending an email is not mentioned. (An example is introduced for *Processing method*.)

Regarding JavaMail

JavaMail offers an API to send and receive emails in Java. Although it is included in Java EE, it can also be used in Java SE as an add-on package. By using JavaMail, the email function can be easily incorporated in Java application.

Also, since it is assumed that an email coordination component of Spring Framework is used in this guideline, the details related to JavaMail API are not covered. Refer JavaMail API Design Specification for JavaMail API specifications.

Note: Mail Session

A mail session (Session) manages the information required for connecting to a mail server.

Following methods are used to fetch a mail session.

- Fetch the mail session managed by Java EE container through JNDI for a typical enterprise application.
- Fetch the mail session defined by resource factory through JNDI in case of Tomcat.
- Fetch the mail session for which a Bean is defined, from DI container using a static factory method.
- Fetch directly from Java source using static factory method of `Session`.

Note that, if `JavaMailSenderImpl` of Spring described later is used, it is possible to connect to a mail server without handling a mail session directly.

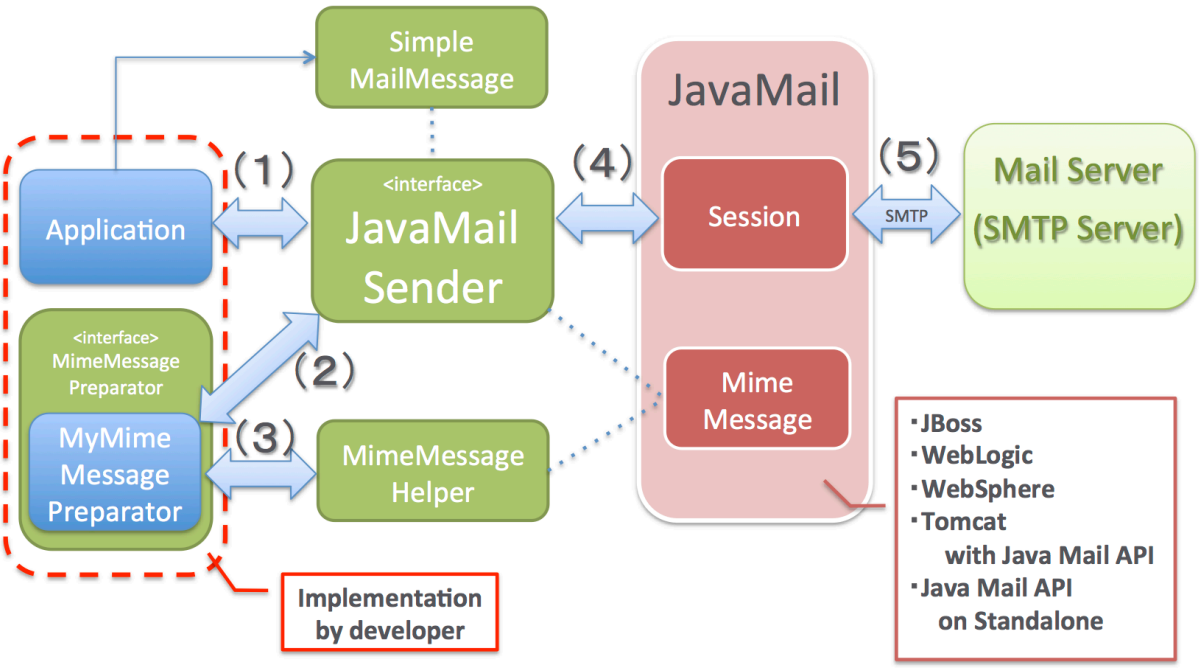
Implementation examples using two methods given below are introduced in this guideline.

- A method wherein a mail session is fetched through JNDI
- A method wherein connection information is specified in `JavaMailSenderImpl` property without handling a session directly

Regarding component of Spring Framework for email coordination

Spring Framework offers a component (`org.springframework.mail` package) for sending an email. The component included in the package conceals the detail logic related to sending an email and carry out low level API handling (API calling of JavaMail).

The method by which the component offered by Spring Framework for email transmission sends an email is explained before the explanation of basic implementation methods.



Sr. No.	Component	Description
(1)	Application	Call `JavaMailSender` method and request to send an email. * While sending a simple message, an email can also be sent by generating `SimpleMailMessage` and specifying address and body text.
(2)	`JavaMailSender`	Call `MimeMessagePreparator` (callback interface to create `MimeMessage` of JavaMail) specified by the application and request to create a message for sending an email (`MimeMessage`). This process is not called while sending the message using * `SimpleMailMessage`.

次のページに続く

Table. 8.1 – 前のページからの続き

Sr. No.	Component	Description
(3)	Application (`MimeMessagePreparator`)	Create a message (`MimeMessage`) for sending the email using `MimeMessageHelper` method. This process is not called while sending the message using * `SimpleMailMessage`.
(4)	`JavaMailSender`	Request to send the email using API of JavaMail.
(5)	JavaMail	Send a message to the email server.

The method to implement a process for sending an email using interface and class below is explained in this guideline.

- `JavaMailSender`

  An interface for JavaMail to send an email.

  It supports both MimeMessage of JavaMail and `SimpleMailMessage` of Spring.

  Further, since `Session` of JavaMail is managed by implementation class of `JavaMailSender` , it is not necessary to handle `Session` directly while writing the code for the process to send an email.

- `JavaMailSenderImpl`

  An implementation class of `JavaMailSender` interface.

  This class supports a method wherein DI is applied to configured `Session` and a method wherein `Session` is created from connection information specified in property.

- `MimeMessagePreparator`

  A callback interface for creating `MimeMessage` of JavaMail.

  It is called from `send` method of `JavaMailSender`.

  The exception generated in `prepare` method of `MimeMessagePreparator` is wrapped in `MailPreparationException` (runtime exception) and thrown again.

- `MimeMessageHelper`

  A helper class to facilitate creation of `MimeMessage` of JavaMail.

  `MimeMessageHelper` provides a number of convenient methods to specify a value in `MimeMessage`.

- `SimpleMailMessage`

  A class to create a simple email message.

  It can be used to create a plain text email in English.

  A `MimeMessage` of JavaMail must be used for creating rich messages such as specifying specific encoding like UTF-8, sending HTML emails and emails with attachments or associating personal names with email addresses.

8.1.2 How to use

Regarding dependent library

When a component of Spring Framework for email coordination is used, following libraries must be added.

- JavaMail

Add a dependency relation for library above to `pom.xml`.

In case of a multi-project configuration, it is added to `pom.xml` (`projectName-domain/pom.xml`) of domain project.

```
<dependencies>

    <!-- (1) -->
    <dependency>
        <groupId>com.sun.mail</groupId>
        <artifactId>javax.mail</artifactId>
    </dependency>

</dependencies>
```

Sr. No.	Description
(1)	Add JavaMail libraries to dependencies. Set `<scope>` to `provided` while using a mail session offered by application server.

Note: In the above setting example, since it is assumed that the dependent library version is managed by the parent project terasoluna-gfw-parent, specifying the version in pom.xml is not necessary. The above dependent library used by terasoluna-gfw-parent is defined by Spring IO Platform.

How to configure JavaMailSender

Define a Bean for applying DI to `JavaMailSender`.

Note: In case of a multi-project configuration, it is recommended to set in `projectName-env.xml` of env project. Note that, in this guideline, it is recommended to adopt a multi-project configuration.

When a mail session offered by application server is used

A configuration example while using a mail session offered by application server is given below.

Table.8.2 **Mail session offered by Application Server**

Sr. No.	Application server	Refer page
1.	Apache Tomcat 8.5	Refer Apache Tomcat 8.5 User Guide(JNDI Resources HOW-TO) (JavaMail Sessions).
2.	Apache Tomcat 8.0	Refer Apache Tomcat 8.0 User Guide(JNDI Resources HOW-TO)(JavaMail Sessions).
3.	Oracle WebLogic Server 12c	Refer Oracle WebLogic Server 12.2.1.0 Documentation.
4.	IBM WebSphere Application Server Version 9.0	Refer WebSphere Application Server Version 9.0.0 documentation.
5.	Red Hat JBoss Enterprise Application Platform Version 7.0	Refer JBoss Enterprise Application Platform 7.0 Product Documentation.
6.	Red Hat JBoss Enterprise Application Platform Version 6.4	Refer JBoss Enterprise Application Platform 6.4 Product Documentation.

Carry out setup for registering a mail session fetched through JNDI, as a Bean.

```
<jee:jndi-lookup id="mailSession" jndi-name="mail/Session" /> <!-- (1) -->
```

Sr. No.	Description
(1)	Specify JNDI name of mail session offered by application server in `jndi-name` attribute of `<jee:jndi-lookup>` element.

Next, define a Bean for `JavaMailSender`.

```
<!-- (1) -->
<bean id="mailSender" class="org.springframework.mail.javamail.JavaMailSenderImpl">
    <property name="session" ref="mailSession" /> <!-- (2) -->
</bean>
```

Sr. No.	Description
(1)	Define a Bean for `JavaMailSenderImpl`.
(2)	Specify a Bean of configured mail session in `session` property.

When a mail session offered by application server is not used (no authentication)

A configuration example wherein authentication is not required is given below.

Define a Bean for `JavaMailSender`.

```
<!-- (1) -->
<bean id="mailSender" class="org.springframework.mail.javamail.JavaMailSenderImpl">
    <property name="host" value="${mail.smtp.host}"/> <!-- (2) -->
    <property name="port" value="${mail.smtp.port}"/> <!-- (3) -->
</bean>
```

Sr. No.	Description
(1)	Define a Bean for `JavaMailSenderImpl`.
(2)	Specify a host name of SMTP server in `host` property. In this example, a value defined in the property file (value corresponding to key "`mail.smtp.host`") is specified.
(3)	Specify a port number of SMTP server in `port` property. In this example, a value defined in the property file (value corresponding to key "`mail.smtp.port`") is specified.

Note: Refer *Properties Management* for details of property file.

When a mail session offered by application server is not used (authenticated)

A configuration example wherein authentication is required is given below.

Define a Bean for `JavaMailSender`.

```
<bean id="mailSender" class="org.springframework.mail.javamail.JavaMailSenderImpl">
    <property name="host" value="${mail.smtp.host}"/>
    <property name="port" value="${mail.smtp.port}"/>
    <property name="username" value="${mail.smtp.user}"/> <!-- (1) -->
    <property name="password" value="${mail.smtp.password}"/> <!-- (2) -->
    <property name="javaMailProperties">
        <props>
            <prop key="mail.smtp.auth">true</prop> <!-- (3) -->
        </props>
    </property>
</bean>
```

Sr. No.	Description
(1)	Specify a user name of SMTP server in `username` property. In this example, a value defined in property file (value corresponding to key "`mail.smtp.user`") is specified.
(2)	Specify a password of SMTP server in `password` property. In this example, a value defined in the property file (value corresponding to key "`mail.smtp.password`") is specified.
(3)	Set `true` in `javaMailProperties` property as a key "`mail.smtp.auth`".

Note: Refer *Properties Management* for details of property file.

Tip: When the connection using TLS is necessary, set `true` in `javaMailProperties` property as a key "`mail.smtp.starttls.enable`". Note that, when SMTP server does not support STARTTLS even when specified as below, plain text is used for communication. When `true` is set in `javaMailProperties` property as a key "`mail.smtp.starttls.required`" whenever required, an error can occur if it is not possible to use STARTTLS.

How to send an email using SimpleMailMessage

When a plain text email (an email which does not require encode specification or attachments) is to be sent in English, `SimpleMailMessage` class offered by Spring is used.

A method to send an email using `SimpleMailMessage` class is explained below.

Example of Bean definition

```
<!-- (1) -->
<bean id="templateMessage" class="org.springframework.mail.SimpleMailMessage">
    <property name="from" value="info@example.com" /> <!-- (2) -->
    <property name="subject" value="Registration confirmation." /> <!-- (3) -->
</bean>
```

Sr. No.	Description
(1)	Define a Bean for `SimpleMailMessage` as a template. Although it is not mandatory to use `SimpleMailMessage` of template, if a fixed location is specified (for e.g. sender email address etc) in the email message as a template, it is not necessary to individually specify it later in the email message.
(2)	Specify details of From header in `from` property.
(3)	Specify details of Subject header in `subject` property.

Implementation example of Java class

```
@Inject
JavaMailSender mailSender; // (1)

@Inject
SimpleMailMessage templateMessage; // (2)

public void register(User user) {
    // omitted

    // (3)
    SimpleMailMessage message = new SimpleMailMessage(templateMessage);
    message.setTo(user.getEmailAddress());
    String text = "Hi "
            + user.getUserName()
            + ", welcome to EXAMPLE.COM!\r\n"
            + "If you were not an intended recipient, Please notify the sender.";
```

```
    message.setText(text);
    mailSender.send(message);

    // omitted
}
```

Sr. No.	Description
(1)	Inject `JavaMailSender`.
(2)	Inject `SimpleMailMessage` as a template for which a Bean is defined.
(3)	Generate a `SimpleMailMessage` instance by using Bean of template, specify To header and body text, and send the message.

Note:

Table.8.3 **Properties that can be set in SimpleMailMessage**

Sr. No.	Property	Description
1.	`from`	Specify From header.
2.	`to`	Specify To header.
3.	`cc`	Specify Cc header.
4.	`bcc`	Specify Bcc header.
5.	`subject`	Specify Subject header.
6.	`replyTo`	Specify Reply-To header.
7.	`sentDate`	Specify Date header. Note that, if it is not explicitly set, system time (`new Date()`) is set automatically at the time of sending an email.
8.	`text`	Specify body text.

When multiple addresses are to be specified in To, Cc and Bcc, specify addresses in an array.

Warning: While setting an email header, an email header injection must be considered. Refer *Email header injection countermeasures* for details.

How to send an email using MimeMessage

When a non-English text email, HTML email and attachments are to be sent, `javax.mail.internet.MimeMessage` class is used. In this guideline, a method to create `MimeMessage` by using `MimeMessageHelper` class is recommended.

In this section, the methods to send an email using `MimeMessageHelper` class are explained below.

- *Sending a text email*
- *Sending a HTML email*
- *Sending an email with attachment*
- *Sending an email with inline resource*

Sending a text email

An implementation example wherein a text email is sent using `MimeMessageHelper` class is given below.

Implementation example of Java class

```
@Inject
JavaMailSender mailSender; // (1)

public void register(User user) {
    // omitted

    // (2)
    mailSender.send(new MimeMessagePreparator() {

        @Override
        public void prepare(MimeMessage mimeMessage) throws Exception {
            MimeMessageHelper helper = new MimeMessageHelper(mimeMessage,
                    StandardCharsets.UTF_8.name()); // (3)
            helper.setFrom("EXAMPLE.COM <info@example.com>"); // (4)
            helper.setTo(user.getEmailAddress()); // (5)
            helper.setSubject("Registration confirmation."); // (6)
            String text = "Hi "
                    + user.getUserName()
                    + ", welcome to EXAMPLE.COM!\r\n"
                    + "If you were not an intended recipient, Please notify the␣
↪sender.";
            helper.setText(text); // (7)
        }
    });
```

```
    // omitted
}
```

Sr. No.	Description
(1)	Inject `JavaMailSender`.
(2)	Send an email using `send` method of `JavaMailSender`. Define an anonymous inner class that implements `MimeMessagePreparator`, in the argument.
(3)	Specify character code and generate `MimeMessageHelper` instance. In this example, UTF-8 is specified in the character code.
(4)	Specify details of From header. In this example, it is set in “Name <Address>” format.
(5)	Specify details of To header.
(6)	Specify details of Subject header.
(7)	Specify details of body text.

Warning: While setting an email header, an email header injection must be considered. Refer *Email header injection countermeasures* for details.

Note: While sending an email in Japanese, ISO-2022-JP can also be used in encoding if it is also necessary to support a mail client which does not support UTF-8. Refer *Considerations for ISO-2022-JP encoding* for the points that should be considered while using ISO-2022-JP in encoding.

Sending a HTML email

An implementation example wherein a HTML email is sent using `MimeMessageHelper` class is shown below.

Implementation example of Java class

```
@Inject
JavaMailSender mailSender; // (1)

public void register(User user) {
    // omitted

    // (2)
    mailSender.send(new MimeMessagePreparator() {

        @Override
        public void prepare(MimeMessage mimeMessage) throws Exception {
            MimeMessageHelper helper = new MimeMessageHelper(mimeMessage,
                    StandardCharsets.UTF_8.name()); // (3)
            helper.setFrom("EXAMPLE.COM <info@example.com>"); // (4)
            helper.setTo(user.getEmailAddress()); // (5)
            helper.setSubject("Registration confirmation."); // (6)
            String text = "<html><body><h3>Hi "
                    + user.getUserName()
                    + ", welcome to EXAMPLE.COM!</h3>"
                    + "If you were not an intended recipient, Please notify the␣
↪sender.</body></html>";
            helper.setText(text, true); // (7)
        }
    });

    // omitted
}
```

Sr. No.	Description
(1)	Inject `JavaMailSender`.
(2)	Send an email using `send` method of `JavaMailSender`. Define an anonymous inner class that implements `MimeMessagePreparator`, in the argument.
(3)	Specify character code and generate `MimeMessageHelper` instance. In this example, UTF-8 is specified in the character code.
(4)	Specify details of From header. In this example, it is set in the “Name <Address>” format.
(5)	Specify details of To header.
(6)	Specify details of Subject header.
(7)	Specify details of body text. Content-Type changes to text/html by specifying `true` in the second argument of `setText` method.

Warning: When a value entered externally is used while generating HTML for email text, countermeasures for XSS attack should be employed.

Sending an email with attachment

An implementation example wherein an email with the attachment is sent using `MimeMessageHelper` class is shown below.

Implementation example of Java class

```
@Inject
JavaMailSender mailSender; // (1)

public void register(User user) {
    // omitted

    // (2)
    mailSender.send(new MimeMessagePreparator() {

        @Override
        public void prepare(MimeMessage mimeMessage) throws Exception {
            MimeMessageHelper helper = new MimeMessageHelper(mimeMessage,
                    true, StandardCharsets.UTF_8.name()); // (3)
            helper.setFrom("EXAMPLE.COM <info@example.com>"); // (4)
            helper.setTo(user.getEmailAddress()); // (5)
            helper.setSubject("Registration confirmation."); // (6)
            String text = "Hi "
                    + user.getUserName()
                    + ", welcome to EXAMPLE.COM!\r\n"
                    + "Please find attached the file.\r\n\r\n"
                    + "If you were not an intended recipient, Please notify the␣
↪sender.";
            helper.setText(text); // (7)
            ClassPathResource file = new ClassPathResource("doc/quickstart.pdf");
            helper.addAttachment("QuickStart.pdf", file); // (8)
        }
    });

    // omitted
}
```

Sr. No.	Description
(1)	Inject `JavaMailSender`.
(2)	Send an email using `send` method of `JavaMailSender`. Define an anonymous inner class that implements `MimeMessagePreparator`, in the argument.
(3)	Specify character code and generate `MimeMessageHelper` instance. In this example, UTF-8 is specified in the character code. It becomes multi-part mode (default `MULTIPART_MODE_MIXED_RELATED`) by specifying `true` in the second argument of the constructor of `MimeMessageHelper`.
(4)	Specify details of From header.
(5)	Specify details of To header.
(6)	Specify details of Subject header.
(7)	Specify details of body text.
(8)	Specify attachment name and identify file to be attached. In this example, `"QuickStart.pdf"` denotes the file name and `doc/quickstart.pdf` file on the class path is attached.

Sending an email with inline resource

An implementation example wherein an email with inline resource is sent using `MimeMessageHelper` class is shown below.

Implementation example of Java class

```
@Inject
JavaMailSender mailSender; // (1)

public void register(User user) {
    // omitted

    // (2)
    mailSender.send(new MimeMessagePreparator() {

        @Override
        public void prepare(MimeMessage mimeMessage) throws Exception {
            MimeMessageHelper helper = new MimeMessageHelper(mimeMessage,
                    true, StandardCharsets.UTF_8.name()); // (3)
            helper.setFrom("EXAMPLE.COM <info@example.com>"); // (4)
            helper.setTo(user.getEmailAddress()); // (5)
            helper.setSubject("Registration confirmation."); // (6)
            String cid = "identifier1234";
            String text = "<html><body><img src='cid:"
                    + cid
                    + "' /><h3>Hi "
                    + user.getUserName()
                    + ", welcome to EXAMPLE.COM!\r\n</h3>"
                    + "If you were not an intended recipient, Please notify the␣
↪sender.</body></html>";
            helper.setText(text, true); // (7)
            ClassPathResource res = new ClassPathResource("image/logo.jpg");
            helper.addInline(cid, res); // (8)
        }
    });

    // omitted
}
```

Sr. No.	Description
(1)	Inject `JavaMailSender`.
(2)	Send an email using `send` method of `JavaMailSender`. Define an anonymous inner class that implements `MimeMessagePreparator` , in the argument.
(3)	Specify character code and generate `MimeMessageHelper` instance. In this example, UTF-8 is specified in the character code. It becomes a multi-part mode by specifying `true` in the second argument of constructor of `MimeMessageHelper`.
(4)	Specify details of From header.
(5)	Specify details of To header.
(6)	Specify details of Subject header.
(7)	Specify details of body text. Content-Type changes to text/html by specifying `true` in the second argument of `setText` method.
(8)	Specify inline resource contents ID and set inline resource. In this example, `"identifier1234"` denotes a content ID and `image/logo.jpg` file on the class path is set.

Note: `addInline` method should be called after `setText` method. If done otherwise, a mail client cannot view the inline resource correctly.

Regarding exceptions while sending an email

The exception that occurs while sending an email using `send` method of `JavaMailSender` is an exception which inherits `org.springframework.mail.MailException`. The exception class that inherits `MailException` and occurrence conditions of respective exceptions are shown in the table below.

Table.8.4 **Exceptions at the time of sending an email**

Sr. No.	Exception class	Occurrence conditions
1.	MailAuthenticationException	Occurs during authentication failure.
2.	MailParseException	Occurs when an invalid value is set in the properties of email message.
3.	MailPreparationException	Occurs if an unexpected error occurs while creating an email message. Unexpected errors, for example, are the errors that occur in the template library. Exceptions occurring in `MimeMessagePreparator` are wrapped in `MailPreparationException` and thrown.
4.	MailSendException	Occurs when an error occurs while sending an email.

Note: Refer *Exception Handling* for transition to error screen corresponding to specific exceptions.

8.1.3 How to extend

How to create an email text using a template

It is not recommended to build an email text directly in Java source as shown in the implementation examples above for the reasons given below.

- Building the email text in Java source causes poor readability and it may cause an error.
- Boundary between display logic and business logic become ambiguous.
- It becomes necessary to modify, compile and deploy Java source in order to change the email text design.

Hence, it is recommended to use a template library to define an email text design. A template library must especially be used when the email text is particularly complex.

Creating the email text using FreeMarker

In this guideline, a method that uses FreeMarker as a template library is described.

- Set a dependent library for using FreeMarker.

  Configuration example of pom.xml

```
<dependencies>

    <!-- (1) -->
    <dependency>
        <groupId>org.freemarker</groupId>
        <artifactId>freemarker</artifactId>
    </dependency>

</dependencies>
```

Sr. No.	Description
(1)	Add a FreeMarker library to dependencies.

Note: In the above setting example, since it is assumed that the dependent library version is managed by the parent project terasoluna-gfw-parent , specifying the version in pom.xml is not necessary. The above dependent library used by terasoluna-gfw-parent is defined by Spring IO Platform.

- Define a Bean for FactoryBean to generate `freemarker.template.Configuration`.

  Configuration example of Bean definition file

```
<!-- (1) -->
<bean id="freemarkerConfiguration"
    class="org.springframework.ui.freemarker.
↪FreeMarkerConfigurationFactoryBean">
    <property name="templateLoaderPath" value="classpath:/META-INF/
↪freemarker/" /> <!-- (2) -->
    <property name="defaultEncoding" value="UTF-8" /> <!-- (3) -->
</bean>
```

Sr. No.	Description
(1)	Define a Bean for `FreeMarkerConfigurationFactoryBean`.
(2)	Specify the location of template file storage in `templateLoaderPath` property. In this example, `META-INF/freemarker/` directory on the class path is specified.
(3)	Specify default encoding in `defaultEncoding` property. In this example, UTF-8 is specified.

Note: Refer to JavaDoc of FreeMarkerConfigurationFactoryBean for the setup other than mentioned above. Also, refer FreeMarker Manual (Programmer's Guide / The Configuration) for setup of FreeMarker itself.

- Create a template file for email text.

Configuration example of template file

```
<#escape x as x?html> <#-- (1) -->
<html>
    <body>
        <h3>Hi ${userName}, welcome to TERASOLUNA.ORG!</h3> <#-- (2) -->

        <div>
            If you were not an intended recipient, Please notify the␣
↪sender.
        </div>
    </body>
</html>
</#escape>
```

Sr. No.	Description
(1)	Specify to apply HTML escape as a countermeasure for XSS attack.
(2)	Embed the value of `userName` specified in the data model.

Note: Refer FreeMarker Manual (Template Language Reference) for details of template language (FIL).

- Generate an email text using a template and send email.

Implementation example of Java class

```
@Inject
JavaMailSender mailSender;

@Inject
Configuration freemarkerConfiguration; // (1)

public void register(User user) {
    // omitted

    mailSender.send(new MimeMessagePreparator() {

        @Override
        public void prepare(MimeMessage mimeMessage) throws Exception {
            MimeMessageHelper helper = new MimeMessageHelper(mimeMessage,
                    StandardCharsets.UTF_8.name());
            helper.setFrom("EXAMPLE.COM <info@example.com>");
            helper.setTo(user.getEmailAddress());
            helper.setSubject("Registration confirmation.");
            Template template = freemarkerConfiguration
                    .getTemplate("registration-confirmation.ftl"); // (2)
            String text = FreeMarkerTemplateUtils
                    .processTemplateIntoString(template, user); // (3)
            helper.setText(text, true);
        }
    });

    // omitted
}
```

Sr. No.	Description
(1)	Inject Configuration.
(2)	Fetch Template using `getTemplate` method of `Configuration`. In this example, "registration-confirmation.ftl" is specified as a template file.
(3)	Based on fetched `Template`, generate a string from the template using `processTemplateIntoString` method of `org.springframework.ui.freemarker.FreeMarkerTemplateUtils`. In this example, `User` object (JavaBeans) consisting of `userName` property is specified as a data model. Accordingly, value of `userName` property is embedded in the location of `${userName}` of template file.

8.1.4 Appendix

Considerations for ISO-2022-JP encoding

When sending an email in Japanese, if the email client that receives the sent mail cannot be restricted, it is necessary to consider use of ISO-2022-JP in encoding. This is because the legacy email client does not support UTF-8.

When encoding based on character set of JIS X 0208 including ISO-2022-JP is set for the string entered by MS932, garbling occurs for seven characters described in the table below.

Before conversion			After conversion		
MS932 Input character	Input value（SJIS）	Unicode（UTF-16）	Unicode（UTF-16）	ISO-2022-JP（JIS）	JIS X 0208 Alternative character
—(Double byte hyphen)	815D	U+2015	U+2014	213E	—(EM dash)
－ (Hyphen-minus)	817C	U+FF0D	U+2212	215D	－ (Double byte minus)
～(Double byte tilde)	8160	U+FF5E	U+301C	2141	～(Tilde)
∥ (Parallel symbol)	8161	U+2225	U+2016	2142	‖ (Pipe sumbol)
￠ (Double byte cent symbol)	8191	U+FFE0	U+00A2	2171	¢ (Cent symbol)
￡ (Double byte pound symbol)	8192	U+FFE1	U+00A3	2172	£ (Pound symbol)
￢ (Double byte negation symbol)	81CA	U+FFE2	U+00AC	224C	¬ (Negation symbol)

This issue occurs during character code conversion through Unicode due to the presence of characters that exist in MS932 but do not exist in JIS X 0208. In order to avoid garbling, the measures such as replacing character codes for the garbled characters with alternate characters must be employed. Note that, conversion process is not necessary while using x-windows-iso2022jp described later.

Implementation example of conversion process is shown below.

```
public static String convertISO2022JPCharacters(String targetStr) {

    if (targetStr == null) {
        return null;
    }

    char[] ch = targetStr.toCharArray();

    for (int i = 0; i < ch.length; i++) {
        switch (ch[i]) {

        // '-'(Double byte hyphen)
        case '\u2015':
            ch[i] = '\u2014';
            break;
        // '－'(Double byte minus)
        case '\uff0d':
            ch[i] = '\u2212';
            break;
        // '～'(Tilde)
        case '\uff5e':
            ch[i] = '\u301c';
            break;
        // '∥'(Pipe)
        case '\u2225':
            ch[i] = '\u2016';
            break;
        // '￠'(Cent symbol)
        case '\uffe0':
            ch[i] = '\u00A2';
            break;
        // '￡'(Pound symbol)
        case '\uffe1':
            ch[i] = '\u00A3';
            break;
        // '￢' (Negation symbol)
        case '\uffe2':
            ch[i] = '\u00AC';
            break;
        default:
            break;
        }
    }

    return String.valueOf(ch);
}
```

Note: Since it is an issue that occurs during mapping in Unicode, conversion is necessary regardless of the character code of input value. Header and body text which may contain strings with Japanese text are used for conversion. From, To, Cc, Bcc, Reply-To, Subject etc are examples of the header that may contain Japanese text and are frequently used in general.

Also, when ISO-2-22-JP is set as encoding, extended characters which are not in scope are garbled.

①	②	③	Ⅰ	Ⅱ	Ⅲ	㍉	㌔	㌢
㎜	㎝	㎞	㍾	㍽	㍼	No.	K.K.	TEL
㊤	㊥	㊦	㈱	㈲	㈹	Σ	∟	⊿

Figure.8.1 **Fig.- Examples of extended characters that are not in scope**

These characters essentially should not be used. If it is necessary to use these characters, settings can be done as follows as JVM start-up options. Moreover, when ISO-2022-JP encoding is set, these characters can be replaced so that they can be mapped with x-windows-iso2022jp.

```
-Dsun.nio.cs.map=x-windows-iso2022jp/ISO-2022-JP
```

Warning: x-windows-iso2022jp is a ISO-2022-JP implementation which includes mapping (MS932 base) that is different from ISO-2022-JP standards. When ISO-2022-JP encoding is specified in the email header, whether the out-of-scope extension characters are handled in the implementation will depend on the email client. Hence it cannot be guaranteed that there will not be any garbling in all email clients even though mapping is done using x-windows-iso2022jp.

When extension characters can also be converted to alternate characters, a method wherein conversion is done in the application independently should also be reviewed similar to seven characters described earlier.

Precautions while using multibyte characters in JavaMail

In JavaMail, body text of mail to be sent ends with multibyte characters, additional characters (”?”, “w” etc) are likely to be output. Following methods can be used to avoid these occurrences.

- Change end character of mail body text to single byte character.
- Change end of mail body text to linefeed code (CRLF)

Email header injection countermeasures

If email header injection attack is successful, an email is sent to an unintended address and thus can end up as "junk mail". When a string that has been entered externally is used in the email header (Subject etc) contents, it becomes necessary to take countermeasures against email header injection attack.

For example, if the following string is set by `setSubject` method of `MimeMessageHelper`, the text can be manipulated by adding a Bcc header.

```
Notification\r\nBcc: attacker@exapmle.com\r\n\r\nManipulated body.
```

Following methods can be considered as countermeasures for email header injection attack.

- Set contents to be specified in email header as a fixed value and output entire string that has been entered externally in the email body text.
- Check that no linefeed character is included in the contents specified in the email header.

Processing method

Since sending an email is a time consuming process, response time can become longer if an email is sent during a web application request. Hence, an email is usually not sent during a web application request and a method is adopted wherein an email is sent asynchronously. Although a process to send the email is not described here in detail, the example is given below which can be used as a reference.

Send an email based on the email information stored in database or message queue

Incorporate the functions given below in the application when you want to send an email based on the email information stored in database or message queue.

- Register the information of the email to be sent (address, body text, attachment etc) in the database (or message queue).
- Fetch the email information that has not been sent from database (or message queue) periodically and send the email through SMTP.
- Register the transmission results in database (or message queue).

Note that, following points must be taken into consideration during review.

- How to check registered email information and email sending results
- Handling in case of an error while sending an email

Tip: When an email is sent continuously by a mailing service, it is determined as spam mail. A method can be considered as a countermeasure wherein the sending sequence is set as random so that emails are not sent continuously for the same domain.

Test using GreenMail

A method wherein GreenMail is used as a fake server to test the email sending function is introduced. GreenMail can also be used by deploying war file besides using it as a library.

Implementation example of test code using GreenMail is shown below.

Configuration example of pom.xml

```
<dependencies>

    <!-- (1) -->
    <dependency>
        <groupId>com.icegreen</groupId>
        <artifactId>greenmail</artifactId>
        <version>1.4.1</version>
        <scope>test</scope>
    </dependency>

</dependencies>
```

Sr.No.	Description
(1)	Add GreenMail library to dependencies.

Implementation example of JUnit source

```
@Inject
EmailService emailService;

@Rule
public final GreenMailRule greenMail = new GreenMailRule(
        ServerSetupTest.SMTP); // (1)
```

```
@Test
public void testSend() {

    String from = "info@example.com";
    String to = "foo@example.com";
    String subject = "Registration confirmation.";
    String text = "Hi "
            + to
            + ", welcome to EXAMPLE.COM!\r\n"
            + "If you were not an intended recipient, Please notify the sender.";
    emailService.send(from, to, subject, text);

    assertTrue(greenMail.waitForIncomingEmail(3000, 1)); // (2)

    Message[] messages = greenMail.getReceivedMessages(); // (3)

    assertNotNull(messages);
    assertEquals(1, messages.length);
    // omitted
}
```

Sr. No.	Description
(1)	Set `GreenMailRule` as a rule which specifies `ServerSetupTest.SMTP`. SMTP port number is set to `3025` as a default.
(2)	Wait for arrival of email by using `waitForIncomingEmail` method. It is used when an email is sent asynchronously by another thread. In this example, it is assumed that the email is sent asynchronously and waiting time for an email arrival is maximum 3 seconds.
(3)	Fetch all received emails by using `getReceivedMessages` method. Emails sent by GreenMail are all received by GreenMail regardless of the address.

8.2 JMS(Java Message Service)

8.2.1 Overview

This chapter explains how to send and receive messages which use components for JMS linking of JMS API and Spring Framework.

What is JMS

JMS is a standard API for using MOM (Message Oriented Middleware) in Java.

JMS architecture exchanges messages from client to client through JMS provider.

Since JMS supports asynchronous messaging, a close coupling can be formed between the clients.

Further, since the message can be stored in Queue by adopting a Point-to-Point model described later, messages can be received in accordance with the client performance.

On the other hand, since the time lag is likely to occur in the messaging between clients, it is not suitable for the processes which require a real time response.

For details of JMS, refer Java Message Service (JMS).

Asynchronous and synchronous messaging can be performed by using JMS.

Note: Use of JMS1.1 is assumed in this guideline.

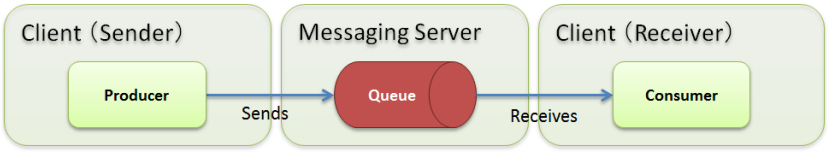
While using, the delivery method and the method to send and receive messages described below must be selected in accordance with the requirements.

- **Delivery model**

  Delivery model consists of two models - Point-to-Point (PTP) and Publisher-Subscriber (Pub/Sub).

A major difference between two models is in whether the sender and recipient ratio is 1:1 or 1:many, and should be selected based on the use.

1. Point-To-Point (PTP) model



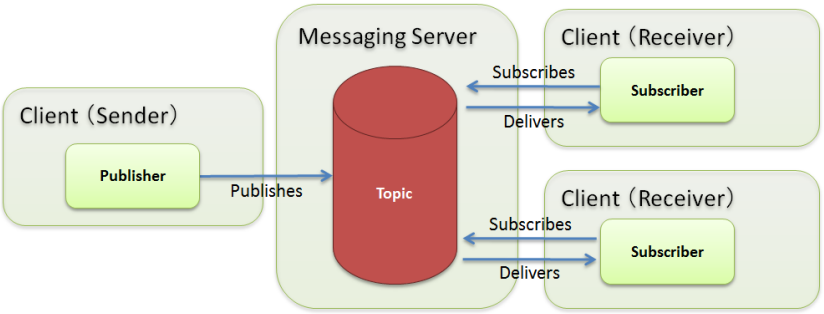
PTP model consists of 2 clients wherein one of the clients (Producer) sends a message and the other client (Consumer) alone receives that message.

Destination of the message in PTP model is called as a Queue.

Producer sends the message to the Queue and the Consumer fetches the message from the Queue, and the process is carried out.

Message is fetched from the consumer or when the message reaches expiry period, the message is deleted from the Queue.

2. Publisher-Subscriber (Pub/Sub) model



Pub/Sub model consists of 2 clients wherein one of the clients (Publisher) issues (Publishes) the message and delivers that message to multiple other clients (Subscribers).

Destination of the message in Pub/Sub model is called as a Topic.

Subscriber raises a subscription request for the Topic and the Publisher issues a message in Topic.

The message is delivered to all the subscribers who have raised a request for subscription.

This guideline explains how to implement PTP model which is widely used in general.

- **A method to send messages**

There are two types of processes for message sending - Synchronous sending method and asynchronous sending method for sending the messages to Queue or Topic however JMS1.1 supports only synchronous sending method.

1. Synchronous sending method

   Message is processed and sent by explicitly calling a function to send messages.

   Subsequent processing is blocked in order to wait until the response is received from JMS provider.

2. Asynchronous sending method

   Message is processed and sent by explicitly calling a function to send messages.

   Subsequent processing is continued since it is not necessary to wait for the response from JMS provider.

   For the details of asynchronous sending method, refer Java Message Service(Version 2.0)“7.3. Asynchronous send”.

- **Message receiving method**

  There are two types of methods for receiving messages - synchronous receiving method and asynchronous receiving method while implementing the process for receiving messages in Queue or Topic.

  As described later, since use cases of synchronous receiving methods are limited, generally asynchronous receiving methods are widely used.

  1. Asynchronous receiving method

     When the message is received by Queue or Topic, the processing for the received message is initiated.

     Since the processing for one message is initiated without terminating the processing for the other message, it is suitable for parallel processing.

  2. Synchronous receiving method

     Receiving the message and related processing is initiated by explicitly calling the function which receives the message.

     The function which receives the message waits till the message is received when the message does not exist in Queue or Topic.

     Hence, the waiting period for the message must be specified by configuring the timeout value.

     As an example of synchronous receiving of message, it can be used when the message accumulated in the Queue, in the Web application is to be fetched and processed at any time like screen operation etc or when the messages are to be processed periodically in a batch.

In JMS, the message consists of following parts.

For details, refer "3. JMS Message Model" of Java Message Service(Version 1.1).

Structure	Description
Header	Control information of message like destination and identifier, extension header (JMSX) of JMS, JMS provider specific header and application specific header are stored for JMS provider and application.
Property	Control information to be added to header is stored.
Payload	Message body is stored. As data types, 5 types of message types namely `javax.jms.BytesMessage`, `javax.jms.MapMessage`, `javax.jms.ObjectMessage`, `javax.jms.StreamMessage`and `javax.jms.TextMessage`are offered. `ObjectMessage` is used while sending a JavaBean. In such a case, JavaBean must be shared between the clients.

Using JMS

When the process is to be implemented by using JMS, it can be executed by using JMS API defined in Java EE (hereafter referred to as JMS API).

However, this guideline assumes the use of JMS linking component of Spring Framework wherein the merits are significant as compared to using JMS API as it is (easy description etc).

Hence, details of JMS API are not described.

For details, refer Java API.

Note: For JMS, Java API is standardized however physical protocols of messages are not standardized.

Note: Since JMS implementation is incorporated in Java EE server as a standard, it can be used as a default (restricted while using JMS provider incorporated in Java EE server), however JMS must be separately implemented in Java EE server like Apache Tomcat wherein JMS is not incorporated.

Using JMS which use component of Spring Framework

Following are provided as the libraries for sending and receiving messages in Spring Framework.

- `spring-jms`

  It offers a component for the messaging which use JMS.

  By using the components included in this library, low level JMS API calling becomes unnecessary and the implementation becomes simplified.

  `spring-messaging` can be used.

- `spring-messaging`

  It offers a component to abstract the infrastructure function which is required for creating application of messaging base.

  It consists of a set of annotations to associate with the message and the method for processing the same.

  By using the components included in the library, implementation style of messaging can be matched.

Although implementation can be done only in `spring-jms`, implementation method can be combined by using `spring-messaging`.
This guideline recommends the use of `spring-messaging` as well.

Here, prior to explaining basic implementation method, it is explained how the messages are sent and received by the components for JMS linking offered by Spring Framework.
At first, the components registered in the description are introduced.
Spring Framework sends and receives messages through JMS API using interfaces and classes shown below.

- `javax.jms.ConnectionFactory`

  An interface for creating a connection with JMS provider.

  It offers a function to create a connection to JMS provider from the application.

- `javax.jms.Destination`

  An interface for indicating the address (Queue or topic).

- `javax.jms.MessageProducer`

  An interface for sending messages.

- `javax.jms.MessageConsumer`

  An interface for receiving messages.

- `javax.jms.Message`

  An interface which shows the message that retains header and body.

  Messages are sent and received by implementation class of the interface.

- `org.springframework.messaging.Message`

  An interface which abstracts messages handled by various messaging systems.

  It can also be used in JMS.

  As mentioned earlier, basically `org.springframework.messaging.Message` offered by spring-messaging is used to comply with implementation method of messaging.

  However, `javax.jms.Message` is used when it is preferable to use `org.springframework.jms.core.JmsTemplate`.

- `org.springframework.jms.core.JmsMessagingTemplate` and `org.springframework.jms.core.Jm`

  A class which creates templates for generation and release of resources for using JMS API.

  The implementation can be simplified by using a message sending and message synchronous receiving function.

  Basically, `JmsMessagingTemplate` which can handle `org.springframework.messaging.Message` is used.

  Since `JmsMessagingTemplate` wraps `JmsTemplate`, configuration can be done by using `JmsTemplate` property.

  However, sometimes it is preferable to use a `JmsTemplate` as it is. Specific use cases are explained later.

- `org.springframework.jms.listener.DefaultMessageListenerContainer`

  `DefaultMessageListenerContainer` receives messages from Queue and initiates `MessageListener` which processes the received messages.

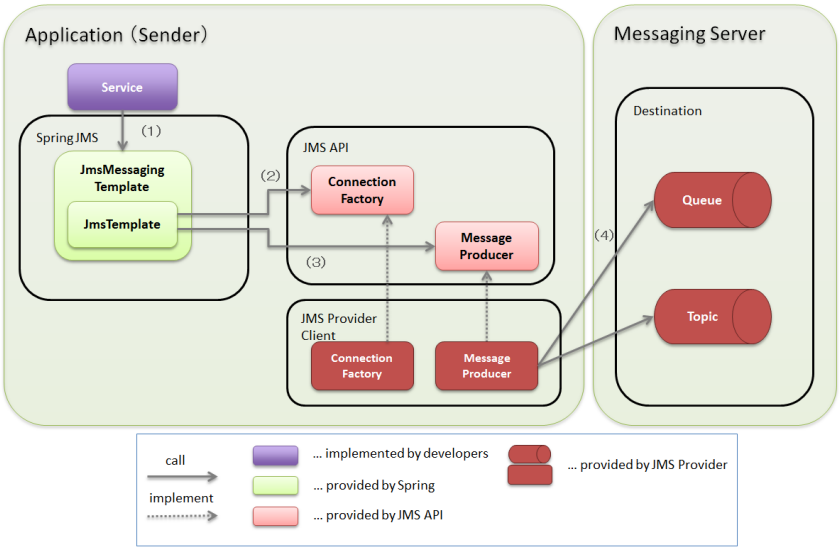
- `@org.springframework.jms.annotation.JmsListener`

  A marker annotation which indicates that it is a method to be handled as `MessageListener` of JMS.

  A `@JmsListener` annotation is assigned for a method which performs a process while receiving messages.

When messages are sent synchronously

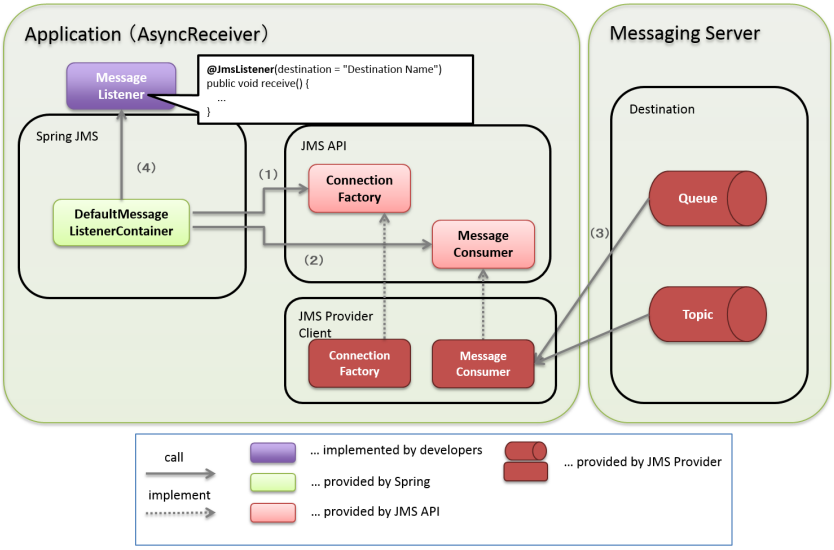
The flow of the process to send messages synchronously is explained using the figure.



Sr.No.	Description
(1)	Execute the process in Service by passing “Destination name for sending” and “payload of messages to be sent” for `JmsMessagingTemplate`. `JmsMessagingTemplate` delegates the process to `JmsTemplate`.
(2)	`JmsTemplate`fetches `javax.jms.Connection` from `ConnectionFactory` fetched through JNDI.
(3)	`JmsTemplate` passes `Destination` and messages to `MessageProducer`. `MessageProducer` is generated from `javax.jms.Session`. (`Session` is generated from `Connection` fetched in (2).) Further, `Destination` is fetched through JNDI based on “Destination name for sending” passed in (1).
(4)	`MessageProducer` sends messages to `Destination` for sending.

When messages are received asynchronously

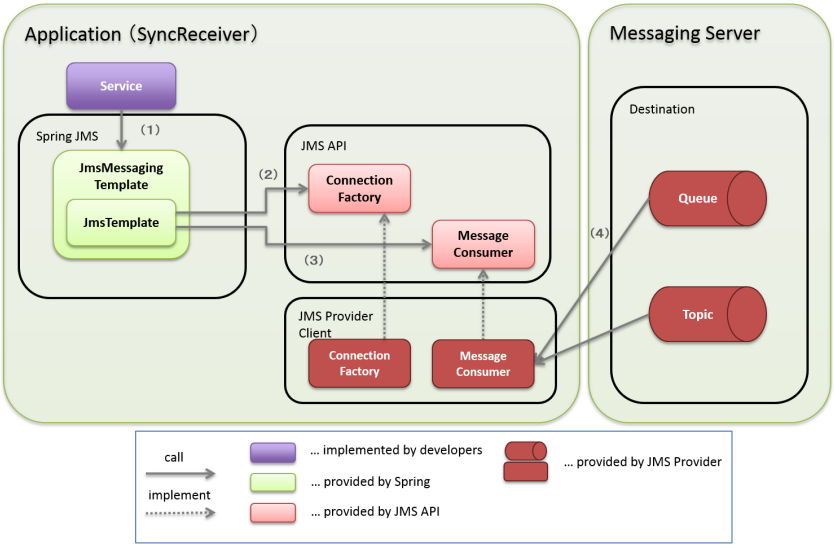
The flow of the process to receive messages asynchronously is explained using a figure.



Sr. No.	Description
(1)	Fetch `Connection` from `ConnectionFactory` fetched through JNDI.
(2)	`DefaultMessageListenerContainer` passes `Destination` to `MessageConsumer`. `MessageConsumer` is generated from `Session`. (`Session` is generated from `Connection` fetched in (1)) Further, `Destination` is fetched through JNDI based on “Destination name for receiving” specified in `@JmsListener` annotation.
(3)	`MessageConsumer` receives messages from `Destination`.
(4)	A method (listener method) configured by `@JmsListener` annotation in `MessageListener` is called using received message as an argument. Listener method is managed by `DefaultMessageListenerContainer`.

When the messages are received synchronously

The flow of the process to receive messages synchronously is explained using a figure.



Sr. No.	Description
(1)	Pass “Destination name for receiving” in Service for `JmsMessagingTemplate`. `JmsMessagingTemplate` delegates the process to `JmsTemplate`.
(2)	`JmsTemplate` fetches `Connection` from `ConnectionFactory` fetched through JNDI.
(3)	`JmsTemplate` passes `Destination` and messages in `MessageConsumer`. `MessageConsumer` is generated from `Session`. (`Session` is generated from `Connection` fetched in (2).) Further, `Destination` is fetched through JNDI based on “Destination name for receiving” passed in (1).
(4)	`MessageConsumer` receives the messages from `Destination`. Message is returned to the Service through `JmsTemplate` or `JmsMessagingTemplate`.

Regarding project configuration

Recommended configuration of the project while using JMS is explained.

When serialised JavaBean is sent and received through `ObjectMessage`, the JavaBean must be shared by sending side and receiving side.

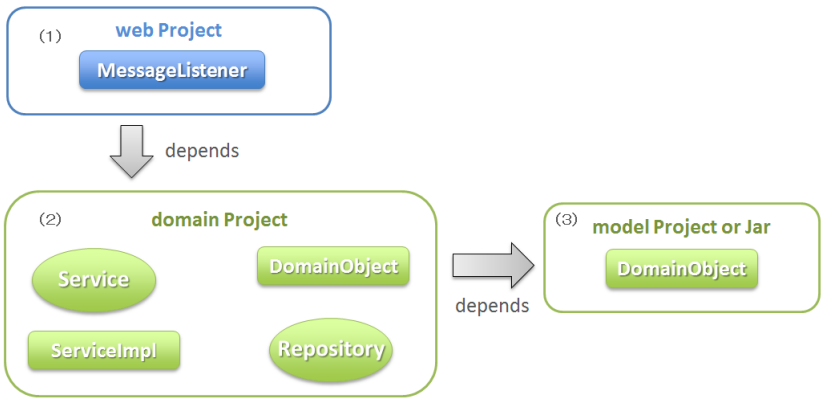
In such a case, it is recommended to add a model project different from the existing blank project.

- **Sharing model**
  - When client of sending and receiving side does not provide a model

    model project is added and Jar file is distributed in the client for communication.
  - When client of sending and receiving side offer a model

    The model provided is added to library.

model project, and, relation between distributed archive file and existing project are as below.



Sr. No.	Project name	Description
(1)	web project	Place the listener class to receive messages asynchronously.
(2)	domain project	Place the Service executed from listener class for receiving messages asynchronously. Besides, Repository etc are same as used conventionally.
(3)	model project or Jar file	A class which is shared between the clients is used among the classes belonging to domain layer.

Following are implemented to add a model project.

- Creating a model project
- Adding a dependency from domain project to model project

For detail addition method, refer *Project configuration is changed for SOAP server.* of *SOAP Web Service (Server/Client)*. shared by JavaBean in the same way.

8.2.2 How to use

Settings that are common for both sending and receiving messages

This section explains common settings required for sending and receiving messages.

Configuration of dependent library

`spring-jms` of Spring Framework is added to pom.xml of domain project in order to use component for linking JMS of Spring Framework.

- `[projectName]-domain/pom.xml`

```
<dependencies>

    <!-- (1) -->
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-jms</artifactId>
    </dependency>

</dependencies>
```

Sr. No.	Description
(1)	Add dependencies to `spring-jms`. Since `spring-jms` is dependent on `spring-messaging`, `spring-messaging` is also added as a transitive dependent library.

A library of JMS provider is added to pom.xml in addition to `spring-jms`.

For additional examples of libraries for pom.xml, refer *JMS provider dependent configuration*.

Note: In the above setting example, since it is assumed that the dependent library version is managed by the parent project terasoluna-gfw-parent , specifying the version in pom.xml is not necessary. The above dependent library used by terasoluna-gfw-parent is defined by Spring IO Platform.

Configuration of `ConnectionFactory`

`ConnectionFactory` definition method consists of two methods - a method defined by application server and a method defined by Bean definition file.

A method defined by application server is selected to make Bean definition file independent of JMS provider unless there is a specific reason to do otherwise.

This chapter explains a method defined by application server only.

Configuration for using a JavaBean fetched in Bean definition file through JNDI must be performed in order to use `ConnectionFactory` defined in the application server.

- [projectName]-domain/src/main/resources/META-INF/spring/[projectName]-infra.xml

```
<!-- (1) -->
<jee:jndi-lookup id="connectionFactory" jndi-name="jms/ConnectionFactory"/>
```

Sr. No.	Description
(1)	Specify JNDI name of ConnectionFactory offered by application server, in jndi-name attribute. Since resource-ref attribute is set to true by default, it is auto-assigned when no prefix exists for JNDI name (java:comp/env/).

Note: When ConnectionFactory for which a Bean is defined is used

When JNDI is not used, ConnectionFactory can be used even when a Bean is defined for implementation class of ConnectionFactory. In such a case, implementation class of ConnectionFactory is dependent on JMS provider. For details, refer *JMS provider dependent configuration* "configuration when JNDI is not used".

Configuration of DestinationResolver

Name resolution of Destination consists of 2 methods - Resolution by JNDI and Resolution by JMS provider.

Resolution by JMS provider is performed by default, however resolution by JNDI is recommended for portability and control if there are no particular issues to address.

Name resolution of Destination can be performed by JNDI name, by using org.springframework.jms.support.destination.JndiDestinationResolver.

How to define JndiDestinationResolver is shown below.

- [projectName]-domain/src/main/resources/META-INF/spring/[projectName]-infra.xml

```
<!-- (1) -->
<bean id="destinationResolver"
   class="org.springframework.jms.support.destination.JndiDestinationResolver
↪">
   <property name="resourceRef" value="true" /> <!-- (2) -->
</bean>
```

Sr. No.	Description
(1)	Define a Bean for `JndiDestinationResolver`.
(2)	When there is no prefix in JNDI name (java:comp/env/), set `true`when it is auto-assigned. Default value is `false`. **Warning:** Note that `resource-ref` attribute of `<jee:jndi-lookup/>` is different from the default value.

A method to synchronously send messages

A method to synchronously send messages from client (Producer) to JMS provider, for PTP model is explained below.

Basic synchronous sending

Process of synchronous sending to JMS provider is achieved by using `JmsMessagingTemplate`.

An implementation example wherein object of `Todo` class is synchronously sent as a message.

At first, how to configure `JmsMessagingTemplate` is shown below.

- `[projectName]-domain/src/main/resources/META-INF/spring/[projectName]-infra.xml`

```
<bean id="cachingConnectionFactory"
   class="org.springframework.jms.connection.CachingConnectionFactory"> <!--␣
↪(1) -->
   <property name="targetConnectionFactory" ref="connectionFactory" /> <!--␣
↪(2) -->
   <property name="sessionCacheSize" value="1" />  <!-- (3) -->
</bean>

<!-- (4) -->
<bean id="jmsTemplate" class="org.springframework.jms.core.JmsTemplate">
   <property name="connectionFactory" ref="cachingConnectionFactory" />
   <property name="destinationResolver" ref="destinationResolver" />
</bean>
```

```
<!-- (5) -->
<bean id="jmsMessagingTemplate" class="org.springframework.jms.core.
↪JmsMessagingTemplate">
    <property name="jmsTemplate" ref="jmsTemplate"/>
</bean>
```

Sr. No.	Description
(1)	Define a Bean for `org.springframework.jms.connection.CachingConnectionFactory` which performs caching of `Session` and `MessageProducer/Consumer`. JMS provider specific `ConnectionFactory` looked up by Bean definition or JNDI name is not used as it is. Cache function can be used by wrapping it in `CachingConnectionFactory`.
(2)	Specify JMS provider specific `ConnectionFactory` which is looked up by Bean definition or JNDI name.
(3)	Specify cache count for `Session`. (Default value is 1) Although 1 is specified in this example, number of caches must be changed appropriately corresponding to performance requirements. When the session is required to be continued even after exceeding cache count, a new session is created and destroyed repeatedly without using a cache. This is likely to cause deterioration of process efficiency resulting in performance degradation.
(4)	Define a Bean for `JmsTemplate`. `JmsTemplate` alternates for a low level API handling (JMS API calling). For attributes that can be configured, refer list of attributes of `JmsTemplate` given below.
(5)	Define a Bean for `JmsMessagingTemplate`. Inject `JmsTemplate` which alternates as a synchronous sending process.

Attributes of `JmsTemplate` for synchronous sending are as below.

It must be configured as per requirement.

Sr. No.	Configuration item	Details	Mandatory	Default value
1.	`connectionFactory`	Configure `ConnectionFactory` to be used.	○	Nil (since it is mandatory)
2.	`pubSubDomain`	Configure for a messaging model. Set `false` for PTP (Queue) model and set `true` for Pub/Sub (Topic).	-	`false`
3.	`sessionTransacted`	Specify whether the transaction control is performed in the session. In this guideline, `false` is recommended as a default for using transaction control described later.	-	`false`
4.	`messageConverter`	Specify message converter. Default can be used for the scope described in the guideline.	-	`SimpleMessageCo` is used.

次のページに続く

Table. 8.5 – 前のページからの続き

Sr. No.	Configuration item	Details	Mandatory	Default value
5.	`destinationResolver`	Specify DestinationResolver. In this guideline, it is recommended to use `JndiDestinationResolver` which performs name resolution in JNDI.	-	`DynamicDestinat` is used. (When `DynamicDestinat` is used, name resolution of Destination is performed by JMS provider.)
6.	`defaultDestination`	Specify existing Destination. When Destination is not specified explicitly, this Destination is used.	-	null (no existing destination)
7.	`deliveryMode`	Select 1 (NON_PERSISTENT) or 2(PERSISTENT) for delivery mode. 2(PERSISTENT) performs perpetuation of messages. 1(NON_PERSISTENT) does not perform perpetuation of messages. Hence, even though performance shows improvement, messages are likely to get lost if JMS provider is restarted. In this guideline, it is recommended to use 2 (PERSISTENT) for avoiding loss of messages. When this configuration is used, note that `explicitQosEnabled` described later must be set to `true`.	-	2(PERSISTENT)

次のページに続く

Table. 8.5 – 前のページからの続き

Sr. No.	Configuration item	Details	Mandatory	Default value
8.	`priority`	Set priority for messages. Priority can be set from 0 to 9. Higher the number, higher is the priority. Priority is evaluated when the message is stored in the Queue at the time of synchronous sending. Messages are stored in such a way that messages with high priority are extracted before the messages with low priority. FIFO (First-In-First-Out) system is employed for the messages with identical priorities. When this configuration is used, note that `explicitQosEnabled` described later must be set to `true`.	-	4
9.	`timeToLive`	Specify validity period of messages in milliseconds. When the message reaches the validity period, JMS provider deletes the message from Queue When this configuration is used, note that `explicitQosEnabled` described later must be set to `true`.	-	0 (Unrestricted)
10.	`explicitQosEnabled`	Specify `true` when `deliveryMode`, `priority` and `timeToLive` are enabled.	-	`false`

(*1)`org.springframework.jms.support.converter.SimpleMessageConverter`

(*2)`org.springframework.jms.support.destination.DynamicDestinationResolver`

Next, a JavaBean is created for sending.

- `[projectName]-domain/src/main/java/com/example/domain/model/Todo.java`

```
package com.example.domain.model;

import java.io.Serializable;

public class Todo implements Serializable { // (1)

    private static final long serialVersionUID = -1L;

    // omitted

    private String description;

    // omitted

    private boolean finished;

    // omitted

    public String getDescription() {
        return description;
    }

    public void setDescription(String description) {
        this.description = description;
    }

    public boolean isFinished() {
        return finished;
    }

    public void setFinished(boolean finished) {
        this.finished = finished;
    }

}
```

Sr. No.	Description
(1)	Basically, a normal JavaBean can be used, however, a `java.io.Serializable` interface must be implemented for a serialised transmission.

A process to perform actual synchronous sending is described in the end.

An implementation example is shown below wherein `Todo` object consisting of specified text is synchronously sent to Queue.

- `[projectName]-domain/src/main/java/com/example/domain/service/todo/TodoServiceImpl.`

```
package com.example.domain.service.todo;

import javax.inject.Inject;
import org.springframework.jms.core.JmsMessagingTemplate;
import org.springframework.stereotype.Service;
import com.example.domain.model.Todo;

@Service
public class TodoServiceImpl implements TodoService {

    @Inject
    JmsMessagingTemplate jmsMessagingTemplate;    // (1)

    @Override
    public void sendMessage(String message) {

        Todo todo = new Todo();
        // omitted

        jmsMessagingTemplate.convertAndSend("jms/queue/TodoMessageQueue",␣
↪todo);  // (2)

    }
}
```

Sr. No.	Description
(1)	Inject `JmsMessagingTemplate`.
(2)	By using `convertAndSend` method of `JmsMessagingTemplate`, JavaBean of argument is converted to implementation class of `org.springframework.messaging.Message` interface and the message is synchronously sent for a specified Destination. `org.springframework.jms.support.converter.SimpleMessageConverter` is used in the default conversion. When `SimpleMessageConverter` is used, the class which implements `javax.jms.Message`, `java.lang.String`, `byte array`, `java.util.Map` and `java.io.Serializable` interface can be sent.

Note: Exception handling of JMS in business logic

As covered in JMS (Java Message Service) introduction, exceptions are converted to run-time exceptions in Spring Framework. Hence, a run-time exception must be handled while handling JMS exception in business logic.

Template class	A method to convert exceptions	Exception after conversion
`JmsMessagingTemplate`	`convertJmsException` method of `JmsMessagingTemplate`	`MessagingException`(*1) and its sub-exception
`JmsTemplate`	`convertJmsAccessException` method of `JmsAccessor`	`JmsException`(*2) and its sub-exception

(*1) `org.springframework.messaging.MessagingException`

(*2) `org.springframework.jms.JmsException`

When message header is to be edited and sent synchronously

Header attribute can be edited and then sent synchronously by specifying header attribute and value of Key-Value format in `convertAndSend` method argument of `JmsMessagingTemplate`. For header details, refer javax.jms.Messages. An implementation example wherein `JMSCorrelationID` of role which links sending and response messages is specified at the time of synchronous sending.

- `[projectName]-domain/src/main/java/com/example/domain/service/todo/TodoServiceImpl.`

```
package com.example.domain.service.todo;

import java.util.Map;
import javax.inject.Inject;
import org.springframework.jms.core.JmsMessagingTemplate;
import org.springframework.stereotype.Service;
import org.springframework.jms.support.JmsHeaders;
import com.example.domain.model.Todo;

@Service
public class TodoServiceImpl implements TodoService {

@Inject
JmsMessagingTemplate jmsMessagingTemplate;

  public void sendMessageWithCorrelationId(String correlationId) {

    Todo todo = new Todo();
    // omitted

    Map<String, Object> headers = new HashMap<>();
    headers.put(JmsHeaders.CORRELATION_ID, correlationId);// (1)

    jmsMessagingTemplate.convertAndSend("jms/queue/TodoMessageQueue",
            todo, headers); // (2)

  }
}
```

Sr. No.	Description
(1)	Create header information by configuring header attribute name and its value, for implementation class of `Map`.
(2)	Synchronously send the message assigned with header information created in (2) by using `convertAndSend` method of `JmsMessagingTemplate`.

Warning: Regarding header attributes that can be edited

Components of header attributes (`JMSDestination`, `JMSDeliveryMode`, `JMSExpiration`, `JMSMessageID`, `JMSPriority`, `JMSRedelivered`and `JMSTimestamp`) are handled as read-only while converting messages using `SimpleMessageConverter` of Spring Framework. Hence, even though read-only header attributes are configured as shown in the implementation example, they are not stored in the header of sent messages. (retained as message properties.) Among read-only header attributes, `JMSDeliveryMode` and `JMSPriority` can be configured in `JmsTemplate` unit. For details, refer list of attributes of `JmsTemplate` of *Basic synchronous sending*.

Transaction control

When data consistency is required to be guaranteed, a transaction control function is used.

An implementation example wherein “Declaration type transaction control” is used is recommended in this guideline.

For details of “Declaration type transaction control”, refer *Regarding transaction management* .

`org.springframework.jms.connection.JmsTransactionManager` is used to achieve transaction control.

Configuration example is shown at first.

- `[projectName]-domain/src/main/resources/META-INF/spring/[projectName]-domain.xml`

```
<!-- (1) -->
<bean id="sendJmsTransactionManager"
   class="org.springframework.jms.connection.JmsTransactionManager">
   <!-- (2) -->
   <property name="connectionFactory" ref="cachingConnectionFactory" />
</bean>
```

Sr. No.	Description
(1)	Define a Bean for `JmsTransactionManager`. **Note: Regarding bean name of TransactionManager** When `@Transactional` annotation is assigned, a Bean registered by Bean name `transactionManager` is used as a default. (For details, refer *Settings for using transaction management*) In Blank project, since `DataSourceTransactionManager` is defined by Bean name called `transactionManager`, Bean is defined with an alias in the configuration above. Hence, when only one `TransactionManager` is used in the application, specification of `transactionManager` attribute in `@Transactional` annotation can be omitted by using `transactionManager` as a Bean name.
(2)	Specify `CachingConnectionFactory` which controls a transaction.

An implementation example is shown below wherein `Todo` object is synchronously sent to Queue by performing transaction control.

- `[projectName]-domain/src/main/java/com/example/domain/service/todo/TodoServiceImpl.`

```
package com.example.domain.service.todo;

import javax.inject.Inject;
import org.springframework.jms.core.JmsMessagingTemplate;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import com.example.domain.model.Todo;

@Service
@Transactional("sendJmsTransactionManager")  // (1)
public class TodoServiceImpl implements TodoService {
    @Inject
    JmsMessagingTemplate jmsMessagingTemplate;

    @Override
    public void sendMessage(String message) {

        Todo todo = new Todo();
        // omitted

        jmsMessagingTemplate.convertAndSend("jms/queue/TodoMessageQueue", todo);
    // (2)
```

```
    }

}
```

Sr. No.	Description
(1)	Declare transaction boundary by using `@Transactional` annotation. Accordingly, transaction starts while starting each method in the class and transaction is committed while terminating a method.
(2)	Send message synchronously to Queue. However, note that the message is actually sent to Queue within the timing wherein the transaction is committed.

In the application wherein DB transaction control is performed, a transaction control policy must be determined after reviewing relation between JMS and DB transaction based on business requirements.

- **When JMS and DB transactions are to be separated and then committed and rolled back**

  A transaction boundary is declared individually when JMS and DB transactions are to be separated.

  An implementation example is shown below wherein `sendJmsTransactionManager` of *Transaction control* is used in JMS transaction control whereas `transactionManager` defined in default configuration of Blank project is used in DB transaction control.

- `[projectName]-domain/src/main/java/com/example/domain/service/todo/TransactionalTod`

```
package com.example.domain.service.todo;

import javax.inject.Inject;
import org.springframework.jms.core.JmsMessagingTemplate;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import com.example.domain.model.Todo;

@Service
@Transactional("sendJmsTransactionManager")  // (1)
```

```
public class TransactionalTodoServiceImpl implements TransactionalTodoService
↪{
    @Inject
    JmsMessagingTemplate jmsMessagingTemplate;

    @Inject
    TodoService todoService;

    @Override
    public void sendMessage(String message) {
        Todo todo = new Todo();
        // omitted

        jmsMessagingTemplate.convertAndSend("jms/queue/TodoMessageQueue",␣
↪todo);

        // omitted
        todoService.update(todo);
    }

}
```

- src/main/java/com/example/domain/service/todo/TodoServiceImpl.java

```
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import com.example.domain.model.Todo;

@Transactional // (2)
@Service
public class TodoServiceImpl implements TodoService {

    @Override
    public void update(Todo todo) {
        // omitted
    }
}
```

Sr. No.	Description
(1)	Declare transaction boundary of JMS by using `@Transactional` annotation. Specify `sendJmsTransactionManager` which performs transaction control of JMS.
(2)	Declare transaction boundary of DB by using `@Transactional` annotation. Since value is omitted, Bean name `transactionManager` is referred as a default. For details of `@Transactional` annotation, refer *Regarding transaction management* of *Domain Layer Implementation*.

- **When JMS and DB transactions are to committed and rolled back together**

  A method which uses global transaction by JTA exists for linking JMS and DB transactions, however "Best Effort 1 Phase Commit" is recommended since overheads are likely to occur for performance, among protocol characteristics. Refer below for details.

  Distributed transactions in Spring, with and without XA

  Spring Distributed transactions using Best Effort 1 Phase Commit

  Warning: When transaction process results are not returned in JMS provider due to issues like loss of connection with JMS provider after receiving a message

  When transaction process results are not returned in JMS provider due to issues like loss of connection with JMS provider after receiving a message, transaction handling depends on JMS provider. Hence, **a design considering loss of received messages** must be carried out. Especially, when loss of messages is absolutely not permissible, **providing a system to compensate for the loss of messages or using a global transaction etc** must be adopted.

  "Best Effort 1 Phase Commit" can be achieved by using `org.springframework.data.transaction.ChainedTransactionManager`.

  An implementation example is shown below wherein `sendJmsTransactionManager` of *Transaction*

control is used in JMS transaction management whereas `transactionManager` defined in default configuration of blank project is used in DB transaction management.

- xxx-env.xml

```
<!-- (1) -->
<bean id="sendChainedTransactionManager" class="org.springframework.data.
↪transaction.ChainedTransactionManager">
    <constructor-arg>
        <list>
            <!-- (2) -->
            <ref bean="sendJmsTransactionManager" />
            <ref bean="transactionManager" />
        </list>
    </constructor-arg>
</bean>
```

Sr. No.	Description
(1)	Define a Bean for `ChainedTransactionManager`.
(2)	Specify JMS and DB transaction manager. Transaction starts in a registered sequence and transaction is committed in the reverse sequence.

Implementation example using the configuration given above is shown below.

- `[projectName]-domain/src/main/java/com/example/domain/service/todo/ChainedTransacti`

```
package com.example.domain.service.todo;

import javax.inject.Inject;
import org.springframework.jms.core.JmsMessagingTemplate;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import com.example.domain.model.Todo;

@Service
@Transactional("sendChainedTransactionManager")  // (1)
public class ChainedTransactionalTodoServiceImpl implements␣
↪ChainedTransactionalTodoService {
  @Inject
  JmsMessagingTemplate jmsMessagingTemplate;

  @Inject
```

```
    TodoSharedService todoSharedService;

    @Override
    public void sendMessage(String message) {
        Todo todo = new Todo();
        // omitted

        jmsMessagingTemplate.convertAndSend("jms/queue/TodoMessageQueue",␣
↪todo); // (2)

        // omitted
        todoSharedService.insert(todo); // (3)
    }

}
```

Sr. No.	Description
(1)	Control JMS and DB transactions by specifying `sendChainedTransactionManager` in `@Transactional` annotation. For details of `@Transactional` annotation, refer *Regarding transaction management* of *Domain Layer Implementation*.
(2)	Send messages synchronously.
(3)	Execute process associated with DB access. In this example, SharedService is executed along with DB update.

Note: If it is necessary to manage multiple transactions together like JMS and DB, for business, a global transaction is considered. For global transactions, refer "when transaction control (global transaction control) is necessary for multiple DB (multiple resources)" of *Settings for using transaction management*.

A method to receive messages asynchronously

As described in "message receiving method" of *What is JMS*, asynchronous receiving is generally used for receiving messages.

Asynchronous receiving can be achieved by registering a listener method assigned by `@JmsListener` annotation for `DefaultMessageListenerContainer` responsible for asynchronous receiving function

Roles of listener method which perform s processing at the time of asynchronous receiving are shown below.

1. **Provide a method to receive messages.**

   Messages can be received by executing a method assigned by `@JmsListener` annotation.

2. **Call business process.**

   Business process is not implemented by listener method. It is delegated to Service method.

3. **Handle exceptions occurring in business logic.**

   Business exceptions and library exceptions occurred during normal operations are handled.

4. **Send process results as messages.**

   In the methods wherein response messages are required to be sent, process results of listener method and business logic should be sent as messages for a specified Destination, by using `org.springframework.jms.listener.adapter.JmsResponse`.

Basic asynchronous receiving

An asynchronous receiving method using `@JmsListener` annotation is explained.

Following configuration is required for the implementation of asynchronous receiving.

- Define JMS Namespace.
- Enable `@JmsListener` annotation.
- Specify `@JmsListener` annotation in the method of component managed by DI container.

Respective detail implementation methods are described below.

- `[projectName]-web/src/main/resources/META-INF/spring/applicationContext.xml`

```
<!-- (1) -->
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:jms="http://www.springframework.org/schema/jms"
    xsi:schemaLocation="http://www.springframework.org/schema/beans http://
↪www.springframework.org/schema/beans/spring-beans.xsd
```

```
        http://www.springframework.org/schema/jms http://www.springframework.
↪org/schema/jms/spring-jms.xsd">

    <!-- (2) -->
    <jms:annotation-driven />


    <!-- (3) -->
    <jms:listener-container
        factory-id="jmsListenerContainerFactory"
        destination-resolver="destinationResolver"
        concurrency="1"
        cache="consumer"
        transaction-manager="jmsAsyncReceiveTransactionManager"/>
```

Sr. No.	Attribute name	Details
(1)	xmlns:jms	Define JMS Namespace. Specify `http://www.springframework.org/schema/jms` as a value. For details of JMS Namespace, refer JMS Namespace Support.
	xsi:schemaLocation	Specify URL of schema. Add `http://www.springframework.org/schema/jms` and `http://www.springframework.org/schema/jms/spring-jms.xsd` to values.
(2)	-	Enable JMS related annotation function of `@JmsListener` annotation and `@SendTo` annotation by using `<jms:annotation-driven />`.

次のページに続く

Table. 8.6 – 前のページからの続き

Sr. No.	Attribute name	Details
(3)	-	Configure `DefaultMessageListenerContainer` by assigning parameters to the factory which generate `DefaultMessageListenerContainer`, by using `<jms:listener-container/>`. `connection-factory` attribute which can specify a Bean of `ConnectionFactory` to be used, exists in the `<jms:listener-container/>`attribute. Default value of `connection-factory` attribute is `connectionFactory`. In this example, since Bean (Bean name is `connectionFactory`) of `ConnectionFactory` shown in *Configuration of ConnectionFactory* is used, `connection-factory` attribute is omitted. `<jms:listener-container/>` also consists of attributes other than introduced here. For details, refer Attributes of the JMS <listener-container> element. **Warning:** Since `DefaultMessageListenerContainer` is equipped with an independent cache function, `CachingConnectionFactory` should not be used in case of asynchronous receiving. For details, refer Javadoc of DefaultMessageListenerContainer. For above, `ConnectionFactory` defined in *Configuration of ConnectionFactory* should be specified in `connection-factory` attribute of `<jms:listener-container/>`.

次のページに続く

Table. 8.6 – 前のページからの続き

Sr. No.	Attribute name	Details
	concurrency	Specify upper limit of parallel number of each listener method managed by DefaultMessageListenerContainer. Default value for concurrency attribute is 1. Lower limit and upper limit of parallel numbers can also be specified. For example, specify "5-10" when lower limit is 5 and upper limit is 10. When the parallel number of listener method has reached specified upper limit, parallel processing is not done and a "waiting" state is reached. A value should be specified as required. **Note:** When you want to specify parallel number in listener method unit, concurrency attribute of @JmsListener annotation can be used.
	destination-resolver	Specify Bean name of DestinationResolver which is used to resolve Destination name at the time of asynchronous receiving. For Bean definition of DestinationResolver, refer *Configuration of DestinationResolver*. When destination-resolver attribute is not specified, DynamicDestinationResolver generated in DefaultMessageListenerContainer is used.
	factory-id	Specify name of DefaultJmsListenerContainerFactory which defines a Bean. Since @JmsListener annotation refers Bean name jmsListenerContainerFactory as a default, it is recommended to consider Bean name as jmsListenerContainerFactory in case of a single <jms:listener-container/>.

次のページに続く

Table. 8.6 – 前のページからの続き

Sr. No.	Attribute name	Details
	`cache`	Specify cache level to determine cache targets like `Connection`, `Session` or `Consumer` etc. Default is `auto`. When Connection is not pooled in the application server while configuring `transaction-manager` attribute described later, specifying `consumer` is recommended for performance improvement. **Note:** In case of `auto`, when `transaction-manager` attribute is not configured, behaviour is same as `consumer`(`Consumer` is cached). However, behaviour is same as `none`(invalid cache) considering the pooling in the application server using global transaction at the time of `transaction-manager` attribute configuration.
	`transaction-manager`	Specify Bean name which performs transaction control at the time of asynchronous receiving. For details, refer *Transaction control*.

Messages can be asynchronously received from specified Destination by specifying `@JmsListener` annotation in the component method managed by DI container. Implementation method is as shown below.

- `[projectName]-web/src/main/java/com/example/listener/todo/TodoMessageListener.java`

```
package com.example.listener.todo;

import org.springframework.jms.annotation.JmsListener;
import org.springframework.stereotype.Component;
import com.example.domain.model.Todo;
@Component
public class TodoMessageListener {

    @JmsListener(destination = "jms/queue/TodoMessageQueue")    // (1)
    public void receive(Todo todo) {
        // omitted
    }

}
```

Sr. No.	Description
(1)	Specify `@JmsListener` annotation for a method, for asynchronous receiving. Specify Destination name for receiving in `destination` attribute.

A list of main attributes of `@JmsListener` annotation is shown below. For details and other attributes, refer Javadoc of @JmsListener annotation.

Sr. No.	Fields	Details
1.	`destination`	Specify Destination to be received.
2.	`containerFactory`	Specify Bean name of `DefaultJmsListenerContainerFactory`which manages the listener method. Default is `jmsListenerContainerFactory`.
3.	`selector`	Specify a message selector which acts as a condition for restricting the message to be received. When the value is not explicitly specified, default is “”(blank character) and all the messages can be received. For how to use, refer *When messages which are asynchronously received are to be restricted*.
3.	`concurrency`	Specify upper limit of parallel numbers for listener method.; Default value for `concurrency` attribute is 1. Lower limit and upper limit of parallel numbers can be specified. For example, specify “5-10” when lower limit is 5 and upper limit is 10. When the parallel number of listener method has reached specified upper limit, parallel processing is not done and a “waiting” state is reached. Value should be specified as required.

Fetching header information of messages

When the header information of messages is to be used in the listener method such as sending process results of asynchronous receiving to the Destination specified on Producer side (Value of header attribute `JMSReplyTo`), `@org.springframework.messaging.handler.annotation.Header` annotation is used.
Implementation example is shown below.

- `[projectName]-web/src/main/java/com/example/listener/todo/TodoMessageListener.java`

```
@JmsListener(destination = "jms/queue/TodoMessageQueue")
public JmsResponse<Todo> receiveAndResponse(
        Todo todo, @Header("jms_replyTo") Destination␣
↪storeResponseMessageQueue) { // (1)

    // omitted

     return JmsResponse.forDestination(todo, storeResponseMessageQueue);
}
```

Sr. No.	Description
(1)	Specify `@Header` annotation to fetch value of header attribute `JMSReplyTo` of receiving message. For a key specified while fetching JMS standard header attribute, refer JmsHeaders constants definition.

Process results after asynchronous receiving are sent as messages

A method which sends the process results of method defined in `@JmsListener` annotation, to Destination as a response message.
Following 2 methods can be listed for specifying sending destination of process results.

- When sending destination of process results is specified statically
- When sending destination of process results is specified dynamically

Respective descriptions are as below.

•When sending destination of process results is specified statically

Process results can be sent as messages to a fixed destination by defining `@SendTo` annotation which specifies a destination, for a method defined by `@JmsListener` annotation.

Implementation example is as shown below.

- `[projectName]-web/src/main/java/com/example/listener/todo/TodoMessageListener.ja`

```
@JmsListener(destination = "jms/queue/TodoMessageQueue")
@SendTo("jms/queue/ResponseMessageQueue") // (1)
public Todo receiveMessageAndSendTo(Todo todo) {

    // omitted
    return todo; // (2)
}
```

Sr. No.	Description
(1)	Default sending destination of process results can be specified by defining `@SendTo` annotation.
(2)	Return data to be sent to Destination defined in `@SendTo` annotation. Permissible return value types are the classes which implement `org.springframework.messaging.Message`, `javax.jms.Message`, `String`, `byte` array, `Map` and `Serializable` interface.

- **When sending destination of process results is changed dynamically**

  When sending destination is to be changed dynamically, `forDestination` or `forQueue` methods of `JmsResponse` class are used

  Process results can be sent to any Destination by dynamically changing Destination for sending or Destination name. Implementation example is as shown below.

  - `[projectName]-web/src/main/java/com/example/listener/todo/TodoMessageListener.ja`

```
@JmsListener(destination = "jms/queue/TodoMessageQueue")
public JmsResponse<Todo> receiveMessageJmsResponse(Todo todo) {

    // omitted

    String resQueue = null;

    if (todo.isFinished()) {
        resQueue = "jms/queue/FinihedTodoMessageQueue";
    } else {
        resQueue = "jms/queue/ActiveTodoMessageQueue";
    }

    return JmsResponse.forQueue(todo, resQueue); // (1)
}
```

Sr. No.	Description
(1)	When Queue for sending is to be changed in accordance with the process details, `forDestination` or `forQueue` methods of `JmsResponse` class are used. In this example, messages are sent from Destination name by using `forQueue` method. **Note:** When `forQueue` method of `JmsResponse` class is used, Destination name is used as a string. For resolving destination name, `DestinationResolver` specified in `DefaultMessageListenerContainer` is used.

Note: When sending destination of process results is specified on Producer side

Using the implementation below, messages of process results can be sent to any destination specified on Producer side.

Implementation location	Implementation details
Producer side	Specify Destination in header attribute `JMSReplyTo` of messages in accordance with JMS standards. For editing of header attribute, refer *When message header is to be edited and sent synchronously*.
Consumer side	Return objects which send a message.

Header attribute `JMSReplyTo` is given priority over the default Destination specified on the Consumer side. For details, refer Response management.

When messages which are asynchronously received are to be restricted

The messages to be received can be restricted by specifying a message selector at the time of receiving.

- `[projectName]-web/src/main/java/com/example/listener/todo/TodoMessageListener.java`

```
@JmsListener(destination = "jms/queue/MessageQueue" , selector = "TodoStatus␣
↪= 'deleted'")    // (1)
public void receive(Todo todo) {
    // omitted
}
```

Sr. No.	Description
(1)	Conditions for receiving can be set by using `selector` attribute. `TodoStatus` of header attribute receives only `deleted` messages. Message selector is based on subset of SQL92 conditional expression syntax. For details, refer Message Selectors.

Input check for the messages which are received asynchronously

An input check must be carried out to ensure that the messages which retain invalid data are not processed in the business logic from a security viewpoint.

Input check is implemented in Service method by using Method Validation and the exception at the time of input check error is handled by the listener method.

This is done to avoid the occurrence of unnecessary rollback process for the exception occurring at the time of input check error, while performing transaction control. For transaction control, refer *Transaction control*.

For details of how to configure and implement Method Validation, refer *Method Validation* of *Input Validation*.

An implementation example is shown below wherein input check is performed for `Todo` object shown in *Basic synchronous sending*.

- `[projectName]-domain/src/main/java/com/example/domain/service/todo/TodoServiceImpl.`

```
package com.example.domain.service.todo;

import javax.validation.Valid;
import org.springframework.validation.annotation.Validated;
import com.example.domain.model.Todo;

@Validated // (1)
public interface TodoService {

    void updateTodo(@Valid Todo todo); // (2)

}
```

Sr. No.	Description
(1)	Declare the interface as a target for input check by attaching a `@Validated` annotation.
(2)	Specify a constraint annotation of Bean Validation as an argument for the method.

- `[projectName]-domain/src/main/java/com/example/domain/model/Todo.java`

```
package com.example.domain.model;

import java.io.Serializable;
import javax.validation.constraints.Null;

// (1)
public class Todo implements Serializable {

    private static final long serialVersionUID = -1L;

    // omitted

    @Null
    private String description;

    // omitted

    private boolean finished;

    // omitted

    public String getDescription() {
        return description;
    }

    public void setDescription(String description) {
        this.description = description;
    }

    public boolean isFinished() {
        return finished;
    }

    public void setFinished(boolean finished) {
        this.finished = finished;
    }

}
```

Sr. No.	Description
(1)	Define input check of JavaBean in Bean Validation. In this example, `@Null` annotation is set as an example. For details, refer "*Input Validation*".

- `[projectName]-web/src/main/java/com/example/listener/todo/TodoMessageListener.java`

```
@Inject
TodoService todoService;

@JmsListener(destination = "jms/queue/MessageQueue")
public void receive(Todo todo) {
    try {
        todoService.updateTodo(todo); // (1)
    } catch (ConstraintViolationException e) { // (2)
        // omitted
    }
}
```

Sr. No.	Description
(1)	Implement Service method which performs input check.
(2)	Capture `ConstraintViolationException` which occurs at the time of constraint violation. Any process can be performed after capturing the exception. For examples of sending messages to another Queue like using a Queue for storing logical error messages, refer *Exception handling at the time of asynchronous receiving*.

Transaction control

A transaction control function is used when data consistency is required to be guaranteed.

Transactions can be controlled at the time of asynchronous message receiving by setting a transaction manager for `<jms:listener-container/>`.

Note: When message returns to Queue, it is asynchronously received once again. Since the cause of error is not resolved, the operations of rollback and asynchronous receiving are repeated. A threshold value for number of resends after rollback can be set depending on JMS provider and when resend count exceeds the threshold value, the message is stored in Dead Letter Queue.

How to configure is shown below.

- `[projectName]-domain/src/main/resources/META-INF/spring/[projectName]-domain.xml`

```
<!-- (1) -->
<bean id="jmsAsyncReceiveTransactionManager"
   class="org.springframework.jms.connection.JmsTransactionManager">
   <!-- (2) -->
   <property name="connectionFactory" ref="connectionFactory" />
</bean>
```

Sr. No.	Description
(1)	Define a Bean for `JmsTransactionManager` of asynchronous receiving.
(2)	Specify `ConnectionFactory` which manages a transaction. Note that, `CachingConnectionFactory` cannot be used at the time of asynchronous receiving.

- `[projectName]-web/src/main/resources/META-INF/spring/applicationContext.xml`

```
<!-- (1) -->
<jms:listener-container
    factory-id="jmsListenerContainerFactory"
    destination-resolver="destinationResolver"
    concurrency="1"
    error-handler="jmsErrorHandler"
    cache="consumer"
    transaction-manager="jmsAsyncReceiveTransactionManager"/>
```

Sr. No.	Attribute name	Details
(1)	`cache`	Specify cache level to determine cache targets such as `Connection`, `Session` or `Consumer`. Default is `auto`. As described earlier in *Basic asynchronous receiving*, `consumer` is specified when connection is not pooled in the application server.
	`transaction-manager`	Specify Bean name for `JmsTransactionManager` to be used. Note that, `JmsTransactionManager` does not manage `CachingConnectionFactory`.

Warning: Since `Connection` or `Session` caching in the application is likely to be prohibited depending on the application server, cache validation and invalidation should be determined in accordance with the specifications of application server to be used.

Note: A method which performs exception handling other than roll back process in case of specific exceptions

When transaction control is enabled, the message returns to Queue due to roll back if the exception occurred in input check is thrown without getting captured. Since listener method asynchronously receives the message again which has returned in Queue, the sequence asynchronous receiving Error occurrence Rollback is repeated a number of times for JMS provider configuration. In case of an error for which the cause of the error is not resolved even after retry, the error handling is done so as not to throw an exception from listener method after capturing, to restrain futile processes mentioned above. For details, refer *Exception handling at the time of asynchronous receiving*.

In the application wherein DB transaction control is required, a transaction control policy must be determined after reviewing the relation between JMS and DB transactions based on business requirements.

- **When the transaction is to be committed and rolled back by separating JMS and DB transactions**

In some cases, JMS transaction is rolled back whereas only DB transaction is committed.

In such a situation, it is necessary to manage JMS and DB transactions separately.

JMS and DB transactions can be separately controlled by defining a `@Transactional` annotation in the Service class called by listener method.

Implementation example is given below wherein `jmsListenerContainerFactory` of *Transaction control* is used for transaction control of JMS and `transactionManager` defined in default configuration of Blank project is used for transaction control of DB.

- `[projectName]-web/src/main/java/com/example/listener/todo/TodoMessageListener.ja`

```
package com.example.listener.todo;

import javax.inject.Inject;
import org.springframework.jms.annotation.JmsListener;
import org.springframework.stereotype.Component;
import com.example.domain.service.todo.TodoService;
import com.example.domain.model.Todo;
@Component
public class TodoMessageListener {
    @Inject
    TodoService todoService;

    @JmsListener(destination = "TransactedQueue") // (1)
    public void receiveTransactedMessage(Todo todo) {

        todoService.update(todo);

    }
}
```

- `[projectName]-domain/src/main/java/com/example/domain/service/todo/TodoServiceIm`

```
package com.example.domain.service.todo;

import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import com.example.domain.model.Todo;

@Transactional // (2)
@Service
public class TodoServiceImpl implements TodoService {

    @Override
    public void update(Todo todo) {
        // omitted
    }
}
```

Sr. No.	Description
(1)	Define `@JmsListener` annotation and specify `DefaultJmsListenerContainerFactory` which enables transaction control of JMS. Since `@JmsListener` annotation refers to Bean name `jmsListenerContainerFactory` as a default, `containerFactory` attribute is omitted.
(2)	Define transaction boundary of DB. Since value is omitted, Bean name `transactionManager` is referred as a default. For details of `@Transactional` annotation, refer *Regarding transaction management* of *Domain Layer Implementation*.

Note: Nesting sequence for transaction boundary depends on the business requirements and JMS provider is often used for linking with external systems. In such a case, JMS transaction boundary is kept outside DB transaction boundary and recovery becomes easier when inward DB transaction is completed earlier. When DB transaction is committed and JMS transaction is rolled back, the message returns to the Queue. Hence the same message is processed again. It should be designed so as to allow DB update process to be re-tried at the time of reexecution of business process.

- **When JMS and DB transactions are to be committed and rolled back together**

  A method which uses global transaction by JTA exists for linking JMS and DB transactions, however "Best Effort 1 Phase Commit" is recommended since overheads are likely to occur for performance, among protocol characteristics. Refer below for details.

  Distributed transactions in Spring, with and without XA

  Spring Distributed transactions using Best Effort 1 Phase Commit

> **Warning: When transaction process results are not returned in JMS provider due to issues like loss of connection with JMS provider after receiving a message**
>
> When transaction process results are not returned in JMS provider due to issues like loss of connection with JMS provider after receiving a message, transaction handling depends on JMS provider. Hence, **a design considering loss of received messages, reprocessing of messages due to roll back** must be performed. Especially, when loss of messages is absolutely not permissible, **providing a system to compensate for the loss of messages or using a global transaction etc** must be adopted.

"Best Effort 1 Phase Commit" can be achieved by using `org.springframework.data.transaction.ChainedTransactionManager`.

An implementation example is shown below wherein `jmsAsyncReceiveTransactionManager` of *Transaction control* is used in JMS transaction control and `transactionManager` defined in default setting of Blank project is used in DB transaction control.

- `[projectName]-env/src/main/resources/META-INF/spring/[projectName]-env.xml`

```
<!-- (1) -->
<bean id="chainedTransactionManager" class="org.springframework.data.
↪transaction.ChainedTransactionManager">
    <constructor-arg>
        <list>
            <!-- (2) -->
            <ref bean="jmsAsyncReceiveTransactionManager" />
            <ref bean="transactionManager" />
        </list>
    </constructor-arg>
</bean>
```

Sr. No.	Description
(1)	Define a Bean for `ChainedTransactionManager`.
(2)	Specify a transaction manager of JMS and DB. Transaction starts in a registered sequence and transaction is committed in the reverse sequence.

- `[projectName]-web/src/main/resources/META-INF/spring/applicationContext.xml`

```
<!-- (1) -->
<jms:listener-container
    factory-id="chainedTransactionJmsListenerContainerFactory"
    destination-resolver="destinationResolver"
    concurrency="1"
    error-handler="jmsErrorHandler"
    cache="consumer"
    transaction-manager="chainedTransactionManager"
    acknowledge="transacted"/>
```

Sr. No.	Attribute name	Details
(1)	-	Define `<jms:listener-container/>` for using `ChainedTransactionManager`.
	`factory-id`	Specify Bean name of `DefaultJmsListenerContainerFactory`. In this example, Bean name is set to `chainedTransactionJmsListenerContainerFactory` by considering its combined use with `<jms:listener-container/>` of *Basic asynchronous receiving*.
	`transaction-manager`	Specify Bean name for `ChainedTransactionManager`.
	`acknowledge`	Specify `transacted` in check response mode in order to enable the transaction. Default is `auto`. **Note:** In `DefaultMessageListenerContainer`, when Bean of implementation class of `org.springframework.transaction.support.ResourceTransa` is specified in `transaction-manager`attribute, the transaction control which uses this Bean is enabled. However, since `ChainedTransactionManager` does not implement `ResourceTransactionManager`, transaction control is not enabled. `transacted` must be specified in `acknowledge` attribute in order to enable transaction control. Accordingly, Session acting as a target for transaction control is generated and transaction control for `ChainedTransactionManager` is enabled.

Implementation example which uses the configuration above is shown below.

- `[projectName]-web/src/main/java/com/example/listener/todo/TodoMessageListener.ja`

```
@Inject
TodoService todoService;

@JmsListener(containerFactory =
↪"chainedTransactionJmsListenerContainerFactory", destination = "jms/
↪queue/TodoMessageQueue") // (1)
public void receiveTodo(Todo todo) {
    // omitted
}
```

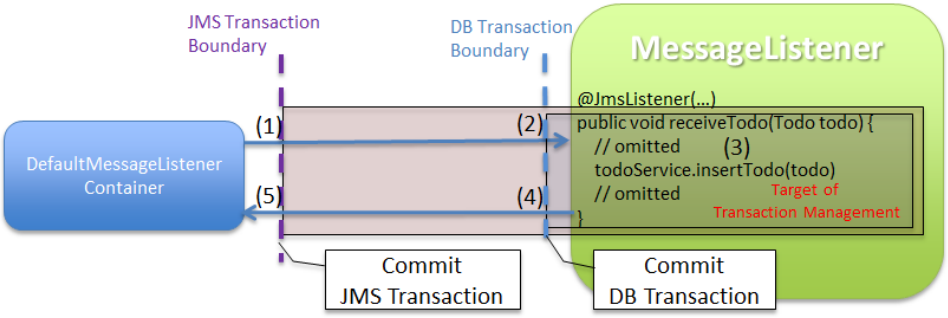
Sr. No.	Description
(1)	Since the Bean name uses `DefaultJmsListenerContainerFactory` of `chainedTransactionJmsListenerContainerFactory`, `chainedTransactionJmsListenerContainerFactory` is specified in `containerFactory` attribute.

Behaviour when an application is created in accordance with the configuration and implementation example above is explained.

- **When processing of listener method is successfully completed**

  JMS and DB transactions are committed together.

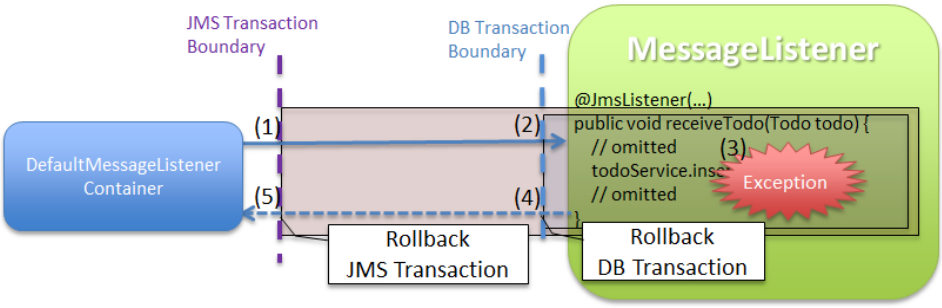
  Transaction starts in the sequence of JMS, DB, and transaction ends in the sequence of DB, JMS after executing listener method.



Sr. No.	Description
(1)	Start JMS transaction.
(2)	Start DB transaction.
(3)	Terminate listener method successfully.
(4)	Commit DB transaction and terminate DB transaction.
(5)	Commit JMS transaction and terminate JMS transaction.

– **When an unexpected exception occurs in listener method or business logic**

When exception occurs in listener method



Sr. No.	Description
(1)	Start JMS transaction.
(2)	Start DB transaction.
(3)	An unexpected exception occurs in listener method or business logic.
(4)	Roll back DB transaction and terminate DB transaction.
(5)	Roll back JMS transaction and terminate JMS transaction. Since JMS transaction is rolled back, the message is returned to the Queue.

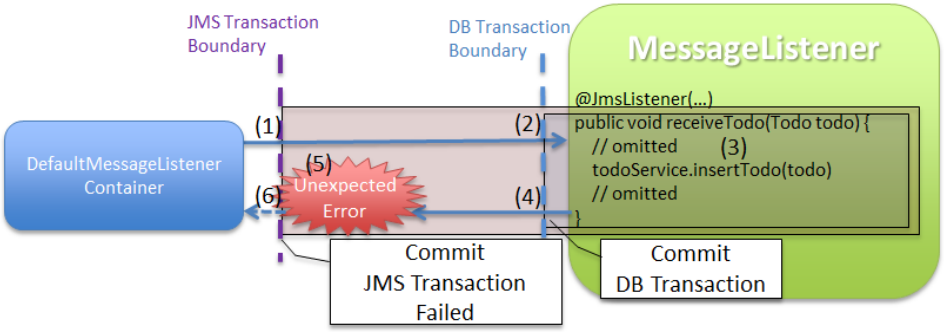
- **When connection with JMS provider is lost after receiving the message and only DB transaction is committed**

  JMS transaction handling is dependent on JMS provider, however, since DB transaction is committed, an inconsistency is likely to occur in JMS and DB status.

  Considering the possibility of rollback of JMS transaction, integrity of the data must be ensured when same message is received multiple times.

  An example to ensure data integrity is shown below.

  - When the process after asynchronous receiving is executed multiple times, process must be designed to ensure same status after the processing.
  - Design so as to record `JMSMessageID`. `JMSMessageID` recorded earlier and `JMSMessageID` of received message are compared for each received message and when they match, received message is destroyed.



Sr. No.	Description
(1)	Start JMS transaction.
(2)	Start DB transaction.
(3)	Successfully terminate listener method.
(4)	Commit DB transaction and terminate DB transaction.
(5)	An unexpected error like JMS provider disconnected occurs.
(6)	An error is likely to occur in JMS transaction commit. Hence, a system to ensure consistency must be provided for loss of messages etc.

Note: When it is necessary to stringently control multiple transactions like JMS and DB by avoiding events given above, use of global transaction is considered. For global transaction, refer various product manuals.

Exception handling at the time of asynchronous receiving

While performing transaction control, exceptions must be handled considering the rollback process.

For details of transaction control, refer *Transaction control*.

Exception handling of JMS is classified in 2 patterns given below based on the objective.

Table.8.7 **Table-Exception handling patterns**

Sr. No.	Objective of handling	An example of exception for handling	Handling method	Handling unit
(1)	When exceptions occurring in business layer are to be individually handled	Business exception like input check error	Listener method (try-catch)	Listener method unit
(2)	When the exceptions thrown by listener method are to be handled uniformly	System exceptions like input output error etc	`ErrorHandler` of `DefaultMessageListenerContainer`	JMSListenerContainer unit

- **When exceptions occurred in the business layer are to be handled individually**

  The exceptions occurring in the business layer like “message contents are invalid” are trapped (try-catch) by listener method and handled by listener method unit.

  While performing transaction control, since exceptions must be thrown in `DefaultMessageListenerContainer` for the cases which require rollback, the captured exception must be thrown again.

  Implementation example is shown below.

  - `[projectName]-web/src/main/java/com/example/listener/todo/TodoMessageListener.ja`

```
@Inject
TodoService todoService;

@JmsListener(destination = "jms/queue/TodoMessageQueue")
public JmsResponse<Todo> receiveTodo(Todo todo) {
    try {
        todoService.insertTodo(todo);
    } catch (BusinessException e) {
```

```
        return JmsResponse.forQueue(todo, "jms/queue/ErrorMessageQueue"); //
↪(1)
    }
    return null; // (2)
}
```

Sr. No.	Description
(1)	An object can be sent to a Queue for storing a logical error message by using `forQueue` method of `JmsResponse` class. In this example, since `BusinessException` which outputs the log in AOP is captured, log output process is not described explicitly. However, exceptions must be handled so as not to eliminate the cause of exception. When the messages are to be processed after roll back, by performing transaction control, the exception thus captured must be thrown.
(2)	When the messages are not sent, set return value to `null`.

- **When the exceptions thrown by listener method are to be handled uniformly**

  While commonly handling exceptions, implementation class of `ErrorHandler` defined in `error-handler` attribute of `<jms:listener-container/>` is used.

  Configuration method is shown below.

  - `[projectName]-web/src/main/resources/META-INF/spring/applicationContext.xml`

```
<!-- (1) -->
<jms:listener-container
    factory-id="jmsListenerContainerFactory"
    destination-resolver="destinationResolver"
    concurrency="1"
    error-handler="jmsErrorHandler"
    cache="consumer"
    transaction-manager="jmsAsyncReceiveTransactionManager"/>

<!-- (2) -->
<bean id="jmsErrorHandler"
    class="com.example.domain.service.todo.JmsErrorHandler">
</bean>
```

Sr. No.	Description
(1)	Define a Bean name of error handling class in `error-handler` attribute of `<jms:listener-container/>`.
(2)	Define a Bean for error handling class.

Implementation method is as shown below.

- `[projectName]-web/src/main/java/com/example/listener/todo/JmsErrorHandler.java`

```
package com.example.listener.todo;

import org.springframework.util.ErrorHandler;
import org.terasoluna.gfw.common.exception.SystemException;

public class JmsErrorHandler implements ErrorHandler {  // (1)

   @Override
    public void handleError(Throwable t) { // (2)
        // omitted
        if (t.getCause() instanceof SystemException) {  // (3)

            // omitted system error handling

        } else {
            // omitted error handling
        }
    }
}
```

Sr. No.	Description
(1)	Create an error handling class which implements `ErrorHandler` interface.
(2)	Exception occurring in the listener method is wrapped in `org.springframework.jms.listener.adapter.ListenerExecutionFailedExcept` and passed as an argument.
(3)	Determine any exception class and implement error handling associated with the exception. `t.getCause()` must be executed to fetch the exception occurred in the application.

A method wherein the messages are synchronously received

Synchronous receiving in JMS provider is achieved by using `JmsMessagingTemplate`.

Messages can be received within any timing by using synchronous receiving.

Architecture should be determined after thoroughly examining the implementation method which does not use synchronous receiving.

Configuration of Bean definition file for synchronous receiving is shown below.

- `[projectName]-domain/src/main/resources/META-INF/spring/[projectName]-infra.xml`

```
<bean id="cachingConnectionFactory"
   class="org.springframework.jms.connection.CachingConnectionFactory"> <!--␣
↪(1) -->
   <property name="targetConnectionFactory" ref="connectionFactory" /> <!--␣
↪(2) -->
   <property name="sessionCacheSize" value="1" />  <!-- (3) -->
</bean>

<!-- (4) -->
<bean id="jmsTemplate" class="org.springframework.jms.core.JmsTemplate">
   <property name="connectionFactory" ref="cachingConnectionFactory" />
   <property name="destinationResolver" ref="destinationResolver" />
</bean>
```

```
<!-- (5) -->
<bean id="jmsMessagingTemplate" class="org.springframework.jms.core.
↪JmsMessagingTemplate">
    <property name="jmsTemplate" ref="jmsTemplate"/>
</bean>
```

Sr. No.	Description
(1)	Define a Bean for `org.springframework.jms.connection.CachingConnectionFactory` which performs caching of `Session` and `MessageProducer/Consumer`. Cache function can be used by using JMS provider specific `ConnectionFactory` which is looked up in Bean definition or JNDI name, by wrapping it in `CachingConnectionFactory` instead of using it as it is.
(2)	Specify `ConnectionFactory`which is wrapped up in Bean definition or JNDI name.
(3)	Set cache number of `Session`. (default value is 1) Although 1 is specified in this example, cache number should be changed appropriately corresponding to performance requirement. If a session is required to be continued even after exceeding this cache number, a new session is created and destroyed repeatedly without using a cache. It is likely to cause reduction in process efficiency resulting in performance degradation.
(4)	Define a Bean for `JmsTemplate`. `JmsTemplate` alternates for a low level API handling (JMS API calling). For the attributes that can be configured, refer list of attributes of `JmsTemplate` shown below.
(5)	Define a Bean for `JmsMessagingTemplate`. Inject `JmsTemplate` which alternates as a synchronous receiving process.

List of attributes of `JmsTemplate` related to synchronous receiving are shown below.

Configuration must be done as required.

Sr. No.	Configuration item	Details	Mandatory	Default value
1.	`connectionFactory`	Set `ConnectionFactory`to be used.	○	Nil (since it is mandatory)
2.	`pubSubDomain`	Set for messaging model. Set `false`for PTP (Queue) model and `true`for Pub/Sub (Topic).	-	`false`
3.	`sessionTransacted`	Set whether the transaction is controlled in the session. In this guideline, since transaction control described later is used, default `false` is recommended.	-	`false`
4.	`sessionAcknowledgeMode`	Set confirmation response mode of session for `sessionAcknowledgeMode`. For details, refer JavaDoc of JMSTemplate. **Todo** Details of sessionAcknowledgeMode will be added later.	-	1
5.	`receiveTimeout`	Set timeout period (milliseconds) at the time of synchronous receiving. If it is not set, wait till the message is received. If the timeout period is not set, it impacts the subsequent processes, hence an appropriate timeout period must always be set.	-	0

次のページに続く

Table. 8.8 – 前のページからの続き

Sr. No.	Configuration item	Details	Mandatory	Default value
6.	`messageConverter`	Set message converter. Default can be used in the range introduced in the guideline.	-	`SimpleMessageCo` is used.
7.	`destinationResolver`	Set DestinationResolver. This guideline recommends setting `JndiDestinationResolver` which performs name resolution by JNDI.	-	`DynamicDestinat` is used. (If `DynamicDestinat` is used, name resolution of Destination is performed by JMS provider.)
8.	`defaultDestination`	Specify existing Destination. When the Destination is not specified explicitly, this Destination is used.	-	null(no existing Destination)

(*1)`org.springframework.jms.support.converter.SimpleMessageConverter`

(*2)`org.springframework.jms.support.destination.DynamicDestinationResolver`

Messages are received synchronously by `receiveAndConvert` message of `JmsMessagingTemplate` class. Implementation example is shown below.

- `[projectName]-domain/src/main/java/com/example/domain/service/todo/TodoServiceImpl.`

```
package com.example.domain.service.todo;

import javax.inject.Inject;
import org.springframework.jms.core.JmsMessagingTemplate;
import org.springframework.stereotype.Service;
import com.example.domain.model.Todo;

@Service
public class TodoServiceImpl implements TodoService {
    @Inject
    JmsMessagingTemplate jmsMessagingTemplate;

    @Override
    public String receiveTodo() {

        // omitted
        Todo retTodo = jmsMessagingTemplate.receiveAndConvert("jms/queue/
↪TodoMessageQueue", Todo.class);    // (1)

    }
}
```

Sr. No.	Description
(1)	Receive message from specified Destination using `receiveAndConvert` method of `JmsMessagingTemplate`. `receiveAndConvert` method can fetch the class for which type conversion is done, by specifying the class for conversion, in the second argument. A header item can be fetched by `Message` object of Spring Framework, by using `receive` method.

8.2.3 Appendix

JMS provider dependent configuration

Configuration varies for each JMS provider. Configuration for each JMS provider is explained below.

While using Apache ActiveMQ

Configuration while using Apache ActiveMQ is explained.

- **JMS provider specific configuration for application server**

  JMS provider requires specific configuration.

  In Apache ActiveMQ, environment variable must be added to starting variable of application server to ensure that it consists of objects wherein payload of received messages is permissible.

  For details, refer ObjectMessage.

  An example of adding environment variable to startup argument of Apache Tomcat is shown below. Refer Configuring JBoss EAP to Run as a Service in case of JBoss Enterprise Application Platform 7.0, Service Configuration in case of JBoss Enterprise Application Platform 6.4 and Starting Managed Servers with a Startup Script in case of Weblogic.

  - `$CATALINA_HOME/bin/setenv.sh`

```
# omitted
# (1)
-Dorg.apache.activemq.SERIALIZABLE_PACKAGES=java.lang,java.util,org.apache.
 ↪activemq,org.fusesource.hawtbuf,com.thoughtworks.xstream.mapper,com.
 ↪example.domain.model
# omitted
```

Sr. No.	Description
(1)	Add package for an arbitrary object to be allowed. `java.lang`,`java.util`, `org.apache.activemq`, `org.fusesource.hawtbuf` and `com.thoughtworks.xstream.mapper` are the settings required while using Apache ActiveMQ. "com.example.domain.model" is added as a required configuration value in this sample.

- **Adding a library**

JMS API is not included in the `spring-jms` library.

Although JMS API is normally included in the JMS provider library, JMS API is added to pom.xml if JMS API is not included in JMS provider library.

`activemq-client` is added to pom.xml of domain project and web project as a library for build.

Further, `activemq-client` and its dependent libraries are added to the application server.

- `[projectName]-domain/pom.xml`
- `[projectName]-web/pom.xml`

```
<dependencies>

    <!-- (1) -->
    <dependency>
        <groupId>org.apache.activemq</groupId>
        <artifactId>activemq-client</artifactId>
        <scope>provided</scope>
    </dependency>

</dependencies>
```

Sr. No.	Description
(1)	Add client library of Apache ActiveMQ to dependencies as a build. Since JMS API is also incorporated in `activemq-client` library, it is not necessary to add JMS API as a library.

Note: In the above setting example, since it is assumed that the dependent library version is managed by the parent project terasoluna-gfw-parent , specifying the version in pom.xml is not necessary. The above dependent library used by terasoluna-gfw-parent is defined by Spring IO Platform.

Warning: The version of the library used while establishing a connection with Apache ActiveMQ is defined in Spring IO Platform which is used by TERASOLUNA Server Framework for Java. Hence, care must be taken while determining Apache ActiveMQ version. Note that, library and middleware

versions are likely to be inconsistent while upgrading the version of TERASOLUNA Server Framework for Java.

- **JNDI registration to application server**

  For registration of JNDI to application server, refer Manually integrating Tomcat and ActiveMQ.

- **Configuration when JNDI is not used.**

  Although this guideline recommends a method to resolve names using JNDI,

  JNDI is not used while connecting to JMS provider during the implementation of a simple test which is not run on the application server.

  In such a case, a Bean of implementation class of `ConnectionFactory` must be generated.

  Further, use of JNDI is also specified for Queue, however, if a Queue specified in the Destination by using a JMS provider function does not exist, a Queue of specified name can be dynamically generated.

  Internal Broker of Apache ActiveMQ must be used for establishing a connection without passing through the application server.

  For configuration of internal Broker for Apache ActiveMQ, refer How do I embed a Broker inside a Connection.

  Following configuration should be added to the context for the testing.

  - `[projectName]-domain/src/main/resources/META-INF/spring/[projectName]-infra.xml`

```
<!-- (1) -->
<bean id="connectionFactory" class="org.apache.activemq.
↪ActiveMQConnectionFactory">
    <constructor-arg value="tcp://localhost:61616"/>  <!-- (2) -->
</bean>
```

Sr. No.	Description
(1)	Define a Bean for `ConnectionFactory` of Apache ActiveMQ.
(2)	Specify beginning URL of Apache ActiveMQ. Beginning URL sets the value for each environment.

Note: When the configuration method of ConnectionFactory is to be changed by JNDI and bean definition, based on development phase, configuration should be described in `[projectName]-env/src/main/resources/META-INF/spring/[projectName]-env.xml`.

Mass-mailing of identical messages

If `JmsMessageTemplate` is to be used for mass mailing of identical message, memory usage is likely to increase.

Hence, a method wherein implementation is done by using `send` method of `JmsTemplate` class must be considered.

Reason being, an instance of `org.springframework.jms.core.MessageCreator` class is generated while sending a message in `JmsMessageTemplate`.

In order to prevent the generation of unnecessary instance, the messages are sent by `send` method of `JmsTemplate` class wherein `MessageCreator` instance is not generated during sending thus reducing the amount of memory used.

An example of code wherein a string is sent 100 times to an identical destination is shown below.

- `[projectName]-domain/src/main/java/com/example/domain/service/todo/TodoServiceImpl.`

```
package com.example.domain.service.todo;

import java.io.IOException;
import javax.inject.Inject;
import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.Session;
import javax.jms.TextMessage;
import org.springframework.jms.core.JmsTemplate;
import org.springframework.jms.core.MessageCreator;
import org.springframework.stereotype.Service;

@Service
```

```
public class TodoServiceImpl implements TodoService {

    @Inject
    JmsTemplate jmsTemplate; // (1)

    @Override
    public void sendManyMessage(final String messageStr) throws IOException {
        MessageCreator mc = new MessageCreator() { // (2)
            public Message createMessage(Session session) throws JMSException
↪{
                TextMessage message = session.createTextMessage(); // (3)
                message.setText(messageStr);

                // omitted
                return message;
            }
        };
        for (int i = 0; i < 100; i++) {
            jmsTemplate.send("jms/queue/TodoMessageQueue", mc); // (4)
        }
    }
}
```

Sr. No.	Description
(1)	When `JmsMessagingTemplate` is used, `MessageCreater` is generated at the time of sending. Hence, `JmsTemplate` which can define generation of `MessageCreater` by isolating it from sending, is used.
(2)	Generate instance of `MessageCreator` for creating `Message` of JMS.
(3)	When messages are sent by `send` method of `JmsTemplate` class, instances of `MessageCreator` are no longer generated for each loop and the amount of memory can be reduced.

Sending and receiving large size data

While handling data of large size like Image data (Data of size 1MB or more as a rough indication), `OutOfMemoryError` is likely to occur due to number of concurrent transactions and heap size. In standard API of JMS, only `StreamMessage` which sends primitive type data and `ByteMessage` which can send uninterpreted byte stream can handle large size data as a stream. Hence, a specific API offered for each JMS provider

is used instead of JMS API.

While using Apache ActiveMQ

A message of large size can be received or sent by using Blob Message. Implementation example is shown below.

Note: Since Apache ActiveMQ specific API is used while using `org.apache.activemq.BlobMessage`, `Message` and `CachingConnectionFactory` offered by Spring Framework cannot be used. It is recommended to separately define `JmsTemplate` for `BlobMessage` while using `BlobMessage` considering the impact on the performance.

- **Configuration**

  While sending a message by using `BlobMessage`, the message is stored in the server which is temporarily run by Apache ActiveMQ, instead of heap area. Definition example for storage destination for the messages is shown below.

  - `[projectName]-domain/src/main/resources/META-INF/spring/[projectName]-infra.xml`

```
<bean id="connectionFactory"
   class="org.apache.activemq.ActiveMQConnectionFactory">
    <property name="brokerURL">
      <!-- (1) -->
      <value>tcp://localhost:61616?jms.blobTransferPolicy.uploadUrl=/tmp</
↪value>
    </property>
</bean>
```

Sr. No.	Description
(1)	Define a directory of Apache ActiveMQ server which temporarily stores messages. `http://localhost:8080/uploads/` is set as a default in `jms.blobTransferPolicy.uploadUrl` wherein location for temporary file can be specified by overloading default or `brokerURL`. For example, the file is temporarily stored in `/tmp`.

- **Sending**

  An implementation example of sending class which use `Blob Message` is shown below.

  - `[projectName]-domain/src/main/java/com/example/domain/service/todo/TodoServiceIm`

```
package com.example.domain.service.todo;
```

```
import java.io.IOException;
import java.io.InputStream;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import javax.inject.Inject;
import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.Session;
import org.apache.activemq.ActiveMQSession;
import org.apache.activemq.BlobMessage;
import org.springframework.jms.core.JmsTemplate;
import org.springframework.jms.core.MessageCreator;
import org.springframework.stereotype.Service;

@Service
public class TodoServiceImpl implements TodoService {
    @Inject
    JmsTemplate jmsTemplate;

    @Override
    public void sendBlobMessage(String inputFilePath) throws IOException {

        Path path = Paths.get(inputFilePath);
        try (final InputStream inputStream = Files.newInputStream(path)) {

            jmsTemplate.send("jms/queue/TodoMessageQueue", new␣
↪MessageCreator() {
                public Message createMessage(Session session) throws␣
↪JMSException {

                    ActiveMQSession activeMQSession = (ActiveMQSession)␣
↪session;  // (1)

                    BlobMessage blobMessage = activeMQSession.
↪createBlobMessage(inputStream);  // (2)
                    return blobMessage;
                }
            });
        }
    }
}
```

Sr. No.	Description
(1)	Use `org.apache.activemq.ActiveMQSession` - a Apache ActiveMQ specific API in `BlobMessage`.
(2)	Generate `BlobMessage` from `ActiveMQSession`by specifying sending data. Arguments of `createBlobMessage` method can be specified by `File`, `InputStream` and `URL` class.

- **Receiving**

  An implementation example of receiving class is shown below.

  - `[projectName]-web/src/main/java/com/example/listener/todo/TodoMessageListener.ja`

```
package com.example.listener.todo;

import java.io.IOException;
import javax.inject.Inject;
import javax.jms.JMSException;
import org.apache.activemq.BlobMessage;
import org.springframework.jms.annotation.JmsListener;
import org.springframework.stereotype.Component;
import com.example.domain.service.todo.TodoService;
@Component
public class TodoMessageListener {
    @Inject
    TodoService todoService;
    @JmsListener(destination = "jms/queue/TodoMessageQueue")
    public void receiveBlobMessage(BlobMessage message) throws IOException,
 ↪ JMSException {
     todoService.fileInputBlobMessage(message);
        // omitted
    }
}
```

  - `[projectName]-domain/src/main/java/com/example/domain/service/todo/TodoServiceIm`

```
package com.example.domain.service.todo;

import java.io.IOException;
import java.io.InputStream;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import org.apache.activemq.BlobMessage;
```

```
import org.springframework.stereotype.Service;

@Service
public class TodoServiceImpl implements TodoService {

    @Override
    public void fileInputBlobMessage(BlobMessage message) throws␣
↪IOException {
        try(InputStream is =  message.getInputStream()){    // (1)
            Path path = Paths.get("outputFilePath");
            Files.copy(is, path);
            // omitted
        }
    }
}
```

Sr. No.	Description
(1)	Fetch data from received `BlobMessage` as `InputStream`.

9

Security countermeasures

9.1 Spring Security Overview

Spring Security is a framework which is used while implementing the security countermeasure function in the application. Spring Security can also be used in stand-alone application, but generally it is used while implementing the security countermeasures for Web application deployed in servlet container. This chapter describes only those functions provided by Spring Security which are considered to be frequently used in general Web application.

Tip: Functions not introduced in the guideline

Spring Security also provides many functions not introduced in this guideline. Refer Spring Security Reference to know all functions provided by Spring Security.

Note: Spring Security version

This guideline assumes the use of Spring Security 4.0 or higher version. Various changes have been applied for upgrading version of Spring Security to 4.0 and the samples described later also have used Spring Security 4.

Refer Migrating from Spring Security 3.x to 4.x (XML Configuration) for the changes.

9.1.1 Spring Security functions

Basic functions of security countermeasures

Spring Security provides the following functions as the basic functions of security countermeasures.

Table.9.1 **Basic functions of security countermeasures**

Function	Description
Authentication function	A function that checks whether the application user is valid.
Authorization function	A function that controls access to the resources and processes provided by the application.

Function to strengthen security countermeasures

Spring Security provides several functions besides the basic functions of authentication and authorization to enhance Web application security.

Table.9.2 **Function to strengthen security countermeasures**

Function	Description
Session management function	A function that protects the user from session hijacking attack and session fixation attack, A function to control session lifecycle (generate, discard, time out).
CSRF countermeasure function	A function to protect the user from Cross site request forgeries (CSRF) attack.
Security header output function	A function to link with security countermeasure function of Web browser and protect the user from the attack where the browser function is misused.

9.1.2 Spring Security architecture

Overview of Spring Security architecture and the role of key components those configure Spring Security are described before explaining each function in detail.

Note: The developers need not be directly aware of the contents described over here while using the default operation provided by Spring Security as it is or while using the mechanism to support the configuration of Spring Security. Therefore, this section can be skipped while reading when the user first wants to know how to use each function.

However, since the contents described here are required while customizing the default operation of Spring Security,

it is recommended that the application architect should go through it once.

Spring Security module

First, the module provided by Spring Security used as a framework stack is introduced.

Set of framework stack modules

Framework stack module is as follows. This guideline also describes the method to implement security countermeasures using these modules.

Table.9.3 **Set of framework stack modules**

Module name	Description
`spring-security-core`	A core component required to implement authentication and authorization functions has been stored. A component included in this module can also be used in the applications executed in stand-alone environment.
`spring-security-web`	A component required to implement security countermeasures of Web application has been stored. A component included in this module performs the process that depends on Web layer (like servlet API).
`spring-security-config`	A component (like the class that supports configuration or the class that analyses XML namespace) to support the setup of components provided by each module has been stored. If this module is used, bean for Spring Security can be defined easily.
`spring-security-taglibs`	A JSP tag library to access the authentication information and authorization function has been stored.
`spring-security-acl`	A component required for authorized control of domain objects like Entity using Access Control List (ACL) has been stored. Since this module is included in the framework stack for the sake of dependency, the method of using this module is not described in this guideline.

Set of modules used according to the requirements

The following modules those are not included in the framework stack, are also provided to support the authentication methods used in general. Use of these modules also needs to be reviewed as per the security requirements.

Table.9.4 **Set of modules used according to the requirements**

Module name	Description
`spring-security-remoting`	A component required to access DNS via JNDI, to access Website for which Basic authentication is necessary and to access the methods via RMI wherein security countermeasures are implemented using Spring Security has been stored.
`spring-security-aspects`	A component required to use AspectJ function while applying the authorization function for the Java methods has been stored. This module is not required when Spring AOP is used as ACF.
`spring-security-messaging`[*5]	A component to add security countermeasures for Web Socket function of Spring has been stored.
`spring-security-data`[*5]	A component to enable accessing authentication information from Spring Data function has been stored.
`spring-security-ldap`	A component required to implement authentication using Lightweight Directory Access Protocol (LDAP) has been stored.
`spring-security-openid`	A component required to implement authentication using OpenID[*1] has been stored.
`spring-security-cas`	A component required to link with Central Authentication Service (CAS)[*2] has been stored.
`spring-security-crypto`	A component for encryption, key generation and password encoding using hash algorithm has been stored. Class contained in this module is also included in `spring-security-core` in framework stack module.

Test module

A module to support the test has been added from Spring Security 4.0.

Table.9.5 **Test module**

Module name	Description
`spring-security-test`[*5]	A component to support the testing of class that depends on Spring Security has been stored. If this module is used, the authentication information required at the time of JUnit testing can be setup easily. Note that, a component which is used by linking with component (`MockMvc`) for testing of Spring MVC is also included.

Set of related modules used according to the requirements

Further, several related modules are also provided.

[*5] It is a module added from Spring Security 4.0.
[*1] In a simple definition, OpenID is a mechanism 'to enable login to multiple sites with 1 ID'.
[*2] CAS is a server component for single sign on provided as OSS.Refer https://www.apereo.org/cas for details.

Table.9.6 **Set of main related modules used according to the requirements**

Module name	Description
`spring-security-oauth2`[*3]	A component required to implement API authorization using the mechanism of OAuth 2.0[*4] has been stored.
`spring-security-oauth`[*3]	A component required to implement API authorization using the mechanism of OAuth 1.0 has been stored.

Framework processing

Spring Security has adopted the architecture that implements security countermeasures for Web application using the servlet filter mechanism and executes the processes in the following flow.

Sr. No.	Description
(1)	Client sends a request to the Web application.
(2)	`FilterChainProxy` class (servlet filter) of Spring Security receives the request, calls method of `HttpFirewall` interface and incorporates the firewall function for `HttpServletRequest` and `HttpServletResponse`.
(3)	`FilterChainProxy` class assigns process to Security Filter (servlet filter) class for security countermeasures provided by Spring Security.
(4)	Security Filter consists of multiple classes and subsequent servlet filter is called if servlet filter process is successfully completed.
(5)	When the last Security Filter process is successfully completed, subsequent process (like servlet filter or servlet) is called and the resources in the Web application are accessed.
(6)	`FilterChainProxy` class sends response of resources returned by the Web application to the client.

The key component that configures the framework process for the Web application is as follows. For details, refer Spring Security Reference -The Security Filter Chain-.

[*3] Refer http://projects.spring.io/spring-security-oauth/ for details.

[*4] OAuth 2.0 is the improved version of issues (like the complexity of signature and authentication flow, incompatibility of client app on mobile or desktop) faced by OAuth 1.0 and it is not backward compatible with OAuth 1.0.

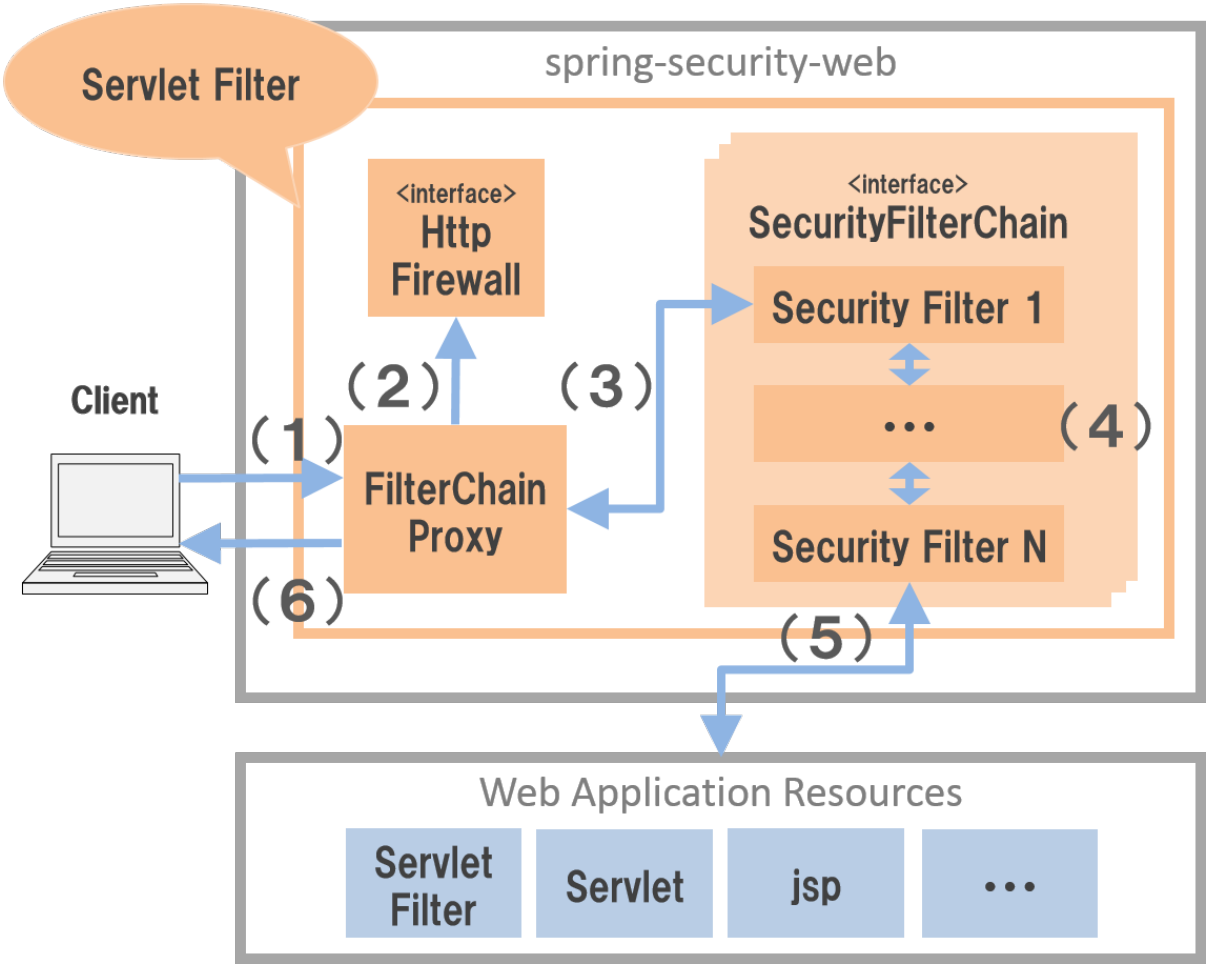


Figure.9.1 **Spring Security Framework architecture**

FilterChainProxy

`FilterChainProxy` class is a servlet filter class that is an entry point of the framework process for the Web application. This class controls the entire flow of the framework process and assigns specific security countermeasure process to Security Filter.

HttpFirewall

`HttpFirewall` interface incorporates the firewall function for `HttpServletRequest` and `HttpServletResponse`. By default, `DefaultHttpFirewall` class is used and the checks for Directory traversal attack and HTTP response splitting attack are implemented.

SecurityFilterChain

`SecurityFilterChain` interface manages Security Filter list to be applied to the request received by `FilterChainProxy`. By default, `DefaultSecurityFilterChain` class is used and the Security Fil-

ter list to be applied is managed for each pattern of request URL.

For example, security countermeasures for the details those vary according to the URL can be applied if the bean is defined as follows.

- Definition example of xxx-web/src/main/resources/META-INF/spring/spring-security.xml

```
<sec:http pattern="/api/**">
    <!-- ... -->
</sec:http>

<sec:http pattern="/ui/**">
    <!-- ... -->
</sec:http>
```

Security Filter

Security Filter class is a servlet filter class that provides a process required to implement framework function and security countermeasure function.

Spring Security is a mechanism to implement the security countermeasures for the Web application by linking multiple Security Filters. Here, core class required to implement authentication and authorization functions is introduced. For details, refer Spring Security Reference -Core Security Filters-.

Table.9.7 **Core Security Filter**

Class name	Description
`SecurityContextPersistenceFilter`	A class that provides a process to share the authentication information across the requests. Authentication information is shared across the requests by storing it in `HttpSession`, in the default implementation.
`UsernamePasswordAuthenticationFilter`	A class that performs the authentication process using user name and password specified in the request parameter. It is used at the time of form authentication.
`LogoutFilter`	A class that performs logout process.
`FilterSecurityInterceptor`	A class to execute authorization process for HTTP request (`HttpServletRequest`).
`ExceptionTranslationFilter`	A class that handles the exception occurred in `FilterSecurityInterceptor` and controls response returned to the client. It returns response that prompts authentication in case of access by unauthenticated user and response that notifies authorization error in case of access by authenticated user in the default implementation.

9.1.3 Spring Security setup

A setup method to apply Spring Security to the Web application is explained.

Here, the simplest method of setup that applies Spring Security to the Web application and displays the default login screen provided by Spring Security is explained. Customization method and extension method required in real application development are described sequentially in the following sections.

Note: When the development project is created from Blank project , the setting described here is already configured. For how to create a development project, refer '*Create Web application development project*' .

Applying dependent library

First, the common library that uses Spring Security as dependency is applied. Refer *Building blocks of Common Library* for the relation between Spring Security and common library.

This guideline assumes that the development project is created using Maven.

- Setup example of xxx-domain/pom.xml

```
<dependency>
    <groupId>org.terasoluna.gfw</groupId>
    <artifactId>terasoluna-gfw-security-core</artifactId>  <!-- (1) -->
</dependency>
```

- Setup example of xxx-web/pom.xml

```
<dependency>
    <groupId>org.terasoluna.gfw</groupId>
    <artifactId>terasoluna-gfw-security-web</artifactId>  <!-- (2) -->
</dependency>
```

Sr. No.	Description
(1)	Add terasoluna-gfw-security-core to dependency when Spring Security function is used in domain layer project.
(2)	Add terasoluna-gfw-security-web to dependency when Spring Security function is used in application layer project.

Note: In the above setting example, since it is assumed that the dependent library version is managed by the

parent project terasoluna-gfw-parent , specifying the version in pom.xml is not necessary.

Creating a bean definition file

XML file is created as below to define a bean for the component of Spring Security. (extracted from Blank project)

- Definition example of xxx-web/src/main/resources/META-INF/spring/spring-security.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:sec="http://www.springframework.org/schema/security"
    xsi:schemaLocation="
        http://www.springframework.org/schema/security http://www.springframework.
↪org/schema/security/spring-security.xsd
        http://www.springframework.org/schema/beans http://www.springframework.org/
↪schema/beans/spring-beans.xsd
    "> <!-- (1) -->

    <sec:http pattern="/resources/**" security="none"/> <!-- (2) -->
    <sec:http> <!-- (3) -->
        <sec:form-login /> <!-- (4) -->
        <sec:logout /> <!-- (5) -->
        <sec:access-denied-handler ref="accessDeniedHandler"/> <!-- (6) -->
        <sec:custom-filter ref="userIdMDCPutFilter" after="ANONYMOUS_FILTER"/> <!--
↪ (7) -->
        <sec:session-management /> <!-- (8) -->
    </sec:http>

    <sec:authentication-manager /> <!-- (9) -->

    <!-- CSRF Protection -->
    <bean id="accessDeniedHandler"
        class="org.springframework.security.web.access.
↪DelegatingAccessDeniedHandler"> <!-- (10) -->
        <!-- omitted -->
    </bean>

    <!-- Put UserID into MDC -->
    <bean id="userIdMDCPutFilter" class="org.terasoluna.gfw.security.web.logging.
↪UserIdMDCPutFilter">  <!-- (11) -->
    </bean>

</beans>
```

Sr. No.	Description
(1)	Enable XML namespace provided by Spring Security. A name `sec` is assigned in the above example. A bean for component of Spring Security can be defined easily if XML namespace is used.
(2)	Define `<sec:http>` tag and perform settings related to the resource path for which the security countermeasures are not required, Refer *Settings not to apply security countermeasures* for details.
(3)	Define `<sec:http>` tag. A bean for the component required to use Spring Security is automatically defined if `<sec:http>` tag is defined.
(4)	Define `<sec:form-login>` tag and perform settings related to login where form authentication is used. Refer *Form authentication* for details
(5)	Define `<sec:logout>` tag and perform settings related to logout. Refer *Logout* for details.
(6)	Define `<sec:access-denied-handler>` tag and define settings to control at the time of access error. Refer *Applying AccessDeniedHandler* and *Transition destination at the time of authorization error* for details.
(7)	Define a filter for common library to store the user information to be output in a log, in MDC.
(8)	Define `<sec:session-management>` tag and perform settings related to session management. Refer *Session Management* for details
(9)	Define `<sec:authentication-manager>` tag and define a bean for component for authentication function. Error occurs at the time of starting a server if this tag is not defined.
(10)	Define a bean for the component that handles access error.
(11)	Define a bean for the component of common library to store the user information to be output in a log in MDC.

Define to generate DI container of Spring using a bean definition file thus created.

- Setup example of xxx-web/src/main/webapp/WEB-INF/web.xml

```
<!-- (1) -->
<listener>
    <listener-class>
        org.springframework.web.context.ContextLoaderListener
    </listener-class>
</listener>
<!-- (2) -->
<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>
        classpath*:META-INF/spring/applicationContext.xml
        classpath*:META-INF/spring/spring-security.xml
    </param-value>
</context-param>
```

Sr. No.	Description
(1)	Specify `ContextLoaderListener` class as the listener class of servlet container.
(2)	Add a bean definition file for Spring Security to `contextClass` parameter of servlet container besides `applicationContext.xml`.

Settings for servlet filter

Finally, the servlet filter class (`FilterChainProxy`) provided by Spring Security is registered in servlet container.

- Setup example of xxx-web/src/main/webapp/WEB-INF/web.xml

```
<!-- (1) -->
<filter>
    <filter-name>springSecurityFilterChain</filter-name>
    <filter-class>
        org.springframework.web.filter.DelegatingFilterProxy
    </filter-class>
</filter>
<!-- (2) -->
<filter-mapping>
    <filter-name>springSecurityFilterChain</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

Sr. No.	Description
(1)	Register bean (`FilterChainProxy`) managed in DI container of Spring in the servlet container using `DelegatingFilterProxy` provided by Spring Framework. Specify the name (`springSecurityFilterChain`) of bean managed in DI container of Spring in the servlet filter name.
(2)	Specify the pattern of URL where Spring Security is to be applied. Apply Spring Security to all requests in the above example.

Settings not to apply security countermeasures

The resource path (like the path to access css file or image file) for which the security countermeasures are not required, can be controlled using `<sec:http>` tag so that the security function (Security Filter) of Spring Security is not applied.

- Definition example of xxx-web/src/main/resources/META-INF/spring/spring-security.xml

```
<sec:http pattern="/resources/**" security="none"/>  <!-- (1) (2) -->
<sec:http>
    <!-- omitted -->
</sec:http>
```

Sr. No.	Description
(1)	Specify the pattern of the path where the security function is not applied in `pattern` attribute.
(2)	Specify `none` in `security` attribute. The security function (Security Filter) of Spring Security is not applied if `none` is specified.

9.2 Authentication

9.2.1 Overview

This chapter explains about Authentication function provided by Spring Security.

Authentication process checks the validity of user using the application.

The standard method to check the validity of a user is to register the users who can use the application in a data store and and verify the authentication information (user name, password etc) input by the user. Although a relational database is commonly used in a data store wherein user information is registered, directory service or external system etc are also used.

Further, there are several methods which ask a user to input authentication information. Although a method wherein input form of HTML or a method wherein an authentication method of HTTP standard defined in RFC (Basic authentication or Digest authentication etc) are used in general, the methods like OpenID authentication or single sign-on authentication are also used.

This section introduces an implementation example wherein the authentication is done by verifying authentication information input in HTML input form and user information stored in the relational database, and explains how to use authentication function of Spring Security.

Authentication process architecture

Spring Security performs authentication with the following flow.

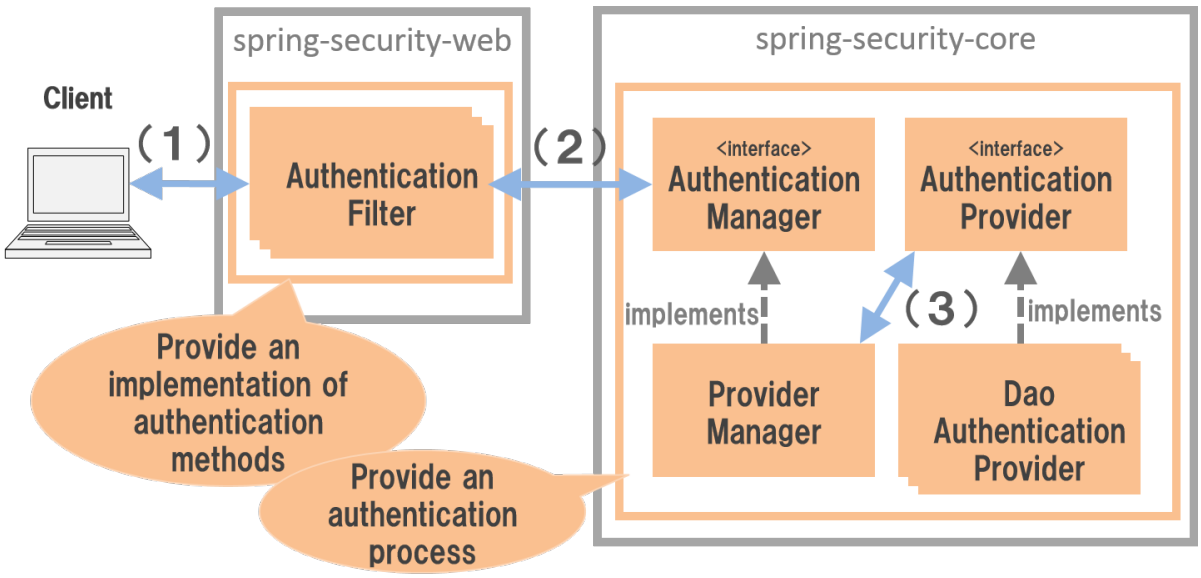


Figure.9.2 **Authentication process architecture**

Sr. No.	Description
(1)	Client specifies credentials (user name and password) for the path wherein authentication is performed and sends a request.
(2)	Authentication Filter fetches credentials from the request and calls authentication process of `AuthenticationManager` class.
(3)	`ProviderManager` (Implementation class of `AuthenticationManager` used by default) delegates actual authentication process to implementation class of `AuthenticationProvider` interface.

Authentication Filter

Authentication Filter is a servlet filter which offers implementation for the authentication method. Primary authentication methods supported by Spring Security are as given below..

Table.9.8 **Main Authentication Filter offered by Spring Security**

Class Name	Description
`UsernamePasswordAuthenticationFilter`	Fetch credentials from HTTP request parameter in servlet filter class for form authentication.
`BasicAuthenticationFilter`	Fetch credentials from authentication header of HTTP request in servlet filter class for Basic authentication.
`DigestAuthenticationFilter`	Fetch credentials from authentication header of HTTP request in servlet filter class for Digest authentication.
`RememberMeAuthenticationFilter`	Fetch credentials from Cookies of HTTP request in servlet filter class for Remember Me authentication. If Remember Me authentication is enabled, user stays logged in even if browser is closed and a session timeout has occurred.

These servlet filters are one of the Authentication filters introduced in *Framework processing*.

Note: When an authentication process not supported by Spring Security must be adopted, it must be incorporated in Spring Security after creating a `Authentication Filter` for adopting this authentication method.

AuthenticationManager

`AuthenticationManager` is an interface for implementing the authentication process. In default implementation (`ProviderManager`) offered by Spring Security, a system is adopted wherein actual authentication process is delegated to `AuthenticationProvider` and process results of authentication process performed by `AuthenticationProvider` are handled.

AuthenticationProvider

`AuthenticationProvider` is an interface which provides implementation of authentication process. A key `AuthenticationProvider` implementation class offered by Spring Security is as given below.

Table.9.9 **A key AuthenticationProvider offered by Spring Security**

Class Name	Description
`DaoAuthenticationProvider`	Implementation class which performs authentication process by checking user credentials and user status registered in the data store. Fetch credentials and user status required for checking, from the class which implements `UserDetails` interface.

Note: When an authentication process not offered by Spring Security must be adopted, it must be incorporated in Spring Security after creating an `AuthenticationProvider` for implementing authentication process.

9.2.2 How to use

Bean definition example and implementation method required for using authentication function is explained.

This section, as explained in *Overview*, explains a method to carry out authentication process by verifying credentials input in HTML input form and user information stored in the relational database.

Form authentication

Spring Security performs form authentication as per the flow given below.

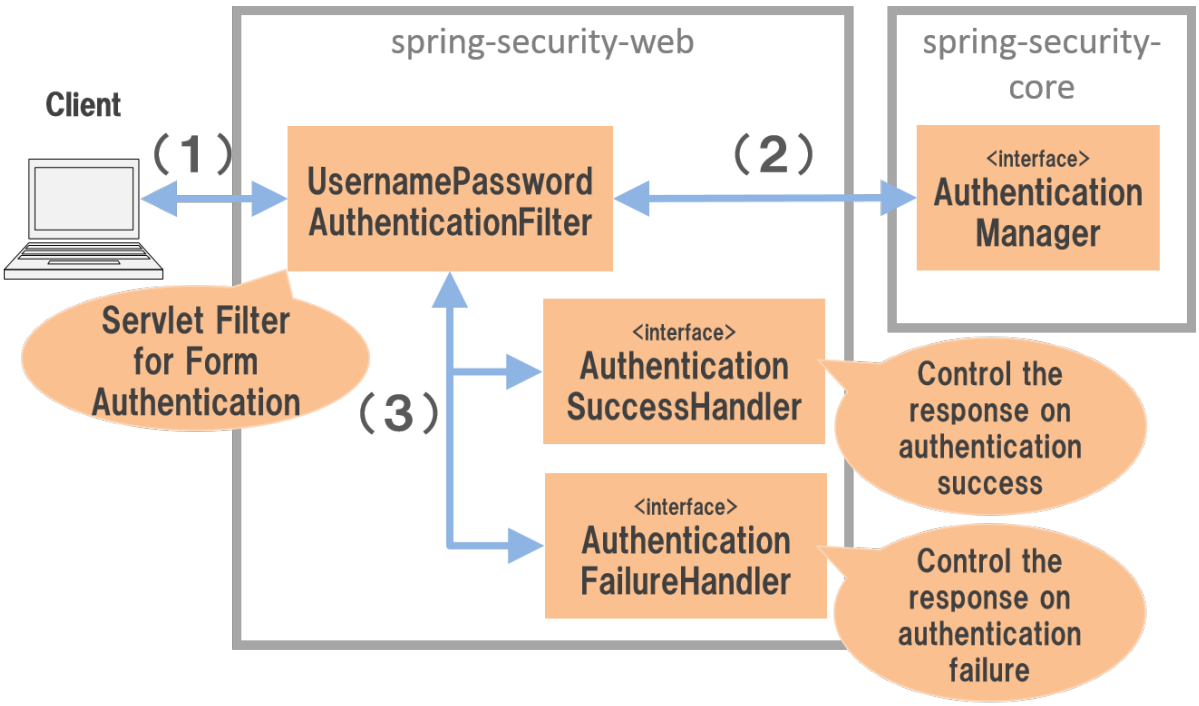


Figure.9.3 **Form authentication system**

Sr. No.	Description
(1)	Client sends credentials (user name and password) as request parameters for the path which carries out form authentication.
(2)	`UsernamePasswordAuthenticationFilter` class fetches credentials from request parameter and calls authentication process of `AuthenticationManager`.
(3)	`UsernamePasswordAuthenticationFilter` class handles the authentication results returned from `AuthenticationManager`. Call `AuthenticationSuccessHandler` method when the authentication is successful, call `AuthenticationFailureHandler` method when a failure occurs in the authentication and perform screen transition.

Applying form authentication

Bean is defined as below when form authentication is used.

- Definition example of spring-security.xml

```
<sec:http>
    <sec:form-login />     <!-- (1) -->
    <!-- omitted -->
</sec:http>
```

Sr. No.	Description
(1)	Form authentication is enabled when `<sec:form-login>` tag is defined.

Tip: About auto-config attribute

An `auto-config` attribute which specifies whether the settings for form authentication (`<sec:form-login>` tag), Basic authentication (`<sec:http-basic>` tag) and logout (`<sec:logout>` tag) are performed automatically, is provided in `<sec:http>`. Default value is `false` (not set automatically) and it is recommended to use default value even in the reference document of Spring Security.

Even in this guideline, a style wherein a tag is specified explicitly is recommended.

Element name	Description
`<form-login>`	Security Filter(`UsernamePasswordAuthenticationFilter`) which performs form authentication process is applied.
`<http-basic>`	Security Filter(`BasicAuthenticationFilter`) which performs Basic authentication in conformance with RFC1945 is applied. For detailed usage method, refer JavaDoc of BasicAuthenticationFilter.
`<logout>`	Security Filter(`LogoutFilter`) which performs logout process is applied. For details of logout process, refer "*Logout*".

Further note that when `auto-config` is not defined, form authentication (`<sec:form-login>` tag) or Basic authentication (`<sec:http-basic>` tag) must be defined. This is required to meet the specification of Spring Security wherein a Bean must be defined for one or more Authentication Filters in a single

`SecurityFilterChain(<sec:http>)`.

Default operation

If user account is accessed for `"/login"` using GET method in the default operation of Spring Security, a default login form provided by Spring Security is displayed and when login button is clicked, `"/login"` is accessed by POST method and authentication process is carried out.

Creating login form

Although a login form for form authentication is provided in Spring Security as a default, it is rarely used as it is. Below, a method wherein an auto-created login form is applied in Spring Security is explained.

At first, a JSP for displaying a login form is created. An implementation example wherein a login form is displayed after receiving a request in Spring MVC is given below.

- How to create a JSP for displaying a login form (xxx-web/src/main/webapp/WEB-INF/views/login/loginForm.jsp)

```
<%@ page contentType="text/html;charset=UTF-8" pageEncoding="UTF-8" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="sec" uri="http://www.springframework.org/security/tags" %>
<%-- omitted --%>
<div id="wrapper">
    <h3>Login Screen</h3>
    <%-- (1) --%>
    <c:if test="${param.containsKey('error')}">
        <t:messagesPanel messagesType="error"
            messagesAttributeName="SPRING_SECURITY_LAST_EXCEPTION"/> <%-- (2) --%>
    </c:if>
    <form:form action="${pageContext.request.contextPath}/login" method="post"> <%-
↪- (3) --%>
        <table>
            <tr>
                <td><label for="username">User Name</label></td>
                <td><input type="text" id="username" name="username"></td>
            </tr>
            <tr>
                <td><label for="password">Password</label></td>
                <td><input type="password" id="password" name="password"></td>
            </tr>
            <tr>
```

```
                <td> </td>
                <td><button>Login</button></td>
            </tr>
        </table>
    </form:form>
</div>
<%-- omitted --%>
```

Sr. No.	Description
(1)	An area to display authentication error.
(2)	Output an exception message which is output at the time of authentication error. It is recommended to output by using `<t:messagesPanel>` tag provided by a common library. For how to use `<t:messagesPanel>` tag, refer "*Message Management*". Note that, when an authentication error occurs, exception object is stored in the session or request scope with the attribute name `"SPRING_SECURITY_LAST_EXCEPTION"`.
(3)	Login form to enter user name and password. Send user name and password to request parameters `username` and `password` respectively. Also note that token value for CSRF measures is sent to request parameter by using `<form:form>`. CSRF measures are explained in "*CSRF Countermeasures*".

Next, login form thus created is applied to Spring Security.

- Definition example of spring-security.xml

```
<sec:http>
  <sec:form-login
      login-page="/login/loginForm"
      login-processing-url="/login" /> <!-- (1)(2) -->
  <sec:intercept-url pattern="/login/**" access="permitAll"/>  <!-- (3) -->
  <sec:intercept-url pattern="/**" access="isAuthenticated()"/> <!-- (4) -->
</sec:http>
```

Sr. No.	Description
(1)	Specify path to display login form in `login-page` attribute. When an anonymous user accesses a Web resource for which authentication is required, the user is redirected to a path specified in the attribute and a login form is displayed. Here, a request is received by Spring MVC and a login form is displayed. For details, refer "*Receive a request by Spring MVC and display login form*".
(2)	Specify path for performing authentication process in `login-processing-url` attribute. Although default path is also `"/login"`, it should be explicitly specified.
(3)	Assign the rights enabling access to all users for the location under `/login` path where login form is stored. For how to specify access policy for web resource, refer "*Authorization*".
(4)	Assign the access rights for web resource handled by the application. In the example above, only authenticated users are granted the rights to access location under root path of web application. For how to specify access policy for web resource, refer "*Authorization*".

Note: Changes in Spring Security 4.0

Default values of following configuration are changed from Spring Security version 4.0

- username-parameter
- password-parameter
- login-processing-url
- authentication-failure-url

Response when authentication is successful

Spring Security offers `AuthenticationSuccessHandler` interface and implementation class as the components to control the response when the authentication is successful.

Table.9.10 **Implementation class of AuthenticationSuccessHandler**

Implementation class	Description
`SavedRequestAwareAuthenticationSuccessHandler`	Implementation class that redirects to a URL which the user have attempted to access prior to authentication. **Implementation class used by default.**
`SimpleUrlAuthenticationSuccessHandler`	Implementation class which redirects or forwards to `defaultTargetUrl`.

Default operation

In the default operation of Spring Security, the request that was denied prior to authentication is saved in HTTP session and when the authentication is successful, the request which was denied access is restored and redirected. Page is displayed if the authenticated user has the access rights for redirected page, authentication error occurs if the user does not have the access rights. `SavedRequestAwareAuthenticationSuccessHandler` class is used to carry out this operation.

Since the transition destination after explicitly displaying login form and performing authentication is the root path of web application (`"/"`), as per default configuration of Spring Security, user is redirected to the root path of Web application when the authentication is successful.

Response when authentication fails

Spring Security offers `AuthenticationFailureHandler` interface and implementation class as the components to control the response when the authentication fails.

Table.9.11 **Implementation class of AuthenticationFailureHandler**

Implementation class	Description
`SimpleUrlAuthenticationFailureHandler`	Implementation class which redirects or forwards to a specified path (`defaultFailureUrl`).
`ExceptionMappingAuthenticationFailureHandler`	Implementation class which can map authentication exception and URL for transition. Since the exception class generated by Spring Security for each error cause vary, the transition destination can be changed for each error type if this implementation class is used.
`DelegatingAuthenticationFailureHandler`	Implementation class which can map authentication exception and `AuthenticationFailureHandler`. Although it resembles `ExceptionMappingAuthenticationFailureHandler`, a flexible behaviour can be supported by specifying `AuthenticationFailureHandler` for each authentication exception.

Default operation

In the default operation of Spring Security, user is redirected to a URL assigned with a query parameter `"error"`in a path which displays login form.

As an example, when the path to display login form is `"/login"`, the user is redirected to `"/login?error"`.

DB authentication

Spring Security performs DB authentication in the flow given below.

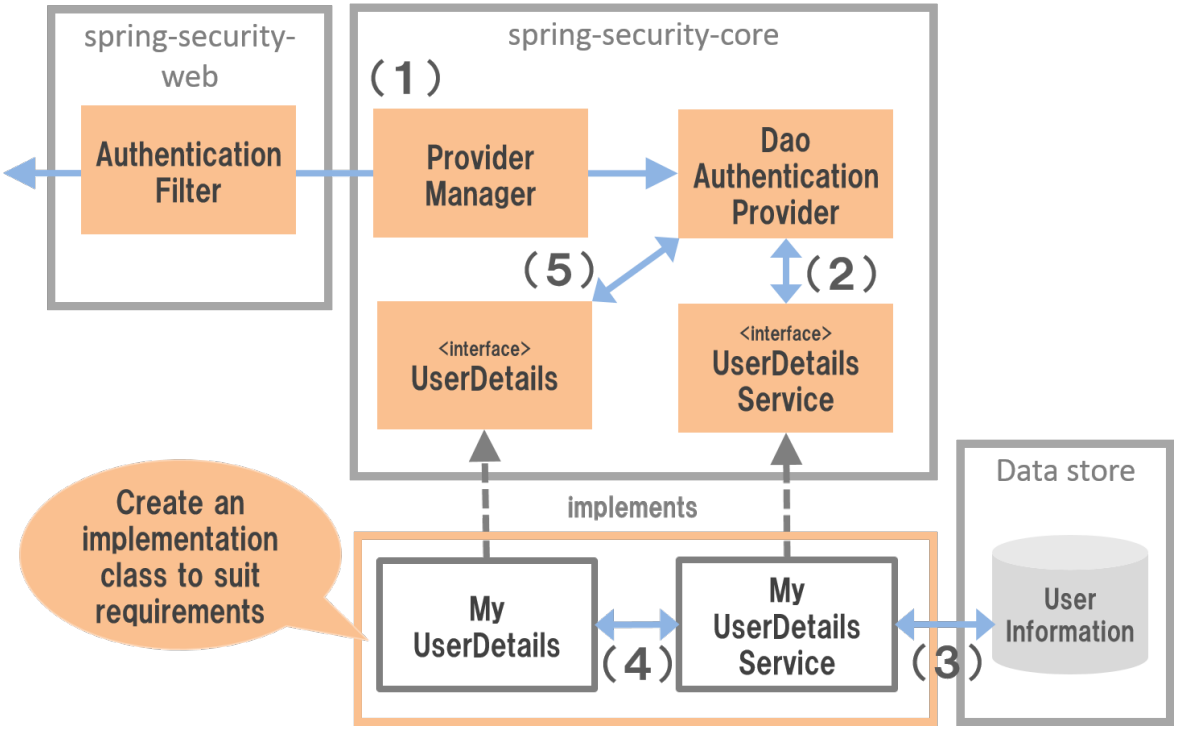


Figure.9.4 **DB authentication system**

Sr. No.	Description
(1)	Spring Security receives authentication request from the client and calls authentication process of `DaoAuthenticationProvider`.
(2)	`DaoAuthenticationProvider` calls user information fetch process of `UserDetailsService`.
(3)	Implementation class of `UserDetailsService` fetches user information from data store.
(4)	Implementation class of `UserDetailsService` generates `UserDetails` from the user information fetched from data store.

次のページに続く

Table. 9.12 – 前のページからの続き

Sr. No.	Description
(5)	`DaoAuthenticationProvider` verifies `UserDetails` returned from `UserDetailsService` and authentication information specified by client, and checks validity of user specified by client.

Note: DB authentication offered by Spring Security

Spring Security offers an implementation class to fetch user information from relational database through JDBC.

- `org.springframework.security.core.userdetails.User` (Implementation class of `UserDetails`)
- `org.springframework.security.core.userdetails.jdbc.JdbcDaoImpl` (Implementation class of `UserDetailsService`)

Since these implementation classes only perform authentication processes of minimum level (password verification, determination of user validity), they are used as it is very rarely. Hence, this guideline explains how to create implementation classes of `UserDetails` and `UserDetailsService`.

Creating UserDetails

`UserDetails` is an interface which provides credentials (user name and password) and user status necessary in the authentication process and defines following methods. When `DaoAuthenticationProvider` is used as `AuthenticationProvider`, implementation class of `UserDetails` is created in accordance with the requirements of application.

UserDetails interface

```
public interface UserDetails extends Serializable {
    String getUsername(); // (1)
    String getPassword(); // (2)
    boolean isEnabled(); // (3)
    boolean isAccountNonLocked(); // (4)
    boolean isAccountNonExpired(); // (5)
    boolean isCredentialsNonExpired(); // (6)
    Collection<? extends GrantedAuthority> getAuthorities(); // (7)
}
```

Sr. No.	Method name	Description
(1)	`getUsername`	Return user name.

次のページに続く

Table. 9.13 – 前のページからの続き

Sr. No.	Method name	Description
(2)	`getPassword`	Return registered password. When the password returned by this method and the password specified by the client do not match, `DaoAuthenticationProvider` throws `BadCredentialsException`.
(3)	`isEnabled`	Determine whether the user is valid. Return `true` in case of valid user. When the user is invalid, `DaoAuthenticationProvider` throws `DisabledException`.
(4)	`isAccountNonLocked`	Determine locked status of the account. When the account is not locked, return `true`. When the account is locked, `DaoAuthenticationProvider` throws `LockedException`.
(5)	`isAccountNonExpired`	Determine expiry status of the account. Return `true` when the account is not expired. When the account is expired, `DaoAuthenticationProvider` throws `AccountExpiredException`.
(6)	`isCredentialsNonExpired`	Determine expiry status of credentials. Return `true` when the credentials are not expired. When the credentials are expired, `DaoAuthenticationProvider` throws `CredentialsExpiredException`.
(7)	`getAuthorities`	Return list of rights assigned to the user. This method is used in the authorization process.

Note: Changing transition destination based on authentication exception

When the user wants to change the screen transition for each exception thrown by `DaoAuthenticationProvider`, `ExceptionMappingAuthenticationFailureHandler` must be used as `AuthenticationFailureHandler`.

As an example, when user wants to transit to password change screen in case of password expiry for the user, screen transition can be changed by handling `CredentialsExpiredException` by using `ExceptionMappingAuthenticationFailureHandler`.

For details, refer *Customising response at the time of authentication failure*.

Note: Credentials provided by Spring Security

Although Spring Security provides an implementation class (`org.springframework.security.core.userdetails.Us` for retaining credentials (user name and password) and user status, it can store only the information required for the authentication process. Since the information of the user (name of the user etc) which is not used in the authentication process is frequently required in the general applications, `User` class can rarely be used as it is.

Here, an implementation class of `UserDetails` which retains account information is created. This example can also be implemented by inheriting `User`. However, here an example wherein `UserDetails` is implemented is introduced.

- How to create implementation class of UserDetails

```
public class AccountUserDetails implements UserDetails { // (1)

    private final Account account;
    private final Collection<GrantedAuthority> authorities;

    public AccountUserDetails(
        Account account, Collection<GrantedAuthority> authorities) {
        // (2)
        this.account = account;
        this.authorities = authorities;
    }

    // (3)
    public String getPassword() {
```

```
        return account.getPassword();
    }
    public String getUsername() {
        return account.getUsername();
    }
    public boolean isEnabled() {
        return account.isEnabled();
    }
    public Collection<GrantedAuthority> getAuthorities() {
        return authorities;
    }

    // (4)
    public boolean isAccountNonExpired() {
        return true;
    }
    public boolean isAccountNonLocked() {
        return true;
    }
    public boolean isCredentialsNonExpired() {
        return true;
    }

    // (5)
    public Account getAccount() {
        return account;
    }

}
```

Sr. No.	Description
(1)	Create a class which implements `UserDetails` interface.
(2)	Retain user information and rights information in the property.
(3)	Implement method defined in `UserDetails` interface.
(4)	Although the example of this sections does not implement check for, “Account locking”, “Account expiry” and “Credentials expiry”, these must be checked in accordance with the requirements.
(5)	Provide a getter method to enable access to account information in the process after successful completion of authentication.

Spring Security provides `User` class as an implementation class of `UserDetails`. When you inherit a `User` class, credentials and user status can be easily retained.

- How to create UserDetails implementation class which inherits User class

```
public class AccountUserDetails extends User {

    private final Account account;

    public AccountUserDetails(Account account, boolean accountNonExpired,
            boolean credentialsNonExpired, boolean accountNonLocked,
            Collection<GrantedAuthority> authorities) {
        super(account.getUsername(), account.getPassword(),
                account.isEnabled(), true, true, true, authorities);
        this.account = account;
    }

    public Account getAccount() {
        return account;
    }
```

```
}
```

Creating UserDetailsService

`UserDetailsService` is an interface to fetch credentials and user status required for authentication process from data store and defines following methods. When `DaoAuthenticationProvider` is used as `AuthenticationProvider`, implementation class of `UserDetailsService` is created in accordance with the requirements of the application.

- UserDetailsService interface

```
public interface UserDetailsService {
    UserDetails loadUserByUsername(String username) throws␣
↪UsernameNotFoundException;
}
```

Here, a service class is created to search account information from the database and generate an instance of `UserDetails`. In this sample, account information is fetched by using `SharedService`. For `SharedService`, refer *Implementation of Service*.

- How to create AccountSharedService interface

```
public interface AccountSharedService {
    Account findOne(String username);
}
```

- How to create implementation class of AccountSharedService

```
// (1)
@Service
@Transactional
public class AccountSharedServiceImpl implements AccountSharedService {
    @Inject
    AccountRepository accountRepository;

    // (2)
    @Override
```

```
    public Account findOne(String username) {
        Account account = accountRepository.findOneByUsername(username);
        if (account == null) {
            throw new ResourceNotFoundException("The given account is not found!␣
↪username="
                    + username);
        }
        return account;
    }
}
```

Sr. No.	Description
(1)	Create a class which implements `AccountSharedService` interface and assign `@Service`. In the example above, `AccountSharedServiceImpl` is registered in DI container using component scan function.
(2)	Search account information from the database. When account information is not found, an exception of common library - `ResourceNotFoundException` is thrown. For how to create a repository, refer "*Spring Security Tutorial*".

- How to create implementation class of UserDetailsService

```
// (1)
@Service
@Transactional
public class AccountUserDetailsService implements UserDetailsService {
    @Inject
    AccountSharedService accountSharedService;

    public UserDetails loadUserByUsername(String username)
            throws UsernameNotFoundException {

        try {
            Account account = accountSharedService.findOne(username);
            // (2)
            return new AccountUserDetails(account, getAuthorities(account));
        } catch (ResourceNotFoundException e) {
            // (3)
            throw new UsernameNotFoundException("user not found", e);
        }
    }
```

```
    // (4)
    private Collection<GrantedAuthority> getAuthorities(Account account) {
        if (account.isAdmin()) {
            return AuthorityUtils.createAuthorityList("ROLE_USER", "ROLE_ADMIN");
        } else {
            return AuthorityUtils.createAuthorityList("ROLE_USER");
        }
    }
}
```

Sr. No.	Description
(1)	Create a class which implements `UserDetailsService` interface and assign `@Service`. In the example above, `UserDetailsService` is registered in DI container using a component scan function.
(2)	Fetch account information by using `AccountSharedService`. When the account information is found, `UserDetails` is generated. In the example above, user name, password and user validity status are fetched from account information.
(3)	When the account information is not found, `UsernameNotFoundException` is thrown.
(4)	Generate information about the rights (role) retained by the user. The rights (role) generated here are used in the authorization process.

Note: Information of rights to be used in the authorization

Authorization process of Spring Security handles the rights information starting with `"ROLE_"`as a role. Hence, when resource access is to be controlled by using role, `"ROLE_"` must be assigned as a prefix to the rights information which is being handled as a role.

Note: Concealing authentication exception information

In the default operation of Spring Security, error handling is performed after changing `UsernameNotFoundException` to an exception called `BadCredentialsException`.

`BadCredentialsException` denotes an error in any field of the credentials specified by the client, specific error causes are not notified to the client.

Applying DB authentication

When the authentication is to be performed by using `UserDetailsService` thus created, `UserDetailsService` thus created must be applied by enabling `DaoAuthenticationProvider`.

- Definition example of spring-security.xml

```
<sec:authentication-manager> <!-- (1) -->
    <sec:authentication-provider user-service-ref="accountUserDetailsService"> <!--
↪ (2) -->
        <sec:password-encoder ref="passwordEncoder" /> <!-- (3) -->
    </sec:authentication-provider>
</sec:authentication-manager>

<bean id="passwordEncoder"
    class="org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder" /> <!-
↪- (4) -->
```

Sr. No.	Description
(1)	Define a Bean for `AuthenticationManager`.
(2)	Define `<sec:authentication-provider>` element in the `<sec:authentication-manager>` element. Specify a Bean of `AccountUserDetailsService` created by "*Creating UserDetailsService*", in `user-service-ref` attribute. Based on this definition, `DaoAuthenticationProvider` of default setup is enabled.
(3)	Specify a Bean of `PasswordEncoder` to be used at the time of password verification.
(4)	Define a Bean for `PasswordEncoder` to be used at the time of password verification. In the example above, `BCryptPasswordEncoder` is defined which performs password hashing by using BCrypt algorithm. For password hashing, refer "*Password hashing*".

Password hashing

When a password is stored in the database, password is not stored as it is and a hash value of the password is generally stored.

Spring Security provides an interface and implementation class for password hashing and operates in coordination with the authentication function.

There are 2 types of interfaces provided by Spring Security.

- `org.springframework.security.crypto.password.PasswordEncoder`
- `org.springframework.security.authentication.encoding.PasswordEncoder`

Although both the interfaces include `PasswordEncoder`, `PasswordEncoder` of `org.springframework.security.authentication.encoding` package is deprecated.

When there are no specific constraints in the password hashing requirements, using the implementation class of

`PasswordEncoder` interface of `org.springframework.security.crypto.password` package is recommended.

Note: For how to use deprecated `PasswordEncoder`, refer "*Using PasswordEncoder of deprecated package*".

Definition of a method for org.springframework.security.crypto.password.PasswordEncoder

```
public interface PasswordEncoder {
    String encode(CharSequence rawPassword);
    boolean matches(CharSequence rawPassword, String encodedPassword);
}
```

Table.9.14 **Method defined in PasswordEncoder**

Method name	Description
`encode`	A method for password hashing. It can be used for hashing of passwords which are stored in the data store during account registration process and password change process.
`matches`	A method to verify plain text password and hashed password. This method can also be used in the authentication process of Spring Security and can also be used to verify current password and the password used earlier during password change process etc.

Spring Security offers following classes as the implementation class of `PasswordEncoder` interface.

Table.9.15 **Implementation class of PasswordEncoder**

Implementation class	Description
`BCryptPasswordEncoder`	Implementation class which performs password hashing and verification by using BCrypt algorithm. **When there are no specific constraints in password hashing requirement, using this class is recommended.** For details, refer JavaDoc of BCryptPasswordEncoder.
`StandardPasswordEncoder`	Implementation class which performs password hashing and verification by using SHA-256 algorithm. For details, refer JavaDoc of StandardPasswordEncoder.
`NoOpPasswordEncoder`	Implementation class which does not perform hashing. It is a class used for testing and is not used in the actual application.

This section explains how to use `BCryptPasswordEncoder`that has been recommended by Spring Security.

BCryptPasswordEncoder

`BCryptPasswordEncoder` is an implementation class which performs password hashing and password verification by using BCrypt algorithm. 16 byte random numbers (`java.security.SecureRandom`) are used in *Salt*, *Stretching* is done for 1024 (2 to the power of 10) times by default..

- Definition example of applicationContext.xml

```
<bean id="passwordEncoder"
    class="org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder" > <!--
↪ (1) -->
    <constructor-arg name="strength" value="11" /> <!-- (2) -->
</bean>
```

Sr. No.	Description
(1)	Specify `BCryptPasswordEncoder` in passwordEncoder class.
(2)	Specify number of rounds for stretching count of hashing in the argument of constructor. This argument can be omitted and the values that can be specified are in the range from `4` to `31`. Note that default value is `10` when the argument value is not specified. The explanation is omitted in the guideline, however `java.security.SecureRandom.SecureRandom` can also be specified as a constructor argument.

Warning: How to use SecureRandom

When `SecureRandom` is to be used in Linux environment, a delay or timeout in the processing is likely to occur. This event depends on the random number generator to be used and description is given in the document given below.

- https://docs.oracle.com/javase/8/docs/api/java/security/SecureRandom.html

When this event occurs, it can be avoided by adding one of the following settings.

- Specify `-Djava.security.egd=file:/dev/urandom` while executing Java command.
- Change `securerandom.source=/dev/random` in ``${JAVA_HOME}/jre/lib/security/java.sec`` to `securerandom.source=/dev/urandom`.

If this event occurs in the version prior to b19 of Java SE 7 (prior to official release), `/dev/./urandom` must be specified instead of `/dev/urandom`. However, algorithm used by `SecureRandom` cannot be avoided in case of `NativePRNG`.

`PasswordEncoder` is used in the class which carries out a process by using `BCryptPasswordEncoder`, by injecting from DI container.

```
@Service
@Transactional
public class AccountServiceImpl implements AccountService {
```

```
    @Inject
    AccountRepository accountRepository;

    @Inject
    PasswordEncoder passwordEncoder; // (1)

    public Account register(Account account, String rawPassword) {
        // omitted
        String encodedPassword = passwordEncoder.encode(rawPassword); // (2)
        account.setPassword(encodedPassword);
        // omitted
        return accountRepository.save(account);
    }

}
```

Sr. No.	Description
(1)	Inject `PasswordEncoder`.
(2)	Call injected `PasswordEncoder` method. Here, the password stored in the data store is hashed.

Note: Salt

A salt is a string which is added to data targeted for hashing. Since number of digits exceed the actual password length by assigning a salt to a password, password analysis using programs like rainbow crack become difficult. Note that, **it is recommended to set different values of salt (random values) for each user.** This is necessary because if identical salt value is used, the string (password) before hashing can be easily identified from the hash value.

Note: Stretching

Information related to stored password is repeatedly encrypted by iterating hash function calculation in order to extend the time required for password analysis as a countermeasure against password brute force attack. However, stretching frequency must be determined considering the system performance since stretching is likely to impact system performance.

In Spring Security, stretching is done for 1024 times (2 to the power of 10) by default, however this frequency can be changed by constructor argument (`strength`). A value from 4 (16 times) to 31 (2, 147, 483, 648 times) can be specified in the `strength`. More the frequency of stretching, stronger the password. However, it is likely to

impact performance due to higher complexity.

Handling authentication event

Spring Security offers a system which links process results of authentication process with other components using an event notification system offered by Spring Security.

If this system is used, following security requirements can be incorporated in the authentication function of Spring Security.

- Save authentication history of successful and failed authentication in database or log.
- Lock account if wrong password is entered in succession.

Authentication event is notified as given below.

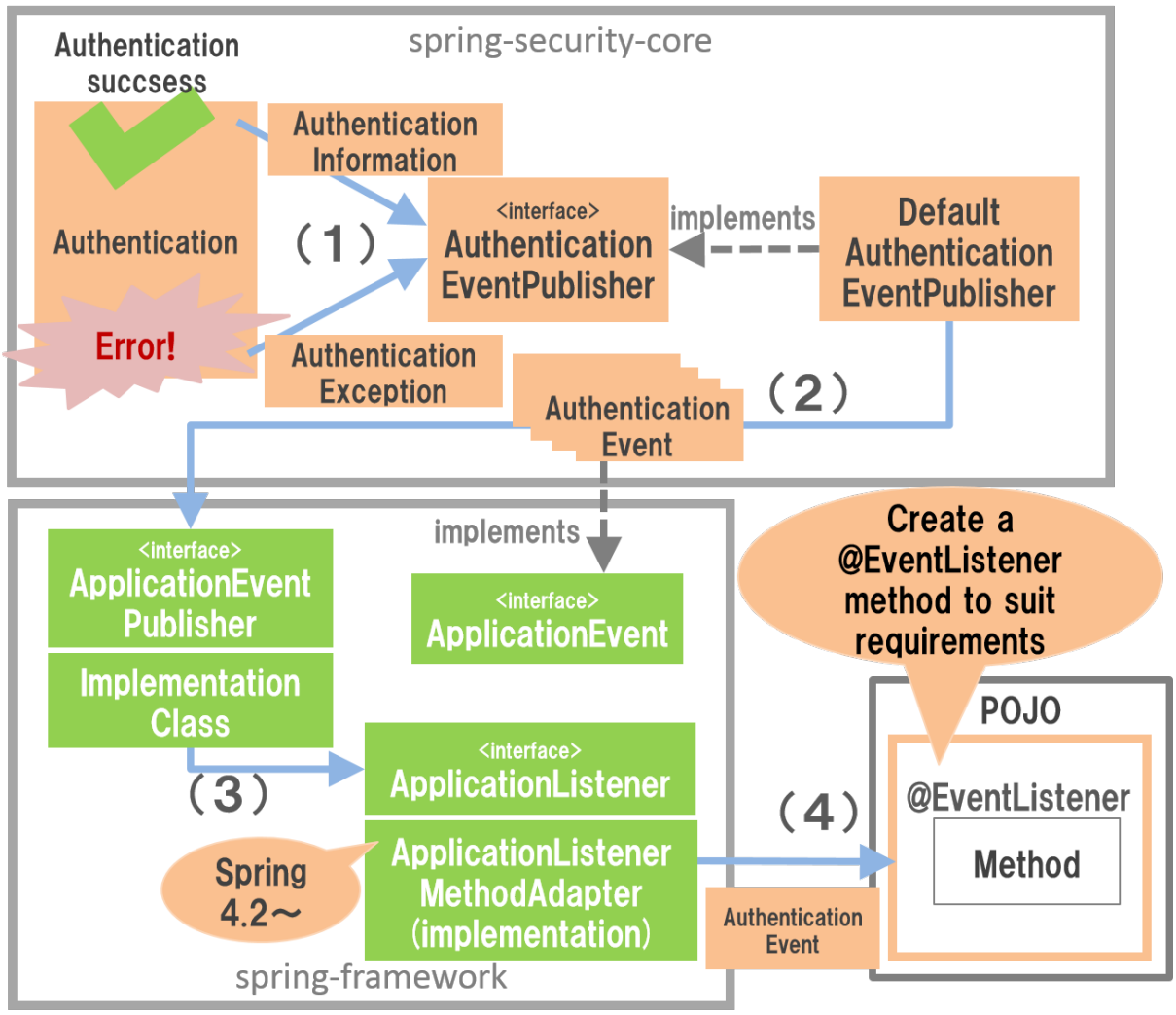


Figure.9.5 **Event notification system**

Sr. No.	Description
(1)	Spring Security authentication function passes authentication results (authentication information and authentication exception) in `AuthenticationEventPublisher` and requests notification of authentication event.
(2)	Default implementation class of `AuthenticationEventPublisher` interface generates an instance of authentication event class corresponding to authentication results, passes to `ApplicationEventPublisher` and requests event notification.
(3)	Implementation class of `ApplicationEventPublisher` interface notifies the event to implementation class of `ApplicationListener` interface.
(4)	`ApplicationListenerMethodAdaptor`, one of the implementation classes of `ApplicationListener` calls a method

Note: Memo

Although it was necessary to receive the event by creating an implementation class of `ApplicationListener`till Spring 4.1, an event can be created in Spring 4.2 only by implementing a method which assigns `@EventListener` to POJO. Note that, even in subsequent versions of Spring 4.2, an event can be received by creating an implementation class of `ApplicationListener` interface as earlier.

Events used by Spring Security can be classified into 2 types - the events which notify successful authentication and the events which notify failure in authentication. Event class offered by Spring Security are explained below.

Authentication successful event

There are 3 main events which are notified by Spring Security at the time of successful authentication. These events are notified in the following order unless an error occurs in between.

Table.9.16 **Event class which notifies that the authentication is successful**

Event class	Description
`AuthenticationSuccessEvent`	An event class to notify that the authentication process by `AuthenticationProvider` is successful. If this event is handled, it can be detected whether the client has specified correct information. It must be noted that an error is likely to occur in the subsequent processes after handling this event.
`SessionFixationProtectionEvent`	An event class to notify that session fixation attack countermeasure process (change process of session ID) is successful. If this event is handled, session ID after the change can be detected.
`InteractiveAuthenticationSuccessEvent`	An event class to notify that authentication process is entirely successful. If this event is handled, it can be detected that the overall authentication process is successful except the screen transition.

Authentication failure event

Main events which are notified by Spring Security at the time of authentication failure are as below. When authentication fails, any one of the events is notified.

Table.9.17 **Event class which notifies that authentication has failed**

Event class	Description
`AuthenticationFailureBadCredentialsEvent`	Event class which notifies that `BadCredentialsException` has occurred.
`AuthenticationFailureDisabledEvent`	Event class which notifies that `DisabledException` has occurred.
`AuthenticationFailureLockedEvent`	Event class which notifies that `LockedException` has occurred.
`AuthenticationFailureExpiredEvent`	Event class which notifies that `AccountExpiredException` has occurred.
`AuthenticationFailureCredentialsExpiredEvent`	Event class which notifies that `CredentialsExpiredException` has occurred.
`AuthenticationFailureServiceExceptionEvent`	Event class which notifies that `AuthenticationServiceException` has occurred.

Creating event listener

When you want to carry out a process by receiving authentication event notification, a class implementing a method that assigns `@EventListener` is created and registered in DI container.

- Implementation example for event listener class

```
@Component
public class AuthenticationEventListeners {

    private static final Logger log =
            LoggerFactory.getLogger(AuthenticationEventListeners.class);

@EventListener // (1)
public void handleBadCredentials(
    AuthenticationFailureBadCredentialsEvent event) { // (2)
    log.info("Bad credentials is detected. username : {}", event.
↪getAuthentication().getName());
    // omitted
}
```

Sr. No.	Description
(1)	Create a method which assigns `@EventListener` to a method.
(2)	Specify an authentication event class which is to be handled in the method argument.

The example above is an example wherein a class is created that handles `AuthenticationFailureBadCredentialsEvent` that is notified when the client specified authentication information is incorrect. Other events can also be handled in the same manner.

Logout

Spring Security performs logout process as per the flow given below.

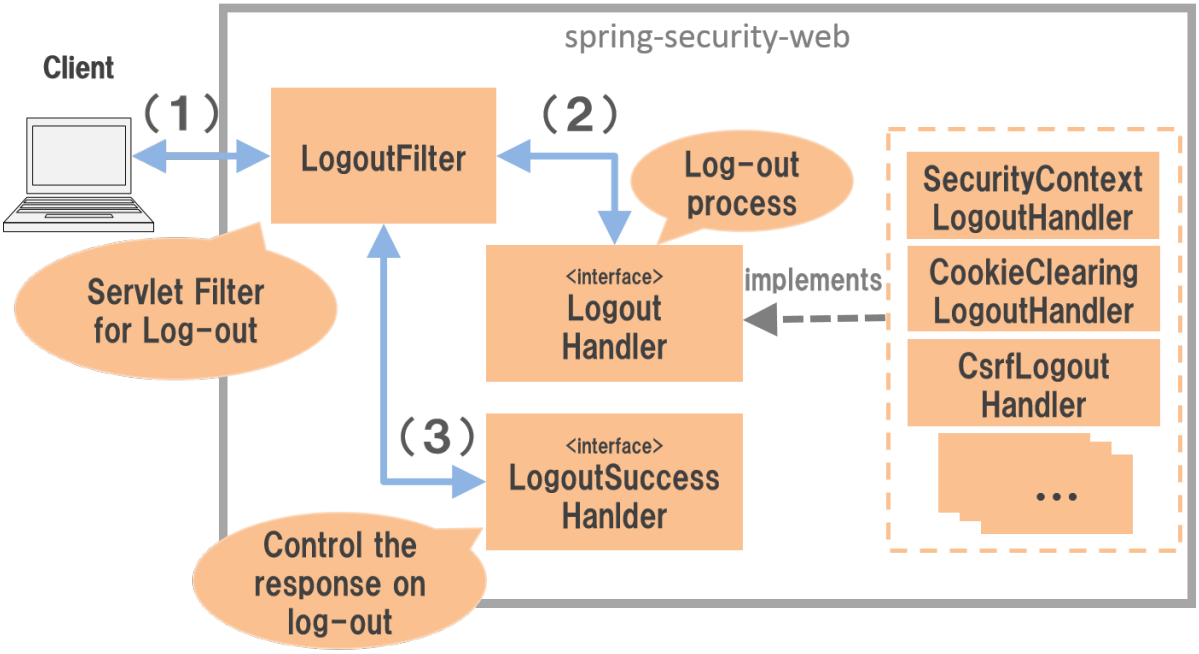


Figure.9.6 **Logout process system**

Sr. No.	Description
(1)	Client sends a request to a path intended for logout process.
(2)	`LogoutFilter` calls `LogoutHandler` method and performs actual logout process.
(3)	`LogoutFilter` calls `LogoutSuccessHandler` method and performs screen transition.

A plurality of implementation classes of `LogoutHandler` play respective roles as below.

Table.9.18 **Implementation class of main LogoutHandler**

Implementation class	Description
`SecurityContextLogoutHandler`	Class which clears authentication information of login user and cancels a session.
`CookieClearingLogoutHandler`	Class which sends a response for deleting specific cookies.
`CsrfLogoutHandler`	Class which deletes a token for CSRF countermeasure.

Since `LogoutHandler`automatically sets the class which supports bean definition offered by Spring Security, in `LogoutFilter`, there is no need for a developer to be specifically aware of the same. Note that, if *Remember Me authentication function* is enabled, implementation class of `LogoutHandler` is also configured to delete the token for Remember Me authentication.

Applying logout process

Define a bean as below for application of logout process.

- Definition example of spring-security.xml

```
<sec:http>
    <!-- omitted -->
    <sec:logout /> <!-- (1) -->
    <!-- omitted -->
</sec:http>
```

Sr. No.	Description
(1)	Logout process is enabled by defining `<sec:logout>` tag.

Note: Changes in Spring Security 4.0

Default values of following configuration changes from Spring Security 4.0 version.

- logout-url

Tip: Delete Cookie

Although description is omitted in the guideline, note that `delete-cookies` attribute exists in `<sec:logout>` tag for deleting Cookie specified at the time of logout. However, it has been reported that a cookie is sometimes not deleted even after using this attribute.

For details, refer following JIRA of Spring Security.

- https://jira.spring.io/browse/SEC-2091

Default operation

In the default operation of Spring Security, if a request is sent to the path `"/logout"`, logout process is carried out. Logout process consists of "clearing authentication information of logged in user" and "cancelling a session".

Further,

- "Deleting token for CSRF measures" when CSRF measures are adopted
- "Deleting token for Remember Me authentication" when Remember Me authentication function is used

are also carried out.

- Implementation example of JSP for calling logout process

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="sec" uri="http://www.springframework.org/security/tags" %>
<%-- omitted --%>
<form:form action="${pageContext.request.contextPath}/logout" method="post"> <%--␣
↪(1) --%>
    <button>Logout</button>
</form:form>
```

Sr. No.	Description
(1)	Create a form for logout. Further, token value of CSRF measure is sent in request parameter by using `<form:form>`. CSRF measures are explained in "*CSRF Countermeasures*".

Note: Sending CSRF tokens

If CSRF measures are enabled, tokens for CSRF measures must be sent by using POST method.

Response when logout is successful

Spring Security offers an interface `LogoutSuccessHandler`and implementation class as the components to control response when the logout is successfully completed.

Table.9.19 **Implementation class of AuthenticationFailureHandler**

Implementation class	Description
`SimpleUrlLogoutSuccessHandler`	Implementation class which redirects in the specified path (`defaultTargetUrl`).

Default operation

In default operation of Spring Security, user is redirected to a URL wherein a query parameter called `"logout"`is assigned to the path for displaying login form.

As an example, when the path to display login form is `"/login"`, the user is redirected to `"/login?logout"`.

Accessing authentication information

Authentication information of the authenticated user is stored in the session during default implementation of Spring Security. Authentication information stored in the session is stored in `SecurityContextHolder` class by `SecurityContextPersistenceFilter` class for each request and can be accessed from anywhere if it is in the same thread.

A method wherein `UserDetails` are fetched from authentication information and the information retained in the fetched `UserDetails` is accessed.

Access from Java

An audit log which records information like "When", "Who", "Which data" and "Type of access" is fetched in the general business application. "Who" for fulfilling these requirements can be fetched from authentication information.

- Implementation example to access authentication information from Java

```
Authentication authentication =
        SecurityContextHolder.getContext().getAuthentication(); // (1)
String userUuid = null;
if (authentication.getPrincipal() instanceof AccountUserDetails) {
    AccountUserDetails userDetails =
            AccountUserDetails.class.cast(authentication.getPrincipal()); // (2)
    userUuid = userDetails.getAccount().getUserUuid(); // (3)
}
if (log.isInfoEnabled()) {
    log.info("type:Audit\tuserUuid:{}\tresource:{}\tmethod:{}",
            userUuid, httpRequest.getRequestURI(), httpRequest.getMethod());
}
```

Sr. No.	Description
(1)	Fetch authentication information (`Authentication` object) from `SecurityContextHolder`.
(2)	Call `Authentication#getPrincipal()` method and fetch `UserDetails` object. When there is no authentication (anonymous user), note that a string indicating an anonymous user is returned.
(3)	Fetch information required for processing from `UserDetails`. Here, a value which uniquely identifies the user (UUID) is fetched.

Warning: Access and coupling of authentication information

In the default operation of Spring Security, since authentication information is stored in ThreadLocal variable, it can be accessed from anywhere if the thread is identical to a thread which receives request. Although this system is convenient, it results in direct dependency of the class which requires authentication information on `SecurityContextHolder` class. Hence, care must be taken since it reduces coupling between components when not used correctly.

Spring Security also offers a system to maintain coupling between components by coordinating with Spring MVC function. How to coordinate with Spring MVC is explained in "*Coordination between authentication process and Spring MVC*". **This guideline recommends fetching authentication information by using coordination with Spring MVC.**

Note: When a filter (FORM_LOGIN_FILTER) for authentication is to be customized, it is necessary to disable the following 2 `SessionAuthenticationStrategy` classes, apart from specifying `<sec:concurrency-control>` element.

- `org.springframework.security.web.authentication.session.ConcurrentSessionControlAut`
  A class to check number of sessions for each logged in user after successful authentication.
- `org.springframework.security.web.authentication.session.RegisterSessionAuthenticati`
  A class to register a session with successful authentication, in session management area.

In version 1.0.x.RELEASE dependent Spring Security 3.1, `org.springframework.security.web.authentication.s` class is provided; however, it is deprecated API from Spring Security 3.2 and it is abolished API from Spring Security 4.0. When upgrading version from Spring Security 3.1 to Spring Security 3.2 or later versions, changes need to be made so that it can be used with combination of following classes.

- `ConcurrentSessionControlAuthenticationStrategy` (added in Spring Security 3.2)
- `RegisterSessionAuthenticationStrategy` (added in Spring Security 3.2)
- `org.springframework.security.web.authentication.session.SessionFixationProtectionSt`

For specific methods of definition, refer to sample code of Spring Security Reference -Web Application Security (Concurrency Control)-.

Access from JSP

User information of login user is displayed on the screen in general Web application. User information of login user while meeting these requirements can be fetched from authentication information.

- Implementation example for accessing authentication information from JSP

```
<%@ taglib prefix="sec" uri="http://www.springframework.org/security/tags" %>
<%-- omitted --%>
Welcome、
<sec:authentication property="principal.account.lastName"/> <%-- (1) --%>
San
```

Sr. No.	Description
(1)	Use <sec:authentication> tag offered by Spring Security and fetch authentication information (Authentication object). Specify path for the property to be accessed in property attribute. When a nested object is to be accessed, property name should be joined by ".".

Tip: How to display authentication information

Although the implementation example for displaying user information which contains authentication information is explained, it is possible to store value in any scope variable by combining var attribute and scope attribute. When the display contents are to be changed according to the login user status, user information can be stored in the variable and display can be changed by using JSTL tag library etc.

In the example above, it can also be displayed as described below. Since scope attribute is omitted in this example, page scope is applied.

```
<%@ taglib prefix="sec" uri="http://www.springframework.org/security/tags" %>
<%-- omitted --%>
<sec:authentication var="principal" property="principal"/>
<%-- omitted --%>
Welcome
${f:h(principal.account.lastName)}
San
```

Coordination between authentication process and Spring MVC

Spring Security offers various components for coordinating with Spring MVC. How to use the components for coordinating with authentication process is explained.

Accessing authentication information

Spring Security offers `AuthenticationPrincipalArgumentResolver` class as a component which delivers authentication information (`UserDetails`) to Spring MVC controller method. If `AuthenticationPrincipalArgumentResolver` is used, `UserDetails` interface or instance of its implementation class can be received as a method argument of controller resulting in enhanced coupling of components.

`AuthenticationPrincipalArgumentResolver` must be applied to Spring MVC first for receiving Authentication information (`UserDetails`) as a controller argument. Bean for applying `AuthenticationPrincipalArgumentResolver` is as below. Note that `AuthenticationPrincipalArgumentResolver` is already cong=figured in the blank project.

- Definition example of spring-mvc.xml

```
  <mvc:annotation-driven>
      <mvc:argument-resolvers>
          <!-- omitted -->
          <!-- (1) -->
          <bean class="org.springframework.security.web.method.annotation.
↪AuthenticationPrincipalArgumentResolver" />
          <!-- omitted -->
      </mvc:argument-resolvers>
</mvc:annotation-driven>
```

Sr. No.	Description
(1)	Apply `AuthenticationPrincipalArgumentResolver` in Spring MVC as an implementation class of `HandlerMethodArgumentResolver`.

Following method is created while receiving authentication information (`UserDetails`) by controller method.

- How to create a method which receives authentication information (UserDetails)

```
@RequestMapping("account")
@Controller
public class AccountController {

    public String view(
            @AuthenticationPrincipal AccountUserDetails userDetails, // (1)
            Model model) {
        model.addAttribute(userDetails.getAccount());
```

```
        return "profile";
    }

}
```

Sr. No.	Description
(1)	Declare an argument to receive authentication information (`UserDetails`) and specify `@org.springframework.security.core.annotation.AuthenticationPrincipal` as an argument annotation. `AuthenticationPrincipalArgumentResolver` configures authentication information (`UserDetails`) in the argument assigned by `@AuthenticationPrincipal`.

9.2.3 How to extend

The section describes customization points and extension methods offered by Spring Security.

Spring Security offers a lot of customization points. Since it is not possible to introduce all the customization points here, we will focus on a few typical customization points here.

Customizing form authentication

Customization points of form authentication process are explained.

Changing authentication path

In Spring Security, the path for carrying out authentication process is "`"/login"`" by default, however it can be changed by defining a bean as below.

- Definition example of spring-security.xml

```
<sec:http>
  <sec:form-login login-processing-url="/authentication" /> <!-- (1) -->
  <!-- omitted -->
</sec:http>
```

Sr. No.	Description
(1)	Specify a path for carrying out authentication process in `login-processing-url` attribute.

Note: When path of authentication process is changed, request destination of *Login form* must be changed as well.

Change request parameter name which sends credentials

In Spring Security, request parameters for sending credentials (user name and password) are "`username`" and "`password`" by default, however, these can be changed by defining a bean as shown below.

- Definition example for spring-security.xml

```
<sec:http>
    <sec:form-login
        username-parameter="uid"
        password-parameter="pwd" /> <!-- (1) (2) -->
    <!-- omitted -->
</sec:http>
```

Sr. No.	Description
(1)	Specify a request parameter name for user name in `username-parameter` attribute.
(2)	Specify a request parameter name for password in `password-parameter` attribute.

Note: When request parameter name is changed, field name in *Login form* must also be changed.

Customising response when authentication is successful

Customization points of response when authentication is successful are explained.

Changing default transition destination

Transition destination (default URL) after displaying login form on its own and by carrying out authentication process is the root path of Web application ("/"), however it can be changed by defining a bean as given below.

- Definition example of spring-security.xml

```
<sec:http>
    <sec:form-login default-target-url="/menu" /> <!-- (1) -->
</sec:http>
```

Sr. No.	Description
(1)	Specify default path for transition in `default-target-url` attribute when the authentication is successful.

Fixing transition destination

In default operation of Spring Security, when a request for a page for which authentication is required is received during unauthenticated stage, the request thus received is temporarily stored in HTTP session and then transition is done to authentication page. Although request is restored and redirected when the authentication is successful, transition to the same screen can be assured by defining a bean as below.

- Definition example of spring-security.xml

```
<sec:http>
    <sec:form-login
        default-target-url="/menu"
        always-use-default-target="true" /> <!-- (1) -->
</sec:http>
```

Sr. No.	Description
(1)	Specify `true`in `always-use-default-target` attribute.

Applying AuthenticationSuccessHandler

When the requirements are not met only by using the system wherein default operations provided by Spring Security are customised, implementation class of `AuthenticationSuccessHandler` interface can be applied directly by defining a bean as given below.

- Definition example of spring-security.xml

```
<bean id="authenticationSuccessHandler" class="com.example.app.security.handler.
↪MyAuthenticationSuccessHandler"> <!-- (1) -->

<sec:http>
    <sec:form-login authentication-success-handler-ref=
↪"authenticationSuccessHandler" /> <!-- (2) -->
</sec:http>
```

Sr. No.	Description
(1)	Define a bean for implementation class of `AuthenticationSuccessHandler` interface.
(2)	Specify `authenticationSuccessHandler` defined in `authentication-success-handler-ref` attribute.

Warning: Responsibility of AuthenticationSuccessHandler

`AuthenticationSuccessHandler` is an interface which performs processes for Web layer at the time of successful authentication (mainly processes related to screen transition). Therefore, a process dependent on business rule (business logic) like clearing number of authentication failures should not be called through implementation class of this interface.

"*Handling authentication event*" system introduced in earlier section should be used for calling the processes that are dependent on business rules.

Customising response at the time of authentication failure

Customization points for the response at the time of authentication failure are explained.

Changing transition destination

In default operation of Spring Security, the user is redirected to a URL wherein a query parameter `"error"`is assigned to path for displaying login form, however it can be changed by defining a bean as given below.

- Definition example of spring-security.xml

```
<sec:http>
    <sec:form-login authentication-failure-url="/loginFailure" /> <!-- (1) -->
</sec:http>
```

Sr.No.	Description
(1)	Specify a path for transition in `authentication-failure-url` attribute at the time of authentication failure.

Applying AuthenticationFailureHandler

When the requirements are not met only by using the system wherein default operations provided by Spring Security are customised, implementation class of `AuthenticationFailureHandler` interface can be applied directly by defining a bean as below.

- Definition example of spring-security.xml

```
 <!-- (1) -->
<bean id="authenticationFailureHandler"
    class="org.springframework.security.web.authentication.
↪ExceptionMappingAuthenticationFailureHandler" />
    <property name="defaultFailureUrl" value="/login/systemError" /> <!-- (2) -->
    <property name="exceptionMappings"> <!-- (3) -->
        <props>
            <prop key="org.springframework.security.authentication.
↪BadCredentialsException"> <!-- (4) -->
                /login/badCredentials
            </prop>
            <prop key="org.springframework.security.core.userdetails.
↪UsernameNotFoundException"> <!-- (5) -->
                /login/usernameNotFound
```

```
            </prop>
            <prop key="org.springframework.security.authentication.
↪DisabledException"> <!-- (6) -->
                /login/disabled
            </prop>
            <!-- omitted -->
        </props>
    </property>
</bean>

<sec:http>
    <sec:form-login authentication-failure-handler-ref=
↪"authenticationFailureHandler" /> <!-- (7) -->
</sec:http>
```

Sr. No.	Description
(1)	Define a bean for implementation class of `AuthenticationFailureHandler` interface.
(2)	Specify default transition destination URL in `defaultFailureUrl` attribute. When the exceptions that do not match the definitions given in (4)~(6) below occur, move to transition destination of the configuration.
(3)	Set implementation class of `org.springframework.security.authentication.AuthenticationServiceExcep` handled in the `exceptionMappings` property and transition destination at the time exception occurrence in `Map` format. Set implementation class of `org.springframework.security.authentication.AuthenticationServiceExcep` in key and set transition destination URL in value.
(4)	`BadCredentialsException` It is thrown at the time of authentication error due to password verification failure.

次のページに続く

Table. 9.20 – 前のページからの続き

Sr. No.	Description
(5)	`UsernameNotFoundException` It is thrown at the time of authentication error due to invalid user ID (non-existent user ID). When `org.springframework.security.authentication.dao.AbstractUserDetailsAut` specifies inherited class in the authentication provider, the exception changes to `BadCredentialsException` if `hideUserNotFoundExceptions` property is not changed to `false`.
(6)	`DisabledException` It is thrown at the time of authentication error due to invalid user ID.
(7)	Set `authenticationFailureHandler` in `authentication-failure-handler-ref` attribute.

Note: Control during occurrence of exception

When an exception defined in `exceptionMappings` property occur, the user is redirected to transition destination mapped in the exception, however, since exception object thus occurred is not stored in the session scope, error message generated by Spring Security cannot be displayed on the screen.

Therefore, error message displayed in the transition destination screen must be generated by the process for redirect (Controller or View process).

Further, it must be added that since the processes referring properties below are not called, operation does not undergo any change even after changing the setup value.

- `useForward`
- `allowSessionCreation`

Customising logout process

Customization points for logout process are explained.

Changing logout path

In Spring Security, default path which carries out logout process is "`"/logout"`", however it can be changed by defining a bean as given below.

- Definition example of spring-security.xml

```
<sec:http>
    <!-- omitted -->
    <sec:logout logout-url="/auth/logout" /> <!-- (1) -->
    <!-- omitted -->
</sec:http>
```

Sr. No.	Description
(1)	Set `logout-url` attribute and specify path to carry out logout process.

Note: When logout path is changed, request destination of *logout form* must be changed as well.

Tip: Behaviour during occurrence of system error When system error occurs, discontinuation of operations is likely. If the operation is not to be continued after occurrence of system error, adopting following measures is recommended.

- Clear session information when the system error occurs.
- Clear authentication information when the system error occurs

An example wherein authentication information at the time of system exception occurrence is cleared using an exception handling function of common library is explained. For details of exception handling function, refer "*Exception Handling*".

```
// (1)
public class LogoutSystemExceptionResolver extends SystemExceptionResolver {
    // (2)
    @Override
    protected ModelAndView doResolveException(HttpServletRequest request,
            HttpServletResponse response, java.lang.Object handler,
            java.lang.Exception ex) {

        // Carry out SystemExceptionResolver
        ModelAndView resulut = super.doResolveException(request, response,
                handler, ex);

        // Clear authentication information (2)
        SecurityContextHolder.clearContext();

        return resulut;
    }
}
```

Sr. No.	Description
(1)	Extend `org.terasoluna.gfw.web.exception.SystemExceptionResolver.SystemExceptionR`
(2)	Clear authentication information.

Further, the same requirement can also be met by clearing the session, besides using the method that clears authentication information. Implement according to the project requirement.

Customising response when logout is successful

Customization points for the response when logout process is successful is explained.

Changing transition destination

- Definition example of spring-security.xml

```
<sec:http>
  <!-- omitted -->
  <sec:logout logout-success-url="/logoutSuccess" /> <!-- (1) -->
  <!-- omitted -->
</sec:http>
```

Sr. No.	Description
(1)	Set `logout-success-url` attribute and specify path for the transition when the logout is successful.

Applying LogoutSuccessHandler

- Definition example of spring-security.xml

```
<!-- (1) -->
<bean id="logoutSuccessHandler" class="com.example.app.security.handler.
↪MyLogoutSuccessHandler" />

<sec:http>
    <!-- omitted -->
    <sec:logout success-handler-ref="logoutSuccessHandler" /> <!-- (2) -->
    <!-- omitted -->
</sec:http>
```

Sr. No.	Description
(1)	Define a bean for implementation class of `LogoutSuccessHandler` interface.
(2)	Set `LogoutSuccessHandler`in `success-handler-ref` attribute.

Customising error message

When authentication fails, error message provided by Spring Security is displayed, however, the error message can be changed.

For details of how to change the message, refer *Message Management*.

Message during system error

When an unexpected error occurs (system error etc) during authentication process, `InternalAuthenticationServiceException` exception is thrown. Since cause exception message is set in the message retained by `InternalAuthenticationServiceException`, it should not be displayed as it is on the screen.

For example, when a DB access error occurs while fetching user information from database, an exception message retained by `SQLException` is displayed on the screen. The measures like handling `InternalAuthenticationServiceException` by using `ExceptionMappingAuthenticationFailureHandler` and transiting to the path for notifying occurrence of system error are necessary for not displaying exception message of system error on the screen.

- Definition example of spring-security.xml

```
  <bean id="authenticationFailureHandler"
      class="org.springframework.security.web.authentication.
↪ExceptionMappingAuthenticationFailureHandler">
      <property name="defaultFailureUrl" value="/login?error" />
      <property name="exceptionMappings">
          <props>
              <prop key="org.springframework.security.authentication.
↪InternalAuthenticationServiceException">
                  /login?systemError
              </prop>
              <!-- omitted -->
          </props>
      </property>
  </bean>

<sec:http>
    <sec:form-login authentication-failure-handler-ref=
↪"authenticationFailureHandler" />
</sec:http>
```

Query parameter (`systemError`) is used for identifying occurrence of system error and transition is made to login form. When `systemError` is specified in the query parameter, in the login form specified in the transition destination, a fixed error message is displayed instead of an authentication exception message.

- Implementation example of login form

```
<c:choose>
    <c:when test="${param.containsKey('error')}">
        <span style="color: red;">
            <c:out value="${SPRING_SECURITY_LAST_EXCEPTION.message}"/>
        </span>
    </c:when>
    <c:when test="${param.containsKey('systemError')}">
        <span style="color: red;">
            System Error occurred.
        </span>
    </c:when>
</c:choose>
```

Note: An implementation example for transiting to login form is introduced, however it is also possible to move to system error screen.

Input check at the time of authentication

A check is carried out in advance for obvious input error on the authentication page during reduction of load on DB server etc. In such a case, input check using Bean validation can also be carried out.

Input check using Bean Validation

An example of input check using Bean Validation is explained below. For details of Bean Validation, refer *Input Validation*.

- Implementation example of form class

```
public class LoginForm implements Serializable {

    // omitted
    @NotEmpty // (1)
    private String username;

    @NotEmpty // (1)
```

```
    private String password;
    // omitted

}
```

Sr. No.	Description
(1)	In this example, `username`and `password` are mandatory respectively.

- Implementation example of controller class

```
@ModelAttribute
public LoginForm setupForm() { // (1)
    return new LoginForm();
}

@RequestMapping(value = "login")
public String login(@Validated LoginForm form, BindingResult result) {
    // omitted
    if (result.hasErrors()) {
        // omitted
    }
    return "forward:/authenticate"; // (2)
}
```

Sr. No.	Description
(1)	Initialise `LoginForm`.
(2)	**Forward** to path specified in `login-processing-url` attribute of `<sec:form-login>`element using “forward”. For setup related to authentication, refer *Customizing form authentication*.

In addition, authentication path is added to Spring Security servlet filter for Spring Security processing even during the transition using Forward.

- Setup example of web.xml

```
<filter>
    <filter-name>springSecurityFilterChain</filter-name>
    <filter-class>
        org.springframework.web.filter.DelegatingFilterProxy
    </filter-class>
```

```
</filter>
<filter-mapping>
    <filter-name>springSecurityFilterChain</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
<!-- (1) -->
<filter-mapping>
    <filter-name>springSecurityFilterChain</filter-name>
    <url-pattern>/authenticate</url-pattern>
    <dispatcher>FORWARD</dispatcher>
</filter-mapping>
```

Sr. No.	Description
(1)	Specify pattern for authentication by using Forward. Here "`/authenticate`" is specified as an authentication path.

Expansion of authentication process

In case of authentication requirements which cannot be handled by authentication provider offered by Spring Security, a class which implements `org.springframework.security.authentication.AuthenticationProvider` interface must be created.

Here, an expansion example for performing DB authentication by using 3 parameters - user name, password and **Company identifier (unique authentication parameter)**is shown below.



A class shown below must be created for implementing above requirements.

Sr. No.	Description
(1)	An implementation class of `org.springframework.security.core.Authentication` interface which retains user name, password and company identifier. It is created by inheriting `org.springframework.security.authentication.UsernamePasswordAuthenticationTok` class.
(2)	An implementation class of `org.springframework.security.authentication.AuthenticationProvider` which performs DB authentication by using user name, password and company identifier. It is created by inheriting `org.springframework.security.authentication.dao.DaoAuthenticationProvider` class.
(3)	An Authentication Filter class for generating `Authentication` which is passed to `AuthenticationManager`(`AuthenticationProvider`) by fetching user name, password and company identifier from request parameters. It is created by inheriting `org.springframework.security.web.authentication.UsernamePasswordAuthenticatio` class.

Note: Since a unique parameter is added as an parameter for authentication in this example, implementation class of `Authentication`interface and Authentication Filter class for generating `Authentication` must be expanded.

When the authentication is to be performed only by user name and password, authentication process can be expanded only by creating implementation class of `AuthenticationProvider` interface

Creating an implementation class of Authentication interface

`UsernamePasswordAuthenticationToken` class is inherited and a class that retains a company identifier

(unique authentication parameter) besides user name and password is created.

```
// import omitted
public class CompanyIdUsernamePasswordAuthenticationToken extends
    UsernamePasswordAuthenticationToken {

    private static final long serialVersionUID = SpringSecurityCoreVersion.SERIAL_
↪VERSION_UID;

    // (1)
    private final String companyId;

    // (2)
    public CompanyIdUsernamePasswordAuthenticationToken(
            Object principal, Object credentials, String companyId) {
        super(principal, credentials);
        this.companyId = companyId;
    }

    // (3)
    public CompanyIdUsernamePasswordAuthenticationToken(
            Object principal, Object credentials, String companyId,
            Collection<? extends GrantedAuthority> authorities) {
        super(principal, credentials, authorities);
        this.companyId = companyId;
    }

    public String getCompanyId() {
        return companyId;
    }

}
```

Sr. No.	Description
(1)	Create a field that retains a company identifier.
(2)	Create a constructor to be used while creating an instance which retains information prior to authentication (information specified by request parameter).
(3)	Create a constructor to be used while creating an instance which retains authenticated information. Authenticated state is reached by passing authorization information to the argument of parent class constructor.

Creating an implementation class of AuthenticationProvider interface

`DaoAuthenticationProvider` class is inherited and a class which performs DB authentication by using user name, password and company identifier is created.

```
// import omitted
public class CompanyIdUsernamePasswordAuthenticationProvider extends
    DaoAuthenticationProvider {

    // omitted

    @Override
    protected void additionalAuthenticationChecks(UserDetails userDetails,
            UsernamePasswordAuthenticationToken authentication)
            throws AuthenticationException {

        // (1)
        super.additionalAuthenticationChecks(userDetails, authentication);

        // (2)
        CompanyIdUsernamePasswordAuthenticationToken␣
↪companyIdUsernamePasswordAuthentication =
                (CompanyIdUsernamePasswordAuthenticationToken) authentication;
        String requestedCompanyId = companyIdUsernamePasswordAuthentication.
↪getCompanyId();
        String companyId = ((SampleUserDetails) userDetails).getAccount().
↪getCompanyId();
        if (!companyId.equals(requestedCompanyId)) {
            throw new BadCredentialsException(messages.getMessage(
                    "AbstractUserDetailsAuthenticationProvider.badCredentials",
                    "Bad credentials"));
        }
    }

    @Override
    protected Authentication createSuccessAuthentication(Object principal,
            Authentication authentication, UserDetails user) {
        String companyId = ((SampleUserDetails) user).getAccount()
                .getCompanyId();
        // (3)
        return new CompanyIdUsernamePasswordAuthenticationToken(user,
                authentication.getCredentials(), companyId,
                user.getAuthorities());
    }

    @Override
```

```
    public boolean supports(Class<?> authentication) {
        // (4)
        return CompanyIdUsernamePasswordAuthenticationToken.class
                .isAssignableFrom(authentication);
    }

}
```

Sr. No.	Description
(1)	Call parent class method and implement check process provided by Spring Security. This process also includes password authentication process.
(2)	When password authentication is successful, check validity of company identifier (unique authentication parameter). In the example above, it is checked whether the requested company identifier and the company identifier retained in the table are identical.
(3)	When password authentication and unique authentication process are successful, `CompanyIdUsernamePasswordAuthenticationToken`is created in an authenticated state and then returned.
(4)	When castable `Authentication` is specified in `CompanyIdUsernamePasswordAuthenticationToken`, perform authentication process by using this class.

Note: User existence check, user status check (checks for invalid user, locked user, expired user etc) are carried out as processes of parent class before calling `additionalAuthenticationChecks` method.

Creating Authentication Filter

`UsernamePasswordAuthenticationFilter` class is inherited and an Authentication Filter class is created for passing authentication information (user name, password and company identifier) to `AuthenticationProvider`.

`attemptAuthentication` method implementation is customised by copying method of `UsernamePasswordAuthenticationFilter` class.

```
// import omitted
public class CompanyIdUsernamePasswordAuthenticationFilter extends
    UsernamePasswordAuthenticationFilter {

    @Override
    public Authentication attemptAuthentication(HttpServletRequest request,
            HttpServletResponse response) throws AuthenticationException {

        if (!request.getMethod().equals("POST")) {
            throw new AuthenticationServiceException("Authentication method not␣
↪supported: "
                    + request.getMethod());
        }

        // (1)
        // Obtain UserName, Password, CompanyId
        String username = super.obtainUsername(request);
        String password = super.obtainPassword(request);
        String companyId = obtainCompanyId(request);
        if (username == null) {
            username = "";
        } else {
            username = username.trim();
        }
        if (password == null) {
            password = "";
        }
        CompanyIdUsernamePasswordAuthenticationToken authRequest =
            new CompanyIdUsernamePasswordAuthenticationToken(username, password,␣
↪companyId);

        // Allow subclasses to set the "details" property
        setDetails(request, authRequest);

        return this.getAuthenticationManager().authenticate(authRequest); // (2)
    }

    // (3)
    protected String obtainCompanyId(HttpServletRequest request) {
        return request.getParameter("companyId");
    }
```

```
}
```

Sr. No.	Description
(1)	Generate an instance of `CompanyIdUsernamePasswordAuthenticationToken` from the authentication information fetched from request parameter (user name, password, company identifier).
(2)	Specify authentication information specified by request parameter (`CompanyIdUsernamePasswordAuthenticationToken` instance) and call `authenticate` method of `org.springframework.security.authentication.AuthenticationManager`. If `AuthenticationManager` method is called, `AuthenticationProvider` authentication process gets called.
(3)	Company identifier is fetched by using request parameter `"companyId"`.

Modifying login form

Company identifier is added for login form (JSP) created by *Creating login form*.

```
<form:form action="${pageContext.request.contextPath}/login" method="post">
    <!-- omitted -->
        <tr>
            <td><label for="username">User Name</label></td>
            <td><input type="text" id="username" name="username"></td>
        </tr>
        <tr>
            <td><label for="companyId">Company Id</label></td>
            <td><input type="text" id="companyId" name="companyId"></td> <!-- (1) -
↪->
        </tr>
        <tr>
            <td><label for="password">Password</label></td>
            <td><input type="password" id="password" name="password"></td>
        </tr>
```

```
    <!-- omitted -->
</form:form>
```

Sr. No.	Description
(1)	Specify "companyId" in input field name of company identifier.

Applying expanded authentication process

DB authentication process which use user name, password and company identifier (unique authentication parameter) are applied in Spring Security.

- Definition example of spring-security.xml

```
<!-- omitted -->

<!-- (1) -->
<sec:http
    entry-point-ref="loginUrlAuthenticationEntryPoint">

    <!-- omitted -->

    <!-- (2) -->
    <sec:custom-filter
        position="FORM_LOGIN_FILTER" ref=
↪"companyIdUsernamePasswordAuthenticationFilter" />

    <!-- omitted -->

    <sec:csrf token-repository-ref="csrfTokenRepository" />

    <sec:logout
        logout-url="/logout"
        logout-success-url="/login" />

    <!-- omitted -->

    <sec:intercept-url pattern="/login" access="permitAll" />
    <sec:intercept-url pattern="/**" access="isAuthenticated()" />

    <!-- omitted -->

</sec:http>
```

```
<!-- (3) -->
<bean id="loginUrlAuthenticationEntryPoint"
    class="org.springframework.security.web.authentication.
↪LoginUrlAuthenticationEntryPoint">
    <constructor-arg value="/login" />
</bean>

<!-- (4) -->
<bean id="companyIdUsernamePasswordAuthenticationFilter"
    class="com.example.app.common.security.
↪CompanyIdUsernamePasswordAuthenticationFilter">
    <!-- (5) -->
    <property name="requiresAuthenticationRequestMatcher">
        <bean class="org.springframework.security.web.util.matcher.
↪AntPathRequestMatcher">
            <constructor-arg index="0" value="/authentication" />
            <constructor-arg index="1" value="POST" />
        </bean>
    </property>
    <!-- (6) -->
    <property name="authenticationManager" ref="authenticationManager" />
    <!-- (7) -->
    <property name="sessionAuthenticationStrategy" ref=
↪"sessionAuthenticationStrategy" />
    <!-- (8) -->
    <property name="authenticationFailureHandler">
        <bean class="org.springframework.security.web.authentication.
↪SimpleUrlAuthenticationFailureHandler">
            <constructor-arg value="/login?error=true" />
        </bean>
    </property>
    <!-- (9) -->
    <property name="authenticationSuccessHandler">
        <bean class="org.springframework.security.web.authentication.
↪SimpleUrlAuthenticationSuccessHandler" />
    </property>
</bean>

<!-- (6') -->
<sec:authentication-manager alias="authenticationManager">
    <sec:authentication-provider ref=
↪"companyIdUsernamePasswordAuthenticationProvider" />
</sec:authentication-manager>
<bean id="companyIdUsernamePasswordAuthenticationProvider"
    class="com.example.app.common.security.
↪CompanyIdUsernamePasswordAuthenticationProvider">
    <property name="userDetailsService" ref="sampleUserDetailsService" />
    <property name="passwordEncoder" ref="passwordEncoder" />
</bean>
```

```
<!-- (7') -->
<bean id="sessionAuthenticationStrategy"
    class="org.springframework.security.web.authentication.session.
↪CompositeSessionAuthenticationStrategy">
    <constructor-arg>
        <util:list>
            <bean class="org.springframework.security.web.csrf.
↪CsrfAuthenticationStrategy">
                <constructor-arg ref="csrfTokenRepository" />
            </bean>
            <bean class="org.springframework.security.web.authentication.session.
↪SessionFixationProtectionStrategy" />
        </util:list>
    </constructor-arg>
</bean>

<bean id="csrfTokenRepository"
    class="org.springframework.security.web.csrf.HttpSessionCsrfTokenRepository" />


<!-- omitted -->
```

Sr. No.	Description
(1)	When "FORM_LOGIN_FILTER" is to be changed by using <sec:custom-filter> tag of (2), following settings must be applied to <sec:http> tag attribute. • Since auto-configuration cannot be used, auto-config="false" is specified or auto-config attribute is deleted. • Since <sec:form-login> tag cannot be used, AuthenticationEntryPoint is specified explicitly by using entry-point-ref attribute.
(2)	Use <sec:custom-filter> tag and change "FORM_LOGIN_FILTER". Specify "FORM_LOGIN_FILTER" in position attribute of <sec:custom-filter> tag and specify a bean for Authentication Filter expanded in ref attribute.

次のページに続く

Table. 9.21 – 前のページからの続き

Sr. No.	Description
(3)	Specify a bean of `AuthenticationEntryPoint`used in `entry-point-ref` attribute of `<sec:http>` tag. Here, a bean of `org.springframework.security.web.authentication.LoginUrlAuthenticationEntryPo` class used while specifying `<sec:form-login>` tag is specified.
(4)	Define a bean of Authentication Filter class used as `"FORM_LOGIN_FILTER"`. Here, a bean of expanded Authentication Filter class (`CompanyIdUsernamePasswordAuthenticationFilter`) is defined.
(5)	Specify `RequestMatcher` instance for detecting a request performing authentication process, in `requiresAuthenticationRequestMatcher` property. It is configured to perform authentication process when a request is raised in the `"/authentication"` path. This is similar to specifying `"/authentication"` in `login-processing-url` attribute of `<sec:form-login>` tag.
(6)	Specify a value set in `alias` attribute of `<sec:authentication-manager>` tag, in `authenticationManager` property. If `alias` attribute of `<sec:authentication-manager>` tag is specified, a bean of `AuthenticationManager` generated by Spring Security can be used for DI of another bean.
(6')	Set expanded `AuthenticationProvider`(`CompanyIdUsernamePasswordAuthenticationProvider`) for `AuthenticationManager` generated by Spring Security.

次のページに続く

Table. 9.21 – 前のページからの続き

Sr. No.	Description
(7)	Specify a bean of component (`SessionAuthenticationStrategy`) that controls handling of a session during successful authentication in `sessionAuthenticationStrategy` property.
(7')	Define a bean of component (`SessionAuthenticationStrategy`) that controls handling of a session during successful authentication. Here, • A component to recreate CSRF token (`CsrfAuthenticationStrategy`) • A component to generate a new session for preventing session fixation attack (`SessionFixationProtectionStrategy`) offered by Spring Security are enabled.
(8)	Specify a handler class in `authenticationFailureHandler` property which is called at the time of authentication failure.
(9)	Specify a handler class in `authenticationSuccessHandler` property which is called at the time of successful authentication.

Note: Regarding auto-config

When Basic authentication process and logout process are to be enabled by specifying `auto-config="false"` or by omitting specification, `<sec:http-basic>` tag and `<sec:logout>` tag must be defined explicitly.

Using PasswordEncoder of deprecated package

Based on security requirements, it is likely that the implementation is not possible using the class which implements `PasswordEncoder` described earlier. In particular, when it is necessary to follow hashing requirements used in the existing account information, it may not meet the requirements of `PasswordEncoder` described earlier.

Basically, existing hashing requirements are as given below.

- Algorithm is SHA-512.
- Stretching frequency is 1000.
- Salt is stored in the account table column and it must be passed from outside of `PasswordEncoder`.

In such a case, implementation class of `org.springframework.security.authentication.encoding.PasswordE` interface must be used to meet requirements instead of implementation class of `org.springframework.security.crypto.password.PasswordEncoder` interface.

Warning: In Spring Security 3.1.4 and earlier versions, a class that implements `org.springframework.security.authentication.encoding.PasswordEncoder`was used in the hashing, however, it is deprecated in the 3.1.4 and subsequent versions.

Using ShaPasswordEncoder

This guideline explains the use of `PasswordEncoder`of deprecated package using an example of `ShaPasswordEncoder`.

When the hashing requirements are as below, requirements can be met by using `ShaPasswordEncoder`.

- Algorithm is SHA-512
- Stretching frequency is 1000

First, define a bean for `ShaPasswordEncoder`.

- Definition example of applicationContext.xml

```
<bean id ="passwordEncoder"
    class="org.springframework.security.authentication.encoding.ShaPasswordEncoder
↪"> <!-- (1) -->
    <constructor-arg value="512" /> <!-- (2) -->
    <property name="iterations" value="1000" /> <!-- (3) -->
</bean>
```

Sr. No.	Description
(1)	Define a bean for `org.springframework.security.authentication.encoding.ShaPasswordEncoder`.
(2)	Specify type of SHA algorithm. The values that can be specified are "`1`, `256`, `384`, `512`". Value is "`1`" when it is not specified.
(3)	Specify stretching frequency at the time of hashing. Value is 1 when it is not specified.

Next, `ShaPasswordEncoder` is applied to authentication process (`DaoAuthenticationProvider`) of Spring Security.

- Definition example of spring-security.xml

```
<bean id="authenticationProvider"
    class="org.springframework.security.authentication.dao.
↪DaoAuthenticationProvider">
    <!-- omitted -->
    <property name="saltSource" ref="saltSource" /> <!-- (1) -->
    <property name="userDetailsService" ref="userDetailsService" />
    <property name="passwordEncoder" ref="passwordEncoder" /> <!-- (2) -->
</bean>

<bean id="saltSource"
    class="org.springframework.security.authentication.dao.ReflectionSaltSource">
↪<!-- (3) -->
    <property name="userPropertyToUse" value="username" /> <!-- (4) -->
</bean>
```

Sr. No.	Description
(1)	Specify a bean for implementation class of `org.springframework.security.authentication.dao.SaltSource` in `saltSource` property. `SaltSource` is an interface that fetches salt from `UserDetails`.
(2)	Specify a bean for implementation class of `org.springframework.security.authentication.encoding.PasswordEncoder` interface in `passwordEncoder` property. In the example above, a bean for `ShaPasswordEncoder` is specified.
(3)	Define a bean for `SaltSource`. In the example above, a class (`ReflectionSaltSource`) that fetches salt from `UserDetails` property using reflection, is used.
(4)	Specify `UserDetails` property where salt is stored. In the example above, a value of `username` property of `UserDetails` is used as a salt.

When deprecated `PasswordEncoder` is to be used in the process of application, `PasswordEncoder` is used after injection.

- Implementation example of Java class

```
@Inject
PasswordEncoder passwordEncoder;

public String register(Customer customer, String rawPassword, String userSalt) {
    // omitted
    String password = passwordEncoder.encodePassword(rawPassword, userSalt); // (1)
    customer.setPassword(password);
    // omitted
}

public boolean matches(Customer customer, String rawPassword, String userSalt) {
    return passwordEncoder.isPasswordValid(customer.getPassword(), rawPassword,␣
↪userSalt); // (2)
}
```

Sr. No.	Description
(1)	When password is to be hashed, use `encodePassword` method. Specify password, salt string in the method argument, in a sequence.
(2)	When the password is to be verified, use `isPasswordValid` method. Specify hashed password, plain text password and salt string in the method argument, in a sequence.

9.2.4 Appendix

Receive a request by Spring MVC and display login form

A method wherein a request is received by Spring MVC and login form is displayed is explained.

- Definition example of spring-mvc.xml

Definition example of Controller which displays login form.

```
@Controller
@RequestMapping("/login")
```

```
public class LoginController { // (1)

    @RequestMapping
    public String index() {
        return "login";
    }
}
```

Sr. No.	Description
(1)	Return "login" as view name. src/main/webapp/WEB-INF/views/login.jsp is output by `InternalResourceViewResolver`

As per this example, it is also possible to substitute by using `<mvc:view-controller>` in case of a controller with only one method which simply returns only the view name.

For details, refer *HTML response*.

Using Remember Me authentication

"Remember Me authentication" is one of the functions to enhance the user experience who frequently access Websites, and retains logged in state of a user for a little longer than normal lifecycle. If this function is used, the user can login again without entering user name and password by using a Token for Remember Me authentication retained by a Cookie, even when the browser is closed or session has timed-out. Note that, this function is enabled only when the user has authorised retaining the logged in state.

Spring Security supports "Remember Me authentication of Hash-Based Token method and Persistent Token method. Hash-Based Token method is used as a default.

When Remember Me authentication is to be used, `<sec:remember-me>` tag is added.

- Definition example of spring-security.xml

```
<sec:http>
    <!-- omitted -->
    <sec:remember-me key="terasoluna-tourreservation-km/ylnHv"
```

```
        token-validity-seconds="#{30 * 24 * 60 * 60}" />  <!-- (1) (2) -->
    <!-- omitted -->
</sec:http>
```

Sr. No.	Description
(1)	Specify a key value in `key` attribute for identifying an application that generates a Token for Remember Me authentication. When key value is not specified, a unique value is generated every time an application starts. Note that, when a key value retained by Hash-Based Token and a key value retained by server vary, it is handled as an invalid Token. In other words, when a Hash-Based Token generated before restarting an application is to be handled as a valid Token, `key` attribute must be specified.
(2)	Specify validity period of Token for Remember Me authentication in seconds, in the `token-validity-seconds` attribute. When it is not specified, the validity period is 14 days by default. In the example above, 30 days has been set as a validity period.

For attributes other than above, refer Spring Security Reference -The Security Namespace (<remember-me>) -.

Note: Changes in Spring Security 4.0

Default value of following settings is changed from Spring Security 4.0 version

- remember-me-parameter
- remember-me-cookie

A flag (checkbox field) to specify use of “Remember Me authentication” function is provided in the login form.

- Implementation example of JSP of login form

```
<form:form action="${pageContext.request.contextPath}/login" method="post">
        <!-- omitted -->
        <tr>
```

```
            <td><label for="remember-me">Remember Me : </label></td>
            <td><input name="remember-me" id="remember-me" type="checkbox" checked=
↪"checked" value="true"></td> <!-- (1) -->
        </tr>
        <!-- omitted -->
</form:form>
```

Sr. No.	Description
(1)	Add a flag (checkbox item) to specify whether "Remember Me authentication" function is used, and specify `remember-me` - a default value of `remember-me-parameter` in field name (request parameter name). Set `true` in `value` attribute of checkbox. If authentication process is carried out after ticking the checkbox, "Remember Me authentication" function is applied for subsequent requests.

Tip: About setup value of value attribute

Description of setting `true` in `value` attribute is given in rememberMeRequested - JavaDoc, however `on`, `yes` and `1` can also be set in the implementation.

9.3 Authorization

9.3.1 Overview

This chapter explains authorization function provided by Spring Security.

Authorization process controls the resources that can be accessed by the users of the application. The standard method to control the resources that can be accessed by the user include defining an access policy for each resource (or a group of resources) and control the resources by checking the access policy when the user tries to access a resource.

Access policy defines whether to allow a user to access a particular resource. An access policy for following 3 types of resources is defined in Spring Security.

- Web resource
- Java method
- Domain object [*1]
- Screen fields of JSP

This section introduces an implementation example (definition example) wherein authorization process is applied to access “Web resource”, “Java method” and “Screen fields of JSP” and explains the authorization function of Spring Security.

Authorization process architecture

Spring Security performs authorization process as per the flow below.

[*1] For authorization function to access domain object , refer Spring Security Reference -Domain Object Security (ACLs)-.

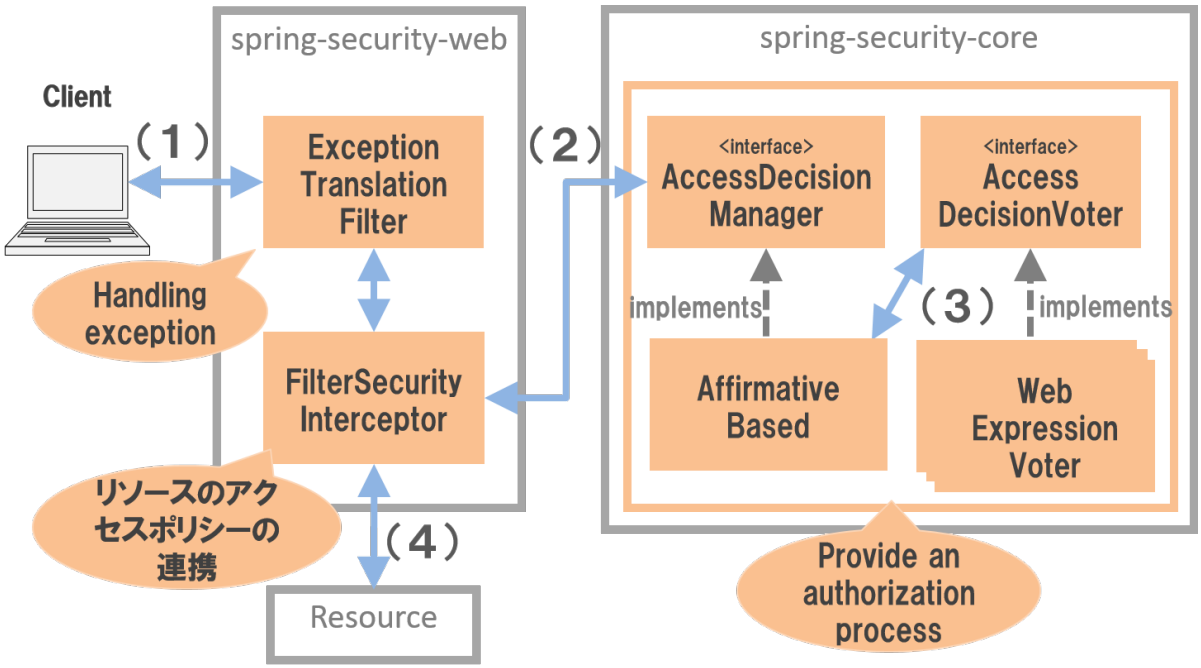


Figure.9.7 **Authorization process architecture**

Sr. No.	Description
(1)	A client accesses any resource.
(2)	`FilterSecurityInterceptor` class calls `AccessDecisionManager` interface method and checks whether the user has access rights for the resource.
(3)	`AffirmativeBased` class (Implementation class of `AccessDecisionManager` used as a default) calls `AccessDecisionVoter` interface method and votes for whether the user has access rights.
(4)	`FilterSecurityInterceptor` accesses the resource only if the access rights have been granted by `AccessDecisionManager`.

ExceptionTranslationFilter

`ExceptionTranslationFilter` is a security filter which handles exceptions generated by authorization process (`AccessDecisionManager`) and sends appropriate response to the client. In the default implementation, a response to ask for authentication in case of access by an unauthenticated user and a response to throw an authorization error in case of an authenticated user are returned.

FilterSecurityInterceptor

`FilterSecurityInterceptor` is a security filter to apply an authorization process for HTTP request and delegates actual authorization process to `AccessDecisionManager`. While calling a method of `AccessDecisionManager` interface, it is linked with the access policy specified in the resource which a client trying to access.

AccessDecisionManager

`AccessDecisionManager` interface checks whether the user has the access rights for the resource which he is trying to access.

Spring Security provides 3 types of implementation classes and all the classes call `AccessDecisionVoter` interface method and determine whether the access rights have been granted. `AccessDecisionVoter` votes for "Assign", "Deny" or Abstain" and then implementation class of `AccessDecisionManager` aggregates the voting results and determines final access rights. `AccessDeniedException` exception is thrown and access is denied if determined as "no access rights".

Note that, if all the voting results indicate "Abstain", it is determined as "no access rights" in Spring Security by default.

Table.9.22 **Implementation class of AccessDecisionManager provided by Spring Security**

Class name	Description
`AffirmativeBased`	An implementation class which assigns the access rights when 1 vote is given to "Assign" during voting by `AccessDecisionVoter`. **Implementation class used as a default.**
`ConsensusBased`	An implementation class which assigns the access rights when the majority of votes are for "Assign" during voting for all `AccessDecisionVoter`. When 1 vote each is given to "Assign" and "Deny" or when it is a tie, it is considered as "Have access rights" by default in Spring Security.
`UnanimousBased`	An implementation class which **does not give access rights** when 1 vote is given to "Deny" during voting by `AccessDecisionVoter`.

Note: Selecting AccessDecisionVoter

If only one `AccessDecisionVoter` is used, no difference is observed in the operation regardless of the implementation class. When multiple `AccessDecisionVoter` are used, implementation class should be selected in accordance with the requirements.

AccessDecisionVoter

`AccessDecisionVoter` is an interface which checks the access policy specified in the resource which the user is trying to access and votes for whether the access rights are to be granted.

Main implementation classes provided by Spring Security are as given below.

Table.9.23 **Main implementation class of AccessDecisionVoter provided by Spring Security**

Class name	Description
`WebExpressionVoter`	An implementation class which carries out voting by checking the rights information retained by authentication information (`Authentication`) through SpEL and request information (`HttpServletRequest`).
`RoleVoter`	An implementation class which carries out voting by checking the role of the user.
`RoleHierarchyVoter`	An implementation class which carries out voting by checking the role hierarchy of a user.
`AuthenticatedVoter`	An implementation class which carries out voting by checking the authentication status.

Note: AccessDecisionVoter to be applied by default

Implementation class of `AccessDecisionVoter` interface applied by default is integrated with `WebExpressionVoter` from Spring Security 4.0 onwards. Since `WebExpressionVoter` can play the role similar to using `RoleVoter`,`RoleHierarchyVoter` and `AuthenticatedVoter`, this guideline will also assume the use of default `WebExpressionVoter` for explaining authorization process.

9.3.2 How to use

A bean definition example (how to specify an access policy) and implementation method required for using authorization function are explained.

How to describe an access policy

How to describe an access policy is explained.

Spring Security supports Spring Expression Language (SpEL) as a method which describes how to specify an access policy. Although there are other methods which do not use SpEL, this guideline explains how to specify an access policy by using Expression. How to use SpEL is briefly explained in this section, however for detailed description, refer Spring Framework Reference Documentation -Spring Expression Language (SpEL)-.

Built-In Common Expressions

Common Expressions provided by Spring Security are as given below.

Table.9.24: **Common Expression provided by Spring Security**

Expression	Description
`hasRole(String role)`	Return `true` when logged in user has a role specified in the argument.
`hasAnyRole(String... roles)`	Return `true` when logged in user has one of the roles specified in the argument.
`isAnonymous()`	Return `true` in case of an anonymous user who has not logged in.
`isRememberMe()`	Return `true` in case of a user logged in by using Remember Me authentication.
`isAuthenticated()`	Return `true` in case of a login.
`isFullyAuthenticated()`	Return `true` in case of a user who has logged in using normal authentication process instead of Remember Me authentication.

次のページに続く

Table. 9.24 – 前のページからの続き

Expression	Description
`permitAll`	Always return `true`.
`denyAll`	Always return `false`.
`principal`	Return user information of authenticated user (an object of a class which implements `UserDetails` interface).
`authentication`	Return authentication information of authenticated user (an object of a class which implements `Authentication` interface).

Note: Accessing authentication information which uses Expression

Since user information and authentication information of a logged in user can be accessed when `principal` and `authentication` are used as Expressions, an access policy can be set by using attributes other than role.

Note: Role name prefix

It was necessary to specify `"ROLE_"` prefix to role name till Spring Security 3.2. However, specifying `"ROLE_"` prefix is no longer required from Spring Security 4.0 onwards.

Example)

- Before Spring Security 3.2 : `hasRole('ROLE_USER')`
- Spring Security 4.0 onwards : `hasRole('USER')`

Built-In Web Expressions

Expressions for Web applications provided by Spring Security are as below.

Table.9.25 **Expressions for Web applications provided by Spring Security**

Expression	Description
`hasIpAddress(String ipAddress)`	Return `true`when requested IP address matches the IP address system specified in the argument.

Using operator

Determination using an operator can also be performed. In the example below, access can be granted if both role and requested IP address match.

- Definition example of spring-security.xml

```
<sec:http>
    <sec:intercept-url pattern="/admin/**" access="hasRole('ADMIN') and␣
↪hasIpAddress('192.168.10.1')"/>
    <!-- omitted -->
</sec:http>
```

List of operators that can be used

Operator	Description
`[expression-1] and [expression-2]`	Return true when both expression-1 and expression-2 are true.
`[expression-1] or [expression-2]`	Return true when one of the expressions is true.
`![expression]`	Return false when expression is true and return true when expression is false.

Authorization of Web resource

Spring Security performs authorization process for Web resource (HTTP request) using a servlet filter system.

Applying authorization process

Define a bean as below when authorization process is to be applied for a Web resource.

- Definition example of spring-security.xml

```
<sec:http>
    <!-- omitted -->
    <sec:intercept-url pattern="/**" access="isAuthenticated()" />  <!-- (1) -->
    <!-- omitted -->
</sec:http>
```

Sr. No.	Description
(1)	Define an access policy in <sec:intercept-url> tag for a HTTP request. Here, an access policy is defined by using SpEL wherein “Only authenticated users are granted access for all the requests under a Web application”.

Note: Default definition of use-expressions

Since default value of use-expressions attribute of <sec:http> tag is changed to true from Spring Security 4.0 onwards, it is no longer necessary to explicitly describe while using true.

Defining access policy

How to define an access policy for a Web resource using a bean definition file is explained.

Specifying a Web resource for applying access policy

First, a resource (HTTP request) for which an access policy is to be applied, is specified. Attribute under <sec:intercept-url> tag is used for the specification of a resource for which an access policy is to be applied.

Table.9.26 **Attribute for specifying a resource for which an access policy is to be applied**

Attribute name	Description
pattern	An attribute which uses a resource matching with path pattern specified in Ant format or regular expression as an application target.
method	An attribute which uses a resource as an application target when access is to be done by using specified HTTP methods (GET, POST etc).
requires-channel	Specify “http” or “https”. An attribute for controlling the access by specified protocol. if it is not specified, either of these can be accessed.

For the attributes other than above, refer <intercept-url>.

- Definition example of <sec:intercept-url> tag pattern attribute (spring-security.xml)

```
<sec:http >
    <sec:intercept-url pattern="/admin/accounts/**" access="..."/>
    <sec:intercept-url pattern="/admin/**" access="..."/>
    <sec:intercept-url pattern="/**" access="..."/>
    <!-- omitted -->
</sec:http>
```

Spring Security matches the requests in the defined order and the definition which is matched at first is applied. Therefore, definition order must be taken into consideration even while specifying an access policy by using a bean definition file.

Tip: Interpretation of path pattern

Path pattern is interpreted in Ant format, in the default operation of Spring Security. When the path pattern is to be specified in the regular expression, "regex" should be specified in request-matcher attribute of <sec:http> tag

```
<sec:http request-matcher="regex">
    <sec:intercept-url pattern="/admin/accounts/.*" access=hasRole('ACCOUNT_
↪MANAGER')" />
    <!-- omitted -->
</sec:http>
```

Warning: Specifications of path matching for *AntPathRequestMatcher* used by Spring Security by default are now case sensitive for Spring Security 4.1 and subsequent versions.

For example, as shown below, when an access policy is to be defined for endpoint of Spring MVC which allocates /Todo/List path, the values specified in pattern attribute of <sec:intercept-url> tag must be aligned for uppercase and lowercase letters like /Todo/List and /Todo/*. If the values not aligned by uppercase and lowercase letters like /todo/list and /todo/** are specified accidentally, it should be noted that intended authorization control cannot be performed.

- Implementation example of Spring MVC endpoint

```
@RequestMapping(value="/Todo/List")
public String viewTodoList(){
    //...
}
```

- Definition example of access policy

```
<sec:http>
    <sec:intercept-url pattern="/Todo/List" access="isAuthenticated()" />
    <!-- omitted -->
</sec:http>
```

Warning: In Spring MVC and Spring Security, the mechanism of matching with the request is strictly different, and there is a vulnerability that breaks through the authorization function of Spring Security and can access the handler method using this difference. For details of this vulnerability, refer to "CVE-2016-5007 Spring Security / MVC Path Matching Inconsistency".

This vulnerability occurs when the Bean of *org.springframework.util.AntPathMatcher* set *true* to *trimTokens* property is applied to Spring MVC. In Spring Framework 4.2 and earlier, the default value of the *trimTokens* property was set to *true* , but since the default value was changed to *false* from Spring Framework 4.3, this vulnerability does not occur unless user intentionally change the property.

In the blank project of TERASOLUNA Server Framework for Java (5.3.x), *trimTokens* property is set to *false* like below, However, if user set the property to *true* , this vulnerability can occur. So don't change the value.

```
<mvc:annotation-driven>
    <!-- ommited -->
    <mvc:path-matching path-matcher="pathMatcher" />
</mvc:annotation-driven>

<bean id="pathMatcher" class="org.springframework.util.AntPathMatcher">
    <property name="trimTokens" value="false" />
</bean>
```

Further, if an access policy for a specific URL is to be specified (wild cards like `*`, `**` etc are not included in `pattern` attribute), an access policy with a pattern with an extension and a pattern with `/` appended at the end of request path must be added.

In the following configuration example, only the users with "ROLE_ADMIN" role are allowed to access by `/restrict`.

```
<sec:http>
    <sec:intercept-url pattern="/restrict.*" access="hasRole('ADMIN')" />
↪<!-- (1) -->
    <sec:intercept-url pattern="/restrict/" access="hasRole('ADMIN')" /> <!
↪-- (2) -->
    <sec:intercept-url pattern="/restrict" access="hasRole('ADMIN')" /> <!-
↪- (3) -->
    <!-- omitted -->
</sec:http>
```

Sr. No.	Description
(1)	Define an access policy of the pattern with an extension (`/restrict.json` etc) in `/restrict`.

(2)	Define an access policy of the pattern wherein / appended at the end of the request path (/restrict/ etc) in /restrict.
(3)	Define an access policy for /restrict.

Specifying an access policy

Next, an access policy is specified. Access policy is specified in access attribute of <sec:intercept-url> tag.

- Definition example of <sec:intercept-url> tag access attribute (spring-security.xml)

```
<sec:http>
    <sec:intercept-url pattern="/admin/accounts/**" access="hasRole('ACCOUNT_
↪MANAGER')"/>
    <sec:intercept-url pattern="/admin/configurations/**" access=
↪"hasIpAddress('127.0.0.1') and hasRole('CONFIGURATION_MANAGER')" />
    <sec:intercept-url pattern="/admin/**" access="hasRole('ADMIN')" />
    <!-- omitted -->
</sec:http>
```

Table.9.28 **Attribute for specifying an access policy**

Attribute name	Description
access	Specify access control expression using SpEL and a role that can be accessed.

A setting example with roles namely “ROLE_USER” and “ROLE_ADMIN” assigned to login user, is shown below.

- Definition example of <sec:intercept-url> tag patternattribute (spring-security.xml)

```
<sec:http>
    <sec:intercept-url pattern="/reserve/**" access="hasAnyRole('USER','ADMIN
↪')" /> <!-- (1) -->
    <sec:intercept-url pattern="/admin/**" access="hasRole('ADMIN')" /> <!--␣
↪(2) -->
```

```
    <sec:intercept-url pattern="/**" access="denyAll" /> <!-- (3) -->
    <!-- omitted -->
</sec:http>
```

Sr. No.	Description
(1)	To access "//reserve/**", either the role "ROLE_USER" or "ROLE_ADMIN" is required. `hasAnyRole` will be described later.
(2)	To access "/admin/**", the role "ROLE_ADMIN" is required. `hasRole` will be described later.
(3)	`denyAll` is set for all patterns Any user should not be able to access a URL for which no rights settings are described. `denyAll` will be described later.

Note: About description sequence of URL pattern

The request received from client is matched with the pattern in intercept-url, starting from the top and access is granted for the matched pattern. Therefore, pattern should always be described from limited patterns.

SpEL is enabled in Spring Security by default. SpEL described in `access` attribute is determined as a true value. If the expression is true, access is granted. How to use is shown below.

- Definition example of spring-security.xml

```
<sec:http>
    <sec:intercept-url pattern="/admin/**" access="hasRole('ADMIN')"/>  <!--␣
↪(1) -->
    <!-- omitted -->
</sec:http>
```

Sr. No.	Description
(1)	Return true if the logged in user retains the specified role, by specifying `hasRole('Role name')`.

For main Expression that can be used, refer *How to describe an access policy*.

Reference of path variables

In Spring Security 4.1 and subsequent versions, a path variable [#fPathVariableDescription]_ can be used while specifying a resource which applies the access policy and can be referred by specifying `#path variable name` in the definition of access policy.

In the example below, the access policy is defined so that the login users can access only their own user information.

- Definition example of spring-security.xml

```
<sec:http>
    <sec:intercept-url pattern="/users/{userName}" access="isAuthenticated()␣
↪and #userName == principal.username"/>
    <!-- omitted -->
</sec:http>
```

Warning: Precautions while defining an access policy which uses the path variable

When an access policy which uses a path variable for the path that can be accessed with an extension, is defined, it is necessary to define it in such a way that the extension part is not stored in the path variable value.

For example, when `/users/{userName}` is defined in the patten and a request path `/users/personName.json` is sent, `personName.json` gets stored in the path variable `#userName` referred in the definition of access policy, instead of `personName`and unintended authorization control is performed.

In order to prevent this event, "Access policy for the path without extension" must be defined after defining "Access policy for the path with extension" as shown in the example below.

- Definition example of spring-security.xml

```
<sec:http>
 <sec:intercept-url pattern="/users/{userName}.*" access=
↪"isAuthenticated() and #userName == principal.username"/> <!-- (1) -->
 <sec:intercept-url pattern="/users/{userName}" access="isAuthenticated()␣
↪and #userName == principal.username"/> <!-- (2) -->
 <!-- omitted -->
</sec:http>
```

Sr. No.	Description
(1)	Define “Access policy for the path with extension”.
(2)	Define “Access policy for the path without extension”.

Authorization for the method

Spring Security performs authorization process for calling a method of Bean which is managed in DI container by using Spring AOP system.

Authorization process for the method is provided by considering its use for calling a domain layer (service layer) method. If authorization process for the method is used, a detailed access policy can be defined since a property of domain object can be checked.

Enabling AOP

When the authorization process for the method is to be used, a component (AOP) for performing authorization process for calling the method must be enabled. If AOP is enabled, access policy can be defined in the method annotation.

Spring Security supports following annotations.

- `@PreAuthorize`, `@PostAuthorize`, `@PreFilter`, `@PostFilter`
- JSR-250 (`javax.annotation.security` package) annotation (`@RolesAllowed` etc.)
- `@Secured`

This guideline explains how to use `@PreAuthorize` and `@PostAuthorize` which enable the use of access policy by Expression.

- Definition example of spring-security.xml

```
<sec:global-method-security pre-post-annotations="enabled" /> <!-- (1) (2) -->
```

Sr. No.	Description
(1)	AOP which performs authorization process for method calling is enabled if `<sec:global-method-security>` tag is assigned.
(2)	Specify `true` in `pre-post-annotations` attribute. If `true` is specified in `pre-post-annotations` attribute, the annotation that can define an access policy by specifying Expression is enabled.

Applying authorization process

An annotation which specifies an access policy is used and an access policy is defined for each method by applying authorization process for the method.

Defining access policy

Specifying an access policy to be applied prior to method execution

When an access policy is to be specified for applying prior to method execution, `@PreAuthorize` is used.

When the results of the Expression specified in `value` attribute of `@PreAuthorize` become `true`, execution of the method is allowed. In the example below, it has been defined that only administrator can access the account information for other persons.

- Definition example of `@PreAuthorize`

```
// (1) (2)
@PreAuthorize("hasRole('ADMIN') or (#username == principal.username)")
public Account findOne(String username) {
    return accountRepository.findOne(username);
}
```

Sr. No.	Description
(1)	Assign `@PreAuthorize` to the method which needs to be authorized.
(2)	Define an access policy for the method, in `value` attribute. Here, an access policy - "Allow access to all accounts for the administrator" and "Allow access to only individual account when other than administrator" is defined.

The part wherein a method argument is accessed from the Expression is discussed here. Specifically, "`#username`" part accesses the argument. A method argument can be accessed by specifying Expression of "# + argument name" format in Expression.

Tip: Annotation which specifies argument name

Spring Security resolves the argument name from debug information output in the class. However an argument name can be explicitly specified by using an annotation (`@org.springframework.security.access.method.P`)

A variable name can be specified explicitly by using the annotation if it is applicable to following cases.

- Debug information of variable is not output in the class
- When it is to be accessed by using a name different from that of actual variable name in the Expression (for example shortened name)

```
@PreAuthorize("hasRole('ADMIN') or (#username == principal.username)")
public Account findOne(@P("username") String username) {
    return accountRepository.findOne(username);
}
```

Note that when `#username` and a method argument - `username` name are identical, `@P` can be omitted. However, since Spring Security resolves the argument name by using an argument name of implementation class, note that the **argument name of implementation class should match with #username specified in @PreAuthorize** when `@PreAuthorize` annotation is defined in the interface.

When compile option added from JDK 8 (`-parameters`) is used, meta data for reflection is generated in the method parameter. Hence argument name is resolved even when the annotation is not specified.

Specifying access policy to be applied after method execution

When an access policy to be applied after execution of the method is specified, `@PostAuthorize` is used.

When the results of the Expression specified in `value` attribute of `@PostAuthorize` become true, results of method execution are returned to the call source. In the example below, it has been defined such that the user is not allowed to access the information of the user belonging to other department.

- Definition example of `@PostAuthorize`

```
@PreAuthorize("...")
@PostAuthorize("(returnObject == null) " +
        "or (returnObject.departmentCode == principal.account.departmentCode)")
public Account findOne(String username) {
    return accountRepository.findOne(username);
}
```

The part wherein the return value of the method is accessed from the Expression is discussed here. Specifically, "`returnObject.departmentCode`" part accesses the return value. The return value of the method can be accessed by specifying `returnObject` in the Expression.

Authorization for JSP screen fields

Spring Security can apply authorization process for JSP screen fields by using a JSP tag library.

Here, a method to apply authorization process for accessing JSP screen fields is explained using a simple definition as an example.

Defining an access policy

A condition to allow display (access policy) is defined in JSP while defining an access policy for JSP screen fields using a JSP tag library.

- Definition example of access policy

```
<%@ taglib prefix="sec" uri="http://www.springframework.org/security/tags" %>

<!-- (1) -->
<sec:authorize access="hasRole('ADMIN')"> <!-- (2) -->
    <h2>Admin Menu</h2>
    <!-- omitted -->
</sec:authorize>
```

Sr. No.	Description
(1)	Enclose part for which an access policy is to be applied with `<sec:authorize>` tag.
(2)	Define an access policy in `access` attribute. Here, an access policy - “allow display in case of administrator” is defined.

Linking with the access policy specified in the Web resource

When an access policy is to be defined for buttons or links (screen fields associated with the requests to the server), it is linked with an access policy defined in the Web resource for request. `url` attribute of `<sec:authorize>` tag is used for linking with access policy specified in the Web resource.

JSP process implemented in `<sec:authorize>` tag is executed only when Web resource specified in `url` attribute can be accessed.

- Example of linking with access policy defined in Web resource

```
<ul>
    <!-- (1) -->
    <sec:authorize url="/admin/accounts"> <!-- (2) -->
        <li>
            <a href="<c:url value='/admin/accounts' />">Account Management</a>
        </li>
    </sec:authorize>
</ul>
```

Sr. No.	Description
(1)	Enclose part which outputs button or link with `<sec:authorize>` tag.
(2)	Specify a URL for accessing a Web resource in `url` attribute of `<sec:authorize>` tag. Here, an access policy - “Allow display when a Web resource allocated with `"/admin/accounts"` URL can be accessed” is defined however it is not necessary to be directly aware of the access policy defined in Web resource.

Note: Specifying a policy by HTTP method

When a different access policy is specified by HTTP method while defining an access policy of Web resource, the definition to be linked must be identified by using `method` attribute of `<sec:authorize>` tag.

Warning: Notes related to display control

When display for a button or link is to be controlled, it should always be linked with an access policy defined in the Web resource.

If an access policy is not defined independently for a Web resource by specifying a direct access policy for button or link, an unauthorized access such as accessing a URL directly cannot be prevented.

Storing determination results of authorization process in the variable

Determination results of authorization process called by using `<sec:authorize>` tag can be reused by storing in the variable.

- Implementation example of JSP

```
<sec:authorize url="/admin/accounts"
               var="hasAccountsAuthority"/> <!-- (1) -->

<c:if test="${hasAccountsAuthority}"> <!-- (2) -->
    <!-- omitted -->
</c:if>
```

Sr. No.	Description
(1)	Specify a variable name for storing determination results in `var` attribute. When access is allowed, `true` is set in the variable.
(2)	Refer variable value and implement display process.

Response at the time of authorization error

Spring Security handles the errors as shown in the following flow and controls the response when access to a resource is denied.

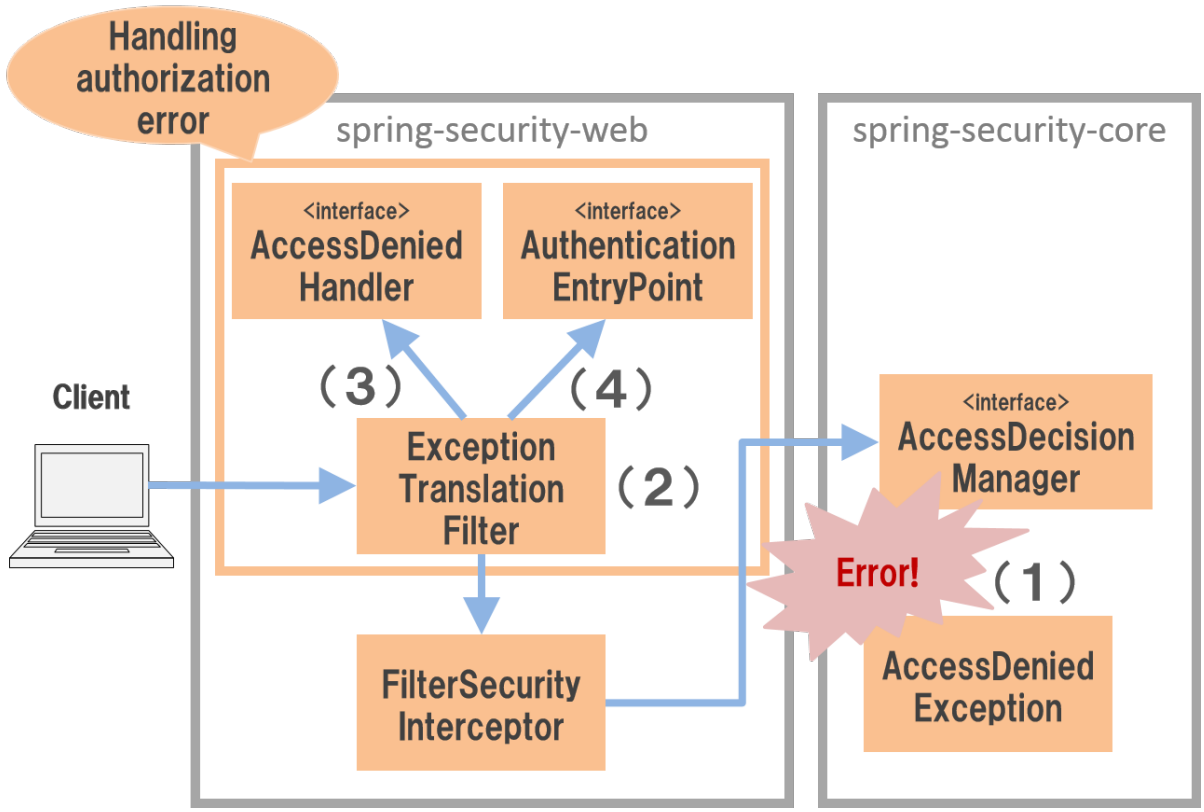


Figure.9.8 **Mechanism for handling authorization error**

Sr. No.	Description
(1)	Spring Security generates a `AccessDeniedException` for denying access to a resource or a method.
(2)	`ExceptionTranslationFilter` class catches `AccessDeniedException` and performs error handling by calling methods of `AccessDeniedHandler` or `AuthenticationEntryPoint` interface.
(3)	Perform error handling by calling a method of `AccessDeniedHandler` interface in case of an access by authenticated user.
(4)	Perform error handling by calling a method of `AuthenticationEntryPoint` interface in case of an access by unauthenticated user.

AccessDeniedHandler

`AccessDeniedHandler` interface handles the error responses when the access is denied to authenticated users. Spring Security offers following classes as the implementation class of `AccessDeniedHandler` interface.

Table.9.30 **Implementation class of AccessDeniedHandler provided by Spring Security**

Class name	Description
`AccessDeniedHandlerImpl`	Set 403 (Forbidden) in HTTP response code and move to specified error page. When error page is not specified, send error response (`HttpServletResponse#sendError`) by setting 403 (Forbidden) in HTTP response code.
`InvalidSessionAccessDeniedHandler`	Delegate the process to implementation class of `InvalidSessionStrategy` interface. This class is used when the settings to detect a session timeout is enabled by using CSRF countermeasure and session management function at the time when CSRF token does not exist in the session (a session timeout has occurred).
`DelegatingAccessDeniedHandler`	Map implementation classes of `AccessDeniedException` and `AccessDeniedHandler` interface and delegate the process to the implementation class of `AccessDeniedHandler` interface corresponding to `AccessDeniedException` thus occurred. `InvalidSessionAccessDeniedHandler` is called by using this mechanism.

In the default setting of Spring Security, `AccessDeniedHandlerImpl`is used wherein an error page is not specified.

AuthenticationEntryPoint

`AuthenticationEntryPoint` interface handles the error responses when the access is denied to unauthenticated users. Spring Security offers following class as an implementation class of `AuthenticationEntryPoint` interface.

Table.9.31 **Main implementation class of AuthenticationEntryPoint offered by Spring Security**

Class Name	Description
`LoginUrlAuthenticationEntryPoint`	Display a login form for form authentication.
`BasicAuthenticationEntryPoint`	Send error response for Basic authentication. Specifically, set 401 (Unauthorized) in HTTP response code, `WWW-Authenticate` header for Basic authentication as a response header and send error response (`HttpServletResponse#sendError`).
`DigestAuthenticationEntryPoint`	Send error response for Digest authentication. Specifically, set 401 (Unauthorised) in the HTTP response code, `WWW-Authenticate` header for Digest authentication as a response header and send error response (`HttpServletResponse#sendError`).
`Http403ForbiddenEntryPoint`	Set 403 (Forbidden) in HTTP response code and send error response (`HttpServletResponse#sendError`).
`DelegatingAuthenticationEntryPoint`	Map implementation class of `RequestMatcher` and `AuthenticationEntryPoint` interface, and delegate the process to implementation class of `AuthenticationEntryPoint` interface corresponding to HTTP request.

Implementation class of `AuthenticationEntryPoint` interface corresponding to authentication method is used in the default setup of Spring Security.

Transition destination at the time of authorization error

In the default setting of Spring Security, when the access is denied to the authenticated user, an error page of application server is displayed. If error page of application server is displayed, it is likely to result in system degradation. Hence it is recommended to display an appropriate error screen. Error page can be specified by defining a bean as below.

- Definition example of spring-security.xml

```
<sec:http>
    <!-- omitted -->
    <sec:access-denied-handler
        error-page="/WEB-INF/views/common/error/accessDeniedError.jsp" /> <!-- (1)␣
↪-->
    <!-- omitted -->
</sec:http>
```

Sr. No.	Description
(1)	Specify an error page for authorization error in `error-page` attribute of `<sec:access-denied-handler>` tag.

Tip: Using error page function of servlet container

Error page for authorization error can also be specified by using an error page function of servlet container.

When error page function of servlet container is used, error page is specified by using `<error-page>` tag of `web.xml`.

```
<error-page>
    <error-code>403</error-code>
    <location>/WEB-INF/views/common/error/accessDeniedError.jsp</location>
</error-page>
```

9.3.3 How to extend

This section explains customization points and extension methods offered by Spring Security.

Spring Security provides a lot of customization points of which only a few will be introduced here. This section will focus on some typical customization points.

Response at the time of authorization error (Authenticated user)

Here, a method to customise the operation when an access from authenticated user is denied, is explained.

Applying AccessDeniedHandler

When the requirements are not met only by customising default operation provided by Spring Security, implementation class of `AccessDeniedHandler` interface can be directly applied.

For example, when an authorization error occurs in Ajax request (REST API etc), error information is displayed in JSON format instead of displaying an error page (HTML). In such a case, an implementation class of `AccessDeniedHandler` interface can be created and applied in Spring Security.

- How to create an implementation class of AccessDeniedHandler interface

```
public class JsonDelegatingAccessDeniedHandler implements AccessDeniedHandler {

    private final RequestMatcher jsonRequestMatcher;
    private final AccessDeniedHandler delegateHandler;

    public JsonDelegatingAccessDeniedHandler(
            RequestMatcher jsonRequestMatcher, AccessDeniedHandler␣
↪delegateHandler) {
        this.jsonRequestMatcher = jsonRequestMatcher;
        this.delegateHandler = delegateHandler;
    }

    public void handle(HttpServletRequest request, HttpServletResponse response,
                       AccessDeniedException accessDeniedException)
            throws IOException, ServletException {
        if (jsonRequestMatcher.matches(request)) {
            // response error information of JSON format
            response.setStatus(HttpServletResponse.SC_FORBIDDEN);
            // omitted
        } else {
            // response error page of HTML format
            delegateHandler.handle(
                    request, response, accessDeniedException);
        }
    }

}
```

- Definition example of spring-security.xml

```
<!-- (1) -->
<bean id="accessDeniedHandler"
      class="com.example.web.security.JsonDelegatingAccessDeniedHandler">
    <constructor-arg>
```

```
        <bean class="org.springframework.security.web.util.matcher.
↪AntPathRequestMatcher">
            <constructor-arg value="/api/**"/>
        </bean>
    </constructor-arg>
    <constructor-arg>
        <bean class="org.springframework.security.web.access.
↪AccessDeniedHandlerImpl">
            <property name="errorPage"
                      value="/WEB-INF/views/common/error/accessDeniedError.jsp"/>
        </bean>
    </constructor-arg>
</bean>

<sec:http>
    <!-- omitted -->
    <sec:access-denied-handler ref="accessDeniedHandler" />  <!-- (2) -->
    <!-- omitted -->
</sec:http>
```

Sr. No.	Description
(1)	Define a bean for implementation class of `AccessDeniedHandler` interface and register in DI container.
(2)	Specify a bean for `AccessDeniedHandler` in `ref` attribute of `<sec:access-denied-handler>` tag.

Response at the time of authorization error (Unauthenticated user)

Here, a method to customise operation when access is denied to an unauthenticated user, is explained.

Applying AuthenticationEntryPoint for each request

Similarly to authenticated user, when authorization error occurs in Ajax request (REST API etc), error information is displayed in JSON format instead of displaying it in login page (HTML). In such a case, implementation class of `AuthenticationEntryPoint` interface for each pattern of request is applied in Spring Security.

- Definition example of spring-security.xml

```
<!-- (1) -->
<bean id="authenticationEntryPoint"
      class="org.springframework.security.web.authentication.
↪DelegatingAuthenticationEntryPoint">
    <constructor-arg>
```

```
        <map>
            <entry>
                <key>
                    <bean class="org.springframework.security.web.util.matcher.
↪AntPathRequestMatcher">
                        <constructor-arg value="/api/**"/>
                    </bean>
                </key>
                <bean class="com.example.web.security.JsonAuthenticationEntryPoint
↪"/>
            </entry>
        </map>
    </constructor-arg>
    <property name="defaultEntryPoint">
        <bean class="org.springframework.security.web.authentication.
↪LoginUrlAuthenticationEntryPoint">
            <constructor-arg value="/login"/>
        </bean>
    </property>
</bean>

<sec:http entry-point-ref="authenticationEntryPoint"> <!-- (2) -->
    <!-- omitted -->
</sec:http>
```

Sr. No.	Description
(1)	Define a bean for implementation class of `AuthenticationEntryPoint` interface and register in DI container. Here, an implementation class of `AuthenticationEntryPoint` interface is applied for each pattern of request by using `DelegatingAuthenticationEntryPoint` class offered by Spring Security.
(2)	Specify a bean for `AuthenticationEntryPoint`in `entry-point-ref` attribute of `<sec:http>` tag.

Note: AuthenticationEntryPoint applied by default

When the implementation class of `AuthenticationEntryPoint` interface corresponding to request is not specified, the implementation class of `AuthenticationEntryPoint` interface defined by Spring Security by default is used. When form authentication is used as an authentication method, `LoginUrlAuthenticationEntryPoint` class is used and login form is displayed.

Role hierarchy

A hierarchy can be set for a role in the authorization process.

Higher level roles can also access the resources for which access is granted to the lower level roles. When the role relation is complex, hierarchy relation must also be established.

For example, when a hierarchy relation is established wherein "ROLE_ADMIN" is a higher role and "ROLE_USER" is a lower role, and if an access policy is set as below, user with "ROLE_ADMIN" rights can access a path under `"/user"`(a path which can be accessed by a user with "ROLE_USER" rights).

- Definition example of spring-security.xml

```
<sec:http>
    <sec:intercept-url pattern="/user/**" access="hasAnyRole('USER')" />
    <!-- omitted -->
</sec:http>
```

Setting hierarchy relation

Role hierarchy relation is resolved by using implementation class of `org.springframework.security.access.hierarchicalroles.RoleHierarchy` interface.

- Definition example of spring-security.xml

```
<bean id="roleHierarchy"
    class="org.springframework.security.access.hierarchicalroles.RoleHierarchyImpl
↪"> <!-- (1) -->
    <property name="hierarchy"> <!-- (2) -->
        <value>
            ROLE_ADMIN > ROLE_STAFF
            ROLE_STAFF > ROLE_USER
        </value>
    </property>
</bean>
```

Sr. No.	Description
(1)	Specify `org.springframework.security.access.hierarchicalroles.RoleHierarchyImpl` class. `RoleHierarchyImpl` is a default implementation class offered by Spring Security.
(2)	Define the hierarchy relation in `hierarchy` property. Format : [Higher role] > [Lower role] In the example above, STAFF can also access resources authorised by USER. ADMIN can also access resources authorised by USER and STAFF.

Applying authorization process of Web resource

A method to apply role hierarchy to authorization process for Web resource and JSP screen fields is explained.

- Definition example of spring-security.xml

```
<!-- (1) -->
<bean id="webExpressionHandler"
    class="org.springframework.security.web.access.expression.
↪DefaultWebSecurityExpressionHandler">
    <property name="roleHierarchy" ref="roleHierarchy"/>  <!-- (2) -->
</bean>

<sec:http>
    <!-- omitted -->
    <sec:expression-handler ref="webExpressionHandler" />  <!-- (3) -->
</sec:http>
```

Sr. No.	Explanation
(1)	Define a Bean for `org.springframework.security.web.access.expression.DefaultWebSecurityExpressi`
(2)	Specify a Bean for implementation class of `RoleHierarchy` interface in `roleHierarchy` property.
(3)	Specify a Bean for implementation class of `org.springframework.security.access.expression.SecurityExpressionHandler` interface in `ref` attribute of `<sec:expression-handler>` tag.

Applying authorization process of method

A method to apply role hierarchy to authorization process for Java method is explained.

- Definition example of spring-security.xml

```
<bean id="methodExpressionHandler"
    class="org.springframework.security.access.expression.method.
↪DefaultMethodSecurityExpressionHandler"> <!-- (1) -->
    <property name="roleHierarchy" ref="roleHierarchy"/> <!-- (2) -->
</bean>

<sec:global-method-security pre-post-annotations="enabled">
    <sec:expression-handler ref="methodExpressionHandler" /> <!-- (3) -->
</sec:global-method-security>
```

Sr. No.	Description
(1)	Define a Bean for `org.springframework.security.access.expression.method.DefaultMethodSecurityEx`
(2)	Specify a Bean for implementation class of `RoleHierarchy` interface in `roleHierarchy` property.
(3)	Specify a Bean for implementation class of `org.springframework.security.access.expression.SecurityExpressionHandler` interface in `ref` attribute of `<sec:expression-handler>` tag.

9.4 Session Management

9.4.1 Overview

This chapter explains the “Security measures required for session management in Web applications” and “Session related functions provided by Spring Security”.

Security measures at the time of using a session

Generally, countermeasures are required for the following attacks for session management in Web application.

Countermeasure	Description
Session hijacking attack	An attack wherein the session ID is stolen using communication eavesdropping, analogy from the regularity and cross-site scripting and using the system with stolen ID by pretending to be the user.
Session fixation attack	The attacker allows the other user to log in to the system by using the previously issued session ID. The attacker uses the system by pretending to be the logged in user.

Countermeasures for session hijacking attacks

Countermeasures for session hijacking attacks enable you to only prevent the session ID from being stolen. If it is stolen, application server cannot determine whether the request is from an authorized user or from the attacker.

Following countermeasures are necessary to protect the application from such session hijacking attacks.

Table.9.32 **Countermeasures for session hijacking attacks**

Countermeasures	Description
Create a session ID that is difficult to presume.	Use a random value for Session ID that is difficult to presume (Secure) instead of a value such as sequential number that can be easily cracked. Basically, the mechanism provided by the application server to create session ID should be used.
Encrypt communication using HTTPS	Encrypt the communication with HTTPS protocol that includes exchange of information which can cause a lot of trouble when stolen. Since communication eavesdropping can be done easily by using a free software etc., it is important to keep it encrypted to prevent it from being decrypted even if it is eavesdropped.
Link Session ID using a Cookie	While linking the session ID between client and server, set it to be linked by using a Cookie and disable the URL Rewriting function.
Specify `HttpOnly` attribute of Cookie	If you specify ‘‘ HttpOnly‘‘ attribute of Cookie, session ID cannot be stolen using cross site scripting since Cookie cannot be accessed from JavaScript.
Specify `Secure` attribute in Cookie	If you specify ‘‘ Secure‘‘ attribute in Cookie, the risk of session ID getting stolen when you accidentally use HTTP communication is reduced since Cookie is sent to the server only during HTTPS communication.

Note: URL Rewriting

URL Rewriting is a mechanism to retain the session with the client that cannot use Cookie. Particularly, the Session ID between client and server is linked by including Session ID as a request parameter of the URL.

- Example of URL Rewriting

```
http://localhost:8080/;jsessionid=7E6EDE4D3317FC5F14FD912BEAC96646
```

`jsessionid=7E6EDE4D3317FC5F14FD912BEAC96646` is a part of Session ID for which URL Rewriting

is done. In the Servlet API specifications, URL rewriting is likely to be done when the following methods are called. These methods are also called in the JSP tag library provided by JSTL and Spring.

- `HttpServletResponse#encodeURL(String)`
- `HttpServletResponse#encodeRedirectURL(String)`

If URL Rewriting is done, session ID in the URL is exposed resulting in higher risk of session ID being stolen. Therefore, it is recommended to disable the URL Rewriting function of Servlet container while supporting only the clients that can use the Cookie.

Countermeasures for Session fixation attack

Following countermeasures are necessary to protect the application from Session fixation attack.

Table.9.33 **Countermeasures for Session fixation attack**

Countermeasure	Description
Disable URL Rewriting function.	If URL Rewriting function is disabled, session ID issued previously cannot be used by the attacker and a new session is started.
Change session ID after login	By changing the session ID after login, the session ID issued previously cannot be used by the attacker.

Session management function provided by Spring Security

Following functions related to session are mainly provided in Spring Security.

Table.9.34 **Session related functions**

Function	Description
Security measures	Countermeasures for attacks using session ID of session hijacking attacks.
Lifecycle control	Function to control the lifecycle of the session from generation to discard of a session.
Timeout control	Function to discard a session due to timeout.
Multiple login control	Function to control a session if the same user logs in for multiple times.

9.4.2 How to use

Countermeasures for Session hijacking attacks

The method to disable a URL rewriting function and link session ID using a Cookie is explained.

Disabling URL Rewriting function by Spring Security

Spring Security provides a mechanism to disable URL Rewriting and this function is applied by default. When it is necessary to support the clients who cannot use a Cookie, a Bean is defined so as to authorize URL rewriting.

- Definition example of spring-security.xml

```
<sec:http disable-url-rewriting="false"> <!-- Enable URL Rewriting by specifying
↪'false' -->
```

Sr. No.	Description
(1)	In Spring Security, since the value of ‘‘ disable-url-rewriting‘‘ is ‘‘ true‘‘ by default, URL Rewriting is not performed. Set `false` in `disable-url-rewriting` attribute of `<sec:http>` element to enable URL Rewriting.

Disabling URL Rewriting function by Servlet Container

A session can be managed securely using the standard specifications of Servlet.

- Definition example of web.xml

```
<session-config>
    <cookie-config>
        <http-only>true</http-only> <!-- (1)  -->
    </cookie-config>
    <tracking-mode>COOKIE</tracking-mode> <!-- (2) -->
</session-config>
```

Sr. No.	Description
(1)	Specify `true` in `<http-only>` element while assigning `HttpOnly` attribute in Cookie. Default value is set to ‘‘ true‘‘ based on the application server used.
(3)	Specify `COOKIE` in `<tracking-mode>` element to disable the URL Rewriting function.

Although it is omitted from the definition example mentioned above, `Secure` attribute can be assigned for the Cookie by adding `<secure>true</secure>` in `<cookie-config>`. However, to secure the cookie, the method wherein a middleware (SSL accelerators and Web server etc.) that performs HTTPS communication with the client is assigned, is used instead of specifying in ‘‘ web.xml‘‘ .

In actual system development, HTTPS is used very rarely in the local development environment. Also, even in the production environment, HTTPS is used for SSL accelerators and communication with Web server and there are many cases where communication with application server is carried out using HTTP. If ‘‘ Secure‘‘ attribute is specified in ‘‘ web.xml‘‘ under such environment, `web.xml` and `web-fragment.xml` will be provided for each execution environment. It is not recommended since file management becomes complicated.

Applying Session management function

A method to use session management function of Spring Security is explained. Define a bean as shown below to use the session management function process of Spring Security.

- Definition example of spring-security.xml

```
<sec:http>
    <!-- omitted -->
    <sec:session-management /> <!-- (1) -->
    <!-- omitted -->
</sec:http>
```

Sr. No.	Description
(1)	Specify `<sec:session-management>` element as the child element of `<sec:http>` element. Session management function is applied when `<sec:session-management>` element is specified.

Countermeasures for Session fixation attack

Spring Security provides the following four options to change the session ID when login is successful, as the countermeasures against session fixation attack.

Table.9.35 **Options for Session fixation attack countermeasures**

Options	Description
`changeSessionId`	Change the session ID using `HttpServletRequest#changeSessionId()` added in Servlet 3.1. (This is the default operation from Servlet 3.1 container onwards)
`migrateSession`	Discard the session that was used before login and create a new session. The objects stored in the session before login are transferred to the new session when this option is used. (This is the default operation in Servlet 3.0 container and earlier versions)
`newSession`	This option changes the session ID in the same way as ‘‘ migrateSession‘‘ , however, the objects stored before login are not transferred to the new session.
`none`	Spring Security does not change the session ID.

Define a bean as shown below to change the default operation.

- Definition example of spring-security.xml

```
<sec:session-management
        session-fixation-protection="newSession"/> <!-- (1) -->
```

Sr. No.	Description
(1)	Specify the countermeasures for session fixation attack in `session-fixation-protection` attribute of `<sec:session-management>` element.

Controlling session lifecycle

Spring Security uses HTTP session for sharing objects such as authentication information across requests. The lifecycle of the session (Generating and discarding a session) is controlled in the Spring Security process.

Note: Session information storage destination

HTTP session is used in the default implementation provided by Spring Security, however, the architecture also enables storing the objects in other than HTTP session (Database and key-value store etc.).

Generating a session

The guidelines by which a session can be generated and used in the Spring Security process can be selected from the following options.

Table.9.36 **Guidelines to generate a session**

Option	Description
`always`	Generate a new session when the session does not exist. If this option is specified, the session is generated even though it is not used in the Spring Security process.
`ifRequired`	Generate and use a new session at the time of storing object in the session when the session does not exist.(Default operation)
`never`	If the session does not exist, session is not generated and used. However, use the session if it already exists.
`stateless`	Session is not generated and used irrespective of whether it exists.

Define a bean as shown below to change the default behaviour.

- Definition example of spring-security.xml

```
<sec:http create-session="stateless"> <!-- (1) -->
    <!-- omitted -->
</sec:http>
```

Sr. No.	Description
(1)	Specify creation guidelines for the session to be changed in `create-session` attribute of `<sec:http>` element.

Discarding a session

Spring Security discards the session at the following timings.

- When logout process is executed
- When authentication process is successful (Session is discarded if `migrateSession` or `newSession`

is used as the countermeasure for Session fixation attack)

Controlling session timeout

When an object is to be stored in the session, it is a common practice to ensure that session of the user who is idle for a certain period of time is automatically discarded by specifying an appropriate session time-out value.

Specifying session timeout

Specify session timeout for Servlet container. In some cases, the specification method independent of server may be provided depending on the application server, however, the specification methods stated in Servlet standard specifications are described here.

- Definition example of web.xml

```
<session-config>
    <session-timeout>60</session-timeout> <!-- (1) -->
    <!-- omitted -->
</session-config>
```

Sr. No.	Description
(1)	Specify an appropriate timeout value (in minutes) in `<session-timeout>` element. If a timeout value is not specified, the default value provided by Servlet Container is used. Also, if a value of 0 or less than 0 is specified, the session timeout function of Servlet container is disabled.

Detection of request with invalid session

Spring Security provides a function to detect a request with an invalid session. Most of the requests handled as invalid sessions are requests after session timeout. By default, this functionality is disabled, however, it can be enabled by defining a bean as shown below.

- Definition example of spring-security.xml

```
<sec:session-management
        invalid-session-url="/error/invalidSession"/>
```

Sr. No.	Description
(1)	Specify the path for redirect destination when a request with invalid session is detected in `invalid-session-url` attribute of `<sec:session-management>` element.

Specifying Exclusion path

If the function to detect a request with invalid session is enabled, all the requests that pass through Servlet filter of Spring Security are checked. Therefore, check is performed even for pages that can be accessed when session is in invalid state.

This operation can be changed by defining a separate bean for the path to be excluded from the check target. For example, a bean is defined as shown below to specify the path to open the top page (`"/"`) in the exclusion path.

- Definition example of spring-security.xml

```
<!-- (1) -->
<sec:http pattern="/"> <!-- (2) -->
    <sec:session-management />
</sec:http>

<!-- (3) -->
<sec:http>
    <!-- omitted -->
    <sec:session-management
            invalid-session-url="/error/invalidSession"/>
    <!-- omitted -->
</sec:http>
```

Sr. No.	Description
(1)	Add new element <sec:http> element to create SecurityFilterChain which is to be applied to ("/") path to open the top page.
(2)	Specify the path pattern to apply SecurityFilterChain created using <sec:http> element of (1). Ant-style path notation and regular expression are the two formats that can be specified for the path pattern. By default, it is handled as Ant-style path pattern. Note that, it is also possible to directly specify ‘‘ RequestMatcher‘‘ object without the path pattern.
(3)	Define <sec:http> element to create SecurityFilterChain to be applied to a path that is not defined separately. It should be defined below the <sec:http> element for separate definition. This is because definition sequence of <sec:http> element is the priority sequence of SecurityFilterChain.

Controlling multiple logins

Spring Security provides a function to control multiple logins using the same user name (login ID). By default this function is disabled. However, it can be enabled by using *Enabling session lifecycle detection*.

Warning: Constraints in multiple login control

In the default implementation provided by Spring Security, following constraints are observed since session information of each user is managed within the application server memory.

First constraint is that the default implementation cannot be used in the system wherein multiple application servers are started concurrently. If multiple application servers are to be used concurrently, it is necessary to create the implementation class to manage session information of each user in the shared area such as database or key-value store (Cache server).

The second constraint is that if session information is restored when application server is stopped or re-started, it may not operate normally. Since it has a function to restore the session state at the time of stop or restart

depending on the application server to be used, an inconsistency may appear in the actual session state and the session information managed by Spring Security. One of the following actions should be taken in case of a likely inconsistency.

- Do not restore session state of the application server.
- Implement a mechanism to restore the session information of Spring Security.
- Store the object in other than HTTP session (Database or key-value store etc.)

This section introduces a method to use default implementation of Spring Security. HTTP session is used in the default implementation provided by Spring Security, however, the architecture also enables storing objects in other than HTTP session (Database or key-value store etc.). However, please note that the method introduced here is the **Implementation method with the constraints of the Warning mentioned above**, during the application.

Todo

The information about the implementation method that does not use in-memory will be added later.

Enabling session lifecycle detection

The function to control multiple login manages the session state for each user by using *session lifecycle (Generating and discarding session) detection mechanism*. Therefore, while using multiple login control function, `HttpSessionEventPublisher` class provided by Spring Security must be registered in Servlet Container.

- Definition example of web.xml

```
<listener>
    <!-- (1) -->
    <listener-class>
        org.springframework.security.web.session.HttpSessionEventPublisher
    </listener-class>
</listener>
```

Sr. No.	Description
(1)	Register `HttpSessionEventPublisher` as Servlet Listener.

Preventing multiple logins (Pre-measure)

A bean is defined as shown below if multiple login is to be prevented by generating an authentication error in case of the users who have already logged in with the same user name (login ID).

- Definition example of bean definition file

```
<sec:session-management>
    <sec:concurrency-control
            max-sessions="1"
            error-if-maximum-exceeded="true"/> <!-- (1) (2) -->
</sec:session-management>
```

Sr. No.	Description
(1)	Specify the number of sessions for which concurrent logins are allowed in `max-sessions` attribute of `<sec:concurrency-control>` element. Specify '' 1'' to prevent multiple logins.
(2)	Specify the operation to be performed when the number of sessions to which user can login concurrently is exceeded, in `error-if-maximum-exceeded` attribute of `<sec:concurrency-control>` element. Specify `true` if a user who has already logged in is handled as a valid user.

Preventing multiple logins (Post-measure)

In case of the users who have already logged in with the same user name (login ID), a bean is defined as below if multiple login is to be prevented by invalidating the users who are already logged in.

- Definition example of spring-security.xml

```
<sec:session-management>
    <sec:concurrency-control
            max-sessions="1"
            error-if-maximum-exceeded="false"
            expired-url="/error/expire"/> <!-- (1) (2) -->
</sec:session-management>
```

Sr. No.	Description
(1)	Specify the operation to be performed when the number of sessions a user can login concurrently has exceeded, in `error-if-maximum-exceeded` attribute of `<sec:concurrency-control>` element. Specify ‘‘ false‘‘ if a new logged-in user is handled as a valid user.
(2)	Specify the path for redirect destination when a request from an invalidated user is detected in `expired-url` attribute of `<sec:concurrency-control>` element. This is because definition sequence of `<sec:http>` element is the priority sequence of `SecurityFilterChain`.

9.5 CSRF Countermeasures

9.5.1 Overview

This chapter explains Cross site request forgeries (hereafter referred to as CSRF) countermeasures function offered by Spring Security.

CSRF is an attack that forces a user to perform unwanted actions on a different website in which the user is logged in. by implementing a script in the Website or by automatic transfer (HTTP redirect).

Following are the methods to prevent server side CSRF attack.

- Embedding confidential information (token)
- Re-entering password
- 'Referer' check

CSRF countermeasures function handles the malicious request sent from the Web page provided by the attacker as an invalid request. Following methods can be used to attack if CSRF countermeasures are not implemented in a Web application.

- User logs in to the Web application for which CSRF countermeasures are not implemented.
- User opens the Web page provided by the attacker under the skilful guidance of the attacker.
- The Web page prepared by the attacker sends a forged request to the Web application for which CSRF countermeasures have not been implemented, using techniques such as automatic submission of form.
- Web applications for which CSRF countermeasures are not implemented handle forged request by the attacker as a legitimate request.

Tip: In OWASP[*1], the method to use token pattern is recommended.

Note: CSRF countermeasures at the time of login

CSRF countermeasures should be implemented not only for the login request but also for the login process. If CSRF countermeasures are not implemented for login process, user is forced to login using the account created by the attacker even before the user realizes it and there is a possibility of the operation history of login being stolen.

[*1] OWASP is an abbreviation of Open Web Application Security Project. It is a not-for-profit international organization dedicated to enable organizations to develop and maintain applications that can be trusted. It advocates measures such as effective approach etc. with respect to security. https://www.owasp.org/index.php/Main_Page

Warning: CSRF measures at the time of the multi-part request (file upload)

About CSRF measures during file upload, *file upload Servlet Filter setting*should be followed.

CSRF countermeasures of Spring Security

Spring Security issues a fixed token value (CSRF token) generated randomly for each session and sends the issued CSRF token as a request parameter (hidden field in the HTML form). Accordingly, a system to determine whether the request is from a normal Web page or from the Web page created by the attacker, is adopted.

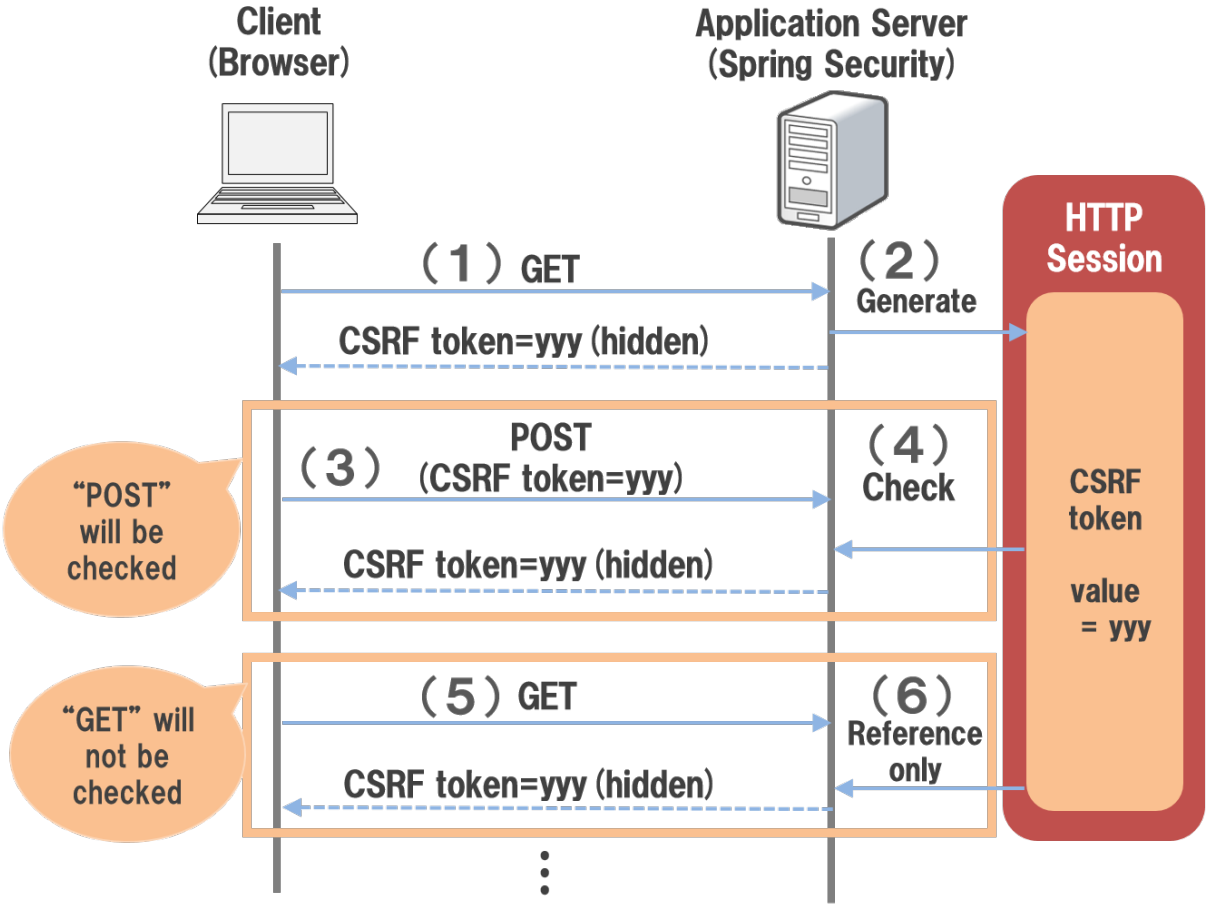


Figure.9.9 **CSRF countermeasures system of Spring Security**

Sr. No.	Description
(1)	Client accesses the application server using the HTTP GET method.
(2)	Spring Security generates a CSRF token and stores it in HTTP session. The generated CSRF token links with the client using the hidden tag of HTML form.
(3)	The client sends a request to the application server by clicking a button on the HTML form. Since the CSRF token is embedded in a hidden field in the HTML form, CSRF token value is sent as a request parameter.
(4)	Spring Security checks if the CSRF token value specified in the request parameter and the CSRF token value retained in the HTTP session are same when it is accessed using HTTP POST method. If the token value does not match, an error is thrown as an invalid request (request from the attacker).
(5)	Client accesses the application server using the HTTP GET method.
(6)	Spring Security does not check the CSRF token value when it is accessed using GET method.

Note: CSRF token using Ajax

Since Spring Security can set the CSRF token value in request header, it is possible to implement CSRF countermeasures for the requests for Ajax etc.

Target request of token check

In the default implementation of Spring Security, check CSRF token for the request that uses the following HTTP methods.

- POST
- PUT

- DELETE
- PATCH

Note: Reason for not checking CSRF token

GET, HEAD, OPTIONS, TRACE methods are not checked as these methods are not used to implement the request to change the application status.

9.5.2 How to use

Applying CSRF countermeasures

By using the `RequestDataValueProcessor`implementation class for CSRF token, the token can be automatically inserted as 'hidden' field using `<form:form>`tag of Spring tag library.

- Setting example of spring-mvc.xml

```
<bean id="requestDataValueProcessor"
    class="org.terasoluna.gfw.web.mvc.support.CompositeRequestDataValueProcessor">
↪<!-- (1)  -->
    <constructor-arg>
        <util:list>
            <bean
                class="org.springframework.security.web.servlet.support.csrf.
↪CsrfRequestDataValueProcessor" /> <!-- (2)  -->
            <bean
                class="org.terasoluna.gfw.web.token.transaction.
↪TransactionTokenRequestDataValueProcessor" />
        </util:list>
    </constructor-arg>
</bean>
```

Sr. No.	Description
(1)	Define a bean for `org.terasoluna.gfw.web.mvc.support.CompositeRequestDataValueProcessor` for which multiple `org.springframework.web.servlet.support.RequestDataValueProcessor` provided by common library can be defined.
(2)	Set the bean definition `org.springframework.security.web.servlet.support.csrf.CsrfRequestDataValuePro` as the first argument of constructor.

CSRF countermeasures function is enabled as a default by the settings described above from Spring Security 4.0 and subsequent versions. Therefore, CSRF countermeasures function should be disabled explicitly if you do not want to use it.

Define a bean as given below if CSRF countermeasures function is not to be used.

- Definition example of spring-security.xml

```
<sec:http>
    <!-- omitted -->
    <sec:csrf disabled="true"/> <!-- Disabled by setting true in the 'disabled'␣
↪attribute -->
    <!-- omitted -->
</sec:http>
```

Linking CSRF token value

Spring Security provides the following two methods to link the CSRF token value between the client and server.

- Output the CSRF token value as a hidden field in the HTML form and link as a request parameter
- Output CSRF token information as HTML meta tag and link by setting the token value in the request header at the time of Ajax communication

Coordination by using Spring MVC

Spring Security has provided several components to coordinate with Spring MVC. How to use the component to coordinate with CSRF countermeasures function is described below.

Auto output of hidden fields

Implement the following JSP while creating a HTML form.

- Implementation example of JSP

```
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>

<c:url var="loginUrl" value="/login"/>
<form:form action="${loginUrl}"> <!-- (1) -->
    <!-- omitted -->
</form:form>
```

Sr. No.	Description
(1)	Use <form:form> element provided by Spring MVC while creating an HTML form.

HTML form shown below is created when <form:form> element provided by Spring MVC is used.

- Output example of HTML

```
<form id="command" action="/login" method="post">
    <!-- omitted -->
    <!-- CSRF token value hidden field output by coordinating with Spring MVC␣
↪function -->
    <div>
        <input type="hidden"
               name="_csrf" value="63845086-6b57-4261-8440-97a3c6fa6b99" />
    </div>
</form>
```

Tip: CSRF token check value that is output

If `CsrfRequestDataValueProcessor` is used in Spring 4, CSRF token inserted <input type="hidden"> tag is output only if the value specified in `method` attribute of <form:form> tag matches with HTTP methods (HTTP methods other than GET, HEAD, TRACE, OPTIONS in Spring Security default implementation) of CSRF token check.

For example, when GET method is specified in `method` attribute as shown below, CSRF token inserted <input type="hidden"> tag is not output.

```
<form:form method="GET" modelAttribute="xxxForm" action="...">
    <%-- ... --%>
</form:form>
```

This is as per the following description

The unique token can also be included in the URL itself, or a URL parameter. However, such placement runs a greater risk that the URL will be exposed to an attacker, thus compromising the secret token.

in OWASP Top 10 and it helps in building a secure Web application.

Coordination while using HTML form

CSRF token value can also be linked using HTML form without using *Linkage with Spring MVC*. If you want to send a request using HTML form, output the CSRF token value as a hidden field in the HTML form and coordinate as a request parameter.

- Implementation example of JSP

```
<%@ taglib prefix="sec" uri="http://www.springframework.org/security/tags" %>

<form action="<c:url value="/login" />" method="post">
    <!-- omitted -->
    <sec:csrfInput /> <!-- (1) -->
    <!-- omitted -->
</form>
```

Sr. No.	Description
(1)	Specify `<sec:csrfInput>` element in `<form>` element of HTML.

The hidden fields are output as shown below if `<sec:csrfInput>` element provided by Spring Security is specified. CSRF token value is linked as a request parameter by displaying the hidden fields in the HTML form. By default, the name of the request parameter to link CSRF token value is `_csrf` .

- Output example of HTML

```
<form action="/login" method="post">
    <!-- omitted -->
    <!-- CSRF token value hidden field -->
    <input type="hidden"
           name="_csrf"
           value="63845086-6b57-4261-8440-97a3c6fa6b99" />
    <!-- omitted -->
</form>
```

Warning: Points to be noted while using GET method

When GET is used as the HTTP method, `<sec: csrfInput>`element should not be specified. If `<sec:csrfInput>` element is specified, there is a high risk of CSRF token value being stolen since CSRF

token value is included in the URL.

Coordination while using Ajax

If you want to send a request using Ajax, output the CSRF token information as HTML meta tag, and link by setting the token value fetched from meta tag in the request header at the time of Ajax communication.

First, output the CSRF token information in HTML meta tag by using the JSP tag library provided by Spring Security.

- Implementation example of JSP

```
<%@ taglib prefix="sec" uri="http://www.springframework.org/security/tags" %>

<head>
    <!-- omitted -->
    <sec:csrfMetaTags /> <!-- (1) -->
    <!-- omitted -->
</head>
```

Sr. No.	Description
(1)	Specify `<sec:csrfMetaTags>` element in `<head>` element of HTML.

If `<sec:csrfMetaTags>` element is specified, the meta tags are output as shown below. By default, the name of the request header to link CSRF token value is `X-CSRF-TOKEN`.

- Output example of HTML

```
<head>
    <!-- omitted -->
    <meta name="_csrf_parameter" content="_csrf" />
    <meta name="_csrf_header" content="X-CSRF-TOKEN" /> <!-- Header name -->
    <meta name="_csrf"
          content="63845086-6b57-4261-8440-97a3c6fa6b99" /> <!-- Token value -->
    <!-- omitted -->
</head>
```

Then, fetch the CSRF token information from meta tag using JavaScript and set the CSRF token value in the request header at the time of Ajax communication. (Implementation example using jQuery is described here)

- Implementation example of JavaScript

```
$(function () {
    var headerName = $("meta[name='_csrf_header']").attr("content"); // (1)
    var tokenValue = $("meta[name='_csrf']").attr("content"); // (2)
```

```
    $(document).ajaxSend(function(e, xhr, options) {
        xhr.setRequestHeader(headerName, tokenValue); // (3)
    });
});
```

Sr. No.	Description
(1)	Fetch the request header name for coordinating with the CSRF token value.
(2)	Fetch CSRF token value.
(3)	Set CSRF token value in request header.

Controlling transition destination in case of token check error

In order to control the transition destination in case of token check error, handle `AccessDeniedException` which is an exception that occurs in CSRF token check error and specify the transition destination corresponding to that exception.

Exception that occurs at the time of CSRF token check error is as follows.

Table.9.37 **Exception class used by CSRF token check**

Class name	Description
`InvalidCsrfTokenException`	Exception class to be used when the token value sent by the client does not match with the token value stored at server side (Mainly invalid request).
`MissingCsrfTokenException`	Exception class to be used when the token value is not stored at server side (Mainly session timeout).

It is possible to set the transition destination for each exception by handling the exception mentioned above using `DelegatingAccessDeniedHandler` class and assigning implementation class of `AccessDeniedHandler` interface respectively.

Define a Bean as shown below if you want to transit to an exclusive error page (JSP) at the time of CSRF token check error. (The following definition example is an excerpt from a blank project)

- Definition example of spring-security.xml

```
<sec:http>
    <!-- omitted -->
    <sec:access-denied-handler ref="accessDeniedHandler"/>  <!-- (1) -->
    <!-- omitted -->
</sec:http>

<bean id="accessDeniedHandler"
    class="org.springframework.security.web.access.DelegatingAccessDeniedHandler">
 <!-- (2) -->
    <constructor-arg index="0">  <!-- (3) -->
        <map>
            <!-- (4) -->
            <entry
                key="org.springframework.security.web.csrf.
↪InvalidCsrfTokenException">
                <bean
                    class="org.springframework.security.web.access.
↪AccessDeniedHandlerImpl">
                    <property name="errorPage"
                        value="/WEB-INF/views/common/error/invalidCsrfTokenError.
↪jsp" />
                </bean>
            </entry>
            <!-- (5) -->
            <entry
                key="org.springframework.security.web.csrf.
↪MissingCsrfTokenException">
                <bean
                    class="org.springframework.security.web.access.
↪AccessDeniedHandlerImpl">
                    <property name="errorPage"
                        value="/WEB-INF/views/common/error/missingCsrfTokenError.
↪jsp" />
                </bean>
            </entry>
        </map>
    </constructor-arg>
    <!-- (6) -->
    <constructor-arg index="1">
        <bean
            class="org.springframework.security.web.access.AccessDeniedHandlerImpl
↪">
            <property name="errorPage"
                value="/WEB-INF/views/common/error/accessDeniedError.jsp" />
        </bean>
    </constructor-arg>
```

```
</bean>
```

Sr. No.	Description
(1)	Specify the Bean name for `AccessDeniedHandler` to control each Exception in ref attribute of `<sec:access-denied-handler>` tag. If all the transition destinations in case of an error are on the same screen, the transition destination should be specified in `error-page` attribute. Refer to *Transition destination at the time of authorization error* when the exceptions are not handled by `<sec:access-denied-handler>`.
(2)	Using `DelegatingAccessDeniedHandler` , define the exception that occurred (`AccessDeniedException` subclass) and the exception handler (`AccessDeniedHandler` Implementation class).
(3)	Using the first argument of constructor, define the exception for which transition destination is to be specified separately (`AccessDeniedException` subclass) and the corresponding exception handler (`AccessDeniedHandler` implementation class) in Map format.
(4)	Specify sub class of `AccessDeniedException` in `key`. Specify `org.springframework.security.web.access.AccessDeniedHandlerImpl` which is an implementation class of `AccessDeniedHandler` , as the `value`. Specify the view to be displayed in `value` by specifying `errorPage` in `name` of `property`. Refer to *Controlling transition destination in case of token check error* for the Exception to be mapped.
(5)	Define if an Exception different from Exception of (4) is to be controlled. In this example, a different transition destination has been set in `InvalidCsrfTokenException`, `MissingCsrfTokenException` respectively.
(6)	Specify the exception handler (`AccessDeniedHandler` implementation class) at the time of default exception (subclass of `AccessDeniedException` not specified in (4)(5)) and the transition destination using second argument of constructor.

Note: Detection of request with an invalid session

When "*Detection of request with invalid session*" process of session management function is enabled, implementation class of `AccessDeniedHandler` interface to be linked with "*Detection of request with invalid session*" process is used for `MissingCsrfTokenException`.

Therefore, if `MissingCsrfTokenException` is thrown, redirect to path (`invalid-session-url`) specified while enabling "*Detection of request with invalid session*" process.

Note: When status code other than 403 is to be returned

If status code other than 403 is to be returned when the CSRF token in request does not match, it is necessary to create an independent AccessDeniedHandler which implements `org.springframework.security.web.access.AccessDeniedHandler` interface.

9.6 Coordinating with browser security countermeasure function

9.6.1 Overview

This chapter explains how to coordinate with the security countermeasure function provided by browser.

Main Web browser provides a few security countermeasure functions so that the functions provided by the browser are not affected. Some security countermeasure functions provided by the browser can control the operations by displaying response header of HTTP at the server side.

Spring Security provides a system to enhance security of Web application by offering function to output the security response header.

Note: Security risk

Even if the security response header is displayed, it does not guarantee 100% elimination of security risk. Ultimately, user should consider it as a support function to reduce the security risk.

Note that, the support status of security header varies depending on the browser.

Note: Overwriting HTTP header

HTTP header may be overwritten by the application even though the following settings are done.

Security headers supported by default

The following 7 response headers are supported by Spring Security by default.

- Cache-Control (Pragma, Expires)
- X-Frame-Options
- X-Content-Type-Options
- X-XSS-Protection
- Strict-Transport-Security
- Content-Security-Policy(Content-Security-Policy-Report-Only)
- Public-Key-Pins(Public-Key-Pins-Report-Only)

Tip: Support status of browser

Some browsers do not support handling these headers. Refer official site of the browser or the following pages.

- https://www.owasp.org/index.php/HTTP_Strict_Transport_Security (Strict-Transport-Security)
- https://www.owasp.org/index.php/Clickjacking_Defense_Cheat_Sheet (X-Frame-Options)
- https://www.owasp.org/index.php/List_of_useful_HTTP_headers (X-Content-Type-Options, X-XSS-Protection, Content-Security-Policy, Public-Key-Pins)

Cache-Control

Cache-Control header indicates a method to cache the contents. Risk of unauthorized users viewing the protected contents can be reduced by disabling caching for the protected contents of the browser..

The following header is output to disable caching the contents.

- Output example of response header

```
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Pragma: no-cache
Expires: 0
```

Note: Overwriting Cache-Control header

Cache-Control header is overwritten when Controller class of Spring MVC defines form class of `@SessionAttributes` or uses Model of `@SessionAttributes` attribute in the request handler.

Note: Browser compatible with HTTP1.0

Pragma header and Expires header are also output to enable Spring Security support the browser compatible with HTTP1.0 as well.

X-Frame-Options

X-Frame-Options header indicates whether the contents within the frame (`<frame>` or `<iframe>` element) are authorized. Risk of confidential information being stolen by using the malicious practice called Clickjacking can be eliminated by disabling the display of contents within a frame.

The following header is output to deny the display within the frame.

- Output example of response header (Default output of Spring Security)

```
X-Frame-Options: DENY
```

Note that, options other than the output example can be specified in X-Frame-Options header.

X-Content-Type-Options

X-Content-Type-Options header indicates a method to determine the contents type. Some browsers ignore the value of Content-Type header and determine the contents by looking at the content details. Risk of attack using cross-site scripting can be reduced when you disable contents view while determining the contents type.

The following header is output in order to disable viewing the content details while determining the type of contents.

- Output example of response header

```
X-Content-Type-Options: nosniff
```

X-XSS-Protection

X-XSS-Protection header indicates the method to detect harmful script using XSS filter function of the browser. Risk of attack using cross-site scripting can be reduced by enabling the XSS filter function and detecting harmful script.

Following header is output to enable XSS filter function and detect harmful script.

- Output example of response header (Default output of Spring Security)

```
X-XSS-Protection: 1; mode=block
```

Further, options other than the output example can be specified in X-XSS-Protection header.

Strict-Transport-Security

Strict-Transport-Security header indicates that user should access the browser by replacing HTTP with HTTPS when user accesses the browser using HTTPS and then tries to access it using HTTP. Risk of user being directed to malicious sites using malicious technique called as Man-in-the-Middle attack can be reduced by disabling HTTP use after accessing the browser using HTTPS.

Following header is output to disable the use of HTTP after accessing browser using HTTPS.

- Output example of response header (Default output of Spring Security)

```
Strict-Transport-Security: max-age=31536000 ; includeSubDomains
```

Note: Strict-Transport-Security

Strict-Transport-Security header is output only when the application server is accessed using HTTPS in the default implementation of Spring Security. Note that, Strict-Transport-Security header value can be changed by specifying the option.

Content-Security-Policy

The Content-Security-Policy header is a header to instruct the content to be allowed to be read by the browser. Since the browser reads only the contents of the whitelist specified in the Content - Security - Policy header, it can reduce the risk of receiving attacks (such as crosssite scripting attacks) executed by loading malicious content.

If you do not send the Content-Security-Policy header, the browser applies the same standard origin policy.

In order to restrict the source of content to only the same origin, the following header is output.

- Output example of response header

```
Content-Security-Policy: default-src 'self'
```

Note: About sending reports when a policy violation occurs

If a report is to be send at policy violation, specify the reporting URI in the report-uri directive.

In order to block the content if there is a violation of the same origin policy and send the report to `/csp_report`, output the following header.

- Output example of response header

```
Content-Security-Policy: default-src 'self'; report-uri /csp_report;
```

In addition, if there is a policy violation, if a report is to be send without blocking content, use the Content-Security-Policy-Report-Only header. By collecting reports using the Content-Security-Policy-Report-Only header and gradually modifying the policy and content,can reduce the risk of not being able to work correctly if the policy is applied to sites that already provide service.

In order to send a report if there is a violation of the same origin policy to `/csp_report`without blocking the content , output the following header.

- Output example of response header

```
Content-Security-Policy-Report-Only: default-src 'self'; report-uri /csp_
↪report;
```

Public-Key-Pins

Public-Key-Pins header presents the public key of the certificate associated with the site, to the browser in order to ensure authenticity of the certificate of the site. Even when the user visits the site again and is directed to a malicious site by using an attack technique called “man-in-the-middle” attack, a mismatch between public key of authentic site certificate retained by browser and public key of certificate presented by malicious site is detected and the access to the site can be blocked.

Following header is output in order to block access to a site when a certificate which does not match the information retained by browser, is detected.

- Output example of response header

```
Public-Key-Pins: max-age=5184000 ; pin-sha256=
↪"d6qzRu9zOECb90Uez27xWltNsj0e1Md7GkYYkVoZWmM=" ; pin-sha256=
↪"E9CZ9INDbd+2eRQozYqqbQ2yXLVKB9+xcprMF+44U1g="
```

Note: Regarding sending a violation report

A report-uri directive is specified similar to Content-Security-Policy in order to send a violation report to browser when the access is blocked.

Further, a Public-Key-Pins-Report-Only header is used instead of Public-Key-Pins header to send a violation report to the browser without blocking the access.

Note: Regarding settings of Public-Key-Pins header

If an error occurs in settings of Public-Key-Pins header, it is likely that user will not be able to access the site for a long period of time. Hence, it is recommended to switch to Public-Key-Pins header after conducting a thorough testing by using Public-Key-Pins-Report-Only header.

9.6.2 How to use

Applying security header output function

A method to apply the above mentioned security header output function is decribed.

The security header output function is added from Spring 3.2 and is applied by default except for the following security header .

- Content-Security-Policy
- Public-Key-Pins

Therefore, a specific definition is not required to enable the security header output function. Further, when the security header output function is not to be applied, it must be disabled explicitly.

Define a bean as given below when the security header output function is to be disabled.

- Definition example for spring-security.xml

```
<sec:http>
    <!-- omitted -->
    <sec:headers disabled="true"/> <!-- Disable by setting true in "disabled"␣
↪attribute -->
    <!-- omitted -->
</sec:http>
```

Selecting security header

Define a bean as given below for selecting the security header to be output. Here, the example denotes output of all security headers provided by Spring Security, but only required headers should be specified in practice.

- Definition example for spring-security.xml

```
<sec:headers defaults-disabled="true"> <!-- (1) -->
    <sec:cache-control/> <!-- (2) -->
    <sec:frame-options/> <!-- (3) -->
    <sec:content-type-options/> <!-- (4) -->
    <sec:xss-protection/> <!-- (5) -->
    <sec:hsts/> <!-- (6) -->
    <sec:content-security-policy policy-directives="default-src 'self'" /> <!--␣
↪(7) -->
    <sec:hpkp report-uri="https://www.example.net/hpkp-report"> <!-- (8) -->
```

```
        <sec:pins>
            <sec:pin algorithm="sha256">
↪d6qzRu9zOECb90Uez27xWltNsj0e1Md7GkYYkVoZWmM=</sec:pin>
            <sec:pin algorithm="sha256">
↪E9CZ9INDbd+2eRQozYqqbQ2yXLVKB9+xcprMF+44U1g=</sec:pin>
        </sec:pins>
    </sec:hpkp>
</sec:headers>
```

Sr. No.	Description
(1)	First disable the registration of the component which outputs the header applied by default.
(2)	Register the component which outputs Cache-Control(Pragma, Expires) header.
(3)	Register the component which outputs Frame-Options header.
(4)	Register the component which outputs X-Content-Type-Options header.
(5)	Register the component which outputs X-XSS-Protection header.
(6)	Register the component which outputs Strict-Transport-Security header.
(7)	Register the component which outputs Content-Security-Policy header and Content-Security-Policy-Report-Only header.
(8)	Register the component which outputs Public-Key-Pins header and Public-Key-Pins-Report-Only header. • When the public key of the certificate presented by the site does not match, a violation report is sent to `https://www.example.net/hpkp-report` without blocking the access. • Public key information for the backup is set to prevent inconsistencies of public key when the certificate is updated for the reasons such as compromise in the certificate and expiry of the certificate etc

Note: Regarding output of Public-Key-Pins header

Default setting of Spring Security outputs Public-Key-Pins-Report-Only header rather than Public-Key-Pins header.

Further, in the default setting of Spring Security, Public-Key-Pins header is output only when the application server is accessed using HTTPS.

Further, a method is also provided which disables security headers which are not required.

- Definition example for spring-security.xml

```
<sec:headers>
    <sec:cache-control disabled="true"/> <!-- Disable by setting true in "disabled"
 attribute -->
</sec:headers>
```

In the above example, only Cache-Control header is not output.

For details of security header, refer Official reference.

Specifying options of security header

Contents which are output by Spring Security by default, can be changed in the following header.

- X-Frame-Options
- X-XSS-Protection
- Strict-Transport-Security
- Content-Security-Policy(Content-Security-Policy-Report-Only)
- Public-Key-Pins(Public-Key-Pins-Report-Only)

An option [*1] can be specified in the attribute of each element by changing the bean definition of Spring Security.

- Definition example for spring-security.xml

```
<sec:frame-options policy="SAMEORIGIN" />
```

Output of custom header

Spring Security can also output the headers which are not provided by default.

A case study wherein following header is output, is explained.

```
X-WebKit-CSP: default-src 'self'
```

Define a bean as follows when the header described above is to be output.

[*1] Refer http://docs.spring.io/spring-security/site/docs/4.1.4.RELEASE/reference/htmlsingle/#nsa-headers for the options which can be specified in each element.

- Definition example for spring-security.xml

```
<sec:headers>
    <sec:header name="X-WebKit-CSP" value="default-src 'self'"/>
</sec:headers>
```

Sr. No.	Description
(1)	Add `<sec:header>` as child element of `<sec:headers>` element and specify the header name in `name` attribute and header value in `value` attribute.

Displaying security header for each request pattern

Spring Security can control the output of security header for each request pattern by using `RequestMatcher` interface system.

For example, when the contents to be protected are stored under the path `/secure/` and Cache-Control header is to be output only when the contents to be protected are accessed, define a bean as follows.

- Definition example for spring-security.xml

```
<!-- (1) -->
<bean id="secureCacheControlHeadersWriter"
      class="org.springframework.security.web.header.writers.
↪DelegatingRequestMatcherHeaderWriter">
    <constructor-arg>
        <bean class="org.springframework.security.web.util.matcher.
↪AntPathRequestMatcher">
            <constructor-arg value="/secure/**"/>
        </bean>
    </constructor-arg>
    <constructor-arg>
        <bean class="org.springframework.security.web.header.writers.
↪CacheControlHeadersWriter"/>
    </constructor-arg>
</bean>

<sec:http>
    <!-- omitted -->
    <sec:headers>
        <sec:header ref="secureCacheControlHeadersWriter"/> <!-- (2) -->
    </sec:headers>
    <!-- omitted -->
</sec:http>
```

Sr. No.	Description
(1)	Specify implementation class of `RequestMatcher` and `HeadersWriter` interface and define a bean for `DelegatingRequestMatcherHeaderWriter` class.
(2)	Add `<sec:header>` as child element of `<sec:headers>` element and specify a bean for `HeaderWriter` defined in (1) in `ref` attribute.

9.7 XSS Countermeasures

9.7.1 Overview

It explains about Cross-site scripting (hereinafter abbreviated as XSS). Cross Site Scripting is injection of malicious scripts across trusted web sites by deliberately using security defects in the web application. For example, when data entered in Web Application (form input etc.) is output in HTML without appropriate escaping, the characters of tag existing in input value are interpreted as HTML as is. If a script with malicious value is run, attacks such as session hijack occur due to cookie tampering and fetching of cookie values.

Stored & Reflected XSS Attacks

XSS attacks are broadly classified into two categories.

Stored XSS Attacks

In Stored XSS Attacks, the malicious code is permanently stored on target servers (such as database). Upon requesting the stored information, the user retrieves the malicious script from the server and ends up running the same.

Reflected XSS Attacks

In Reflected attacks, the malicious code sent as a part of the request to the server is reflected back along with error messages, search results, or other different types of responses. When a user clicks the malicious link or submits a specially crafted form, the injected code returns a result reflecting an occurrence of attack on user's browser. The browser ends up executing the malicious code because the value came from a trusted server.

Both Stored XSS Attacks and Reflected XSS Attacks can be prevented by escaping output value.

9.7.2 How to use

When the input from user is output as is, the system gets exposed to XSS vulnerability. Therefore, as a countermeasure against XSS vulnerability, it is necessary to escape the characters which have specific meaning in the HTML markup language.

Escaping should be divided into 3 types if needed.

Escaping types:

- Output Escaping
- JavaScript Escaping
- Event handler Escaping

Output Escaping

Escaping HTML special characters is a fundamental countermeasure against XSS vulnerability. Example of HTML special characters that require escaping and example after escaping these characters are as follows:

Before escaping	After escaping
&	&
<	<
>	>
"	"
'	'

To prevent XSS, `f:h()` should be used in all display items that are to be output as strings. An example of application where input value is to be re-output on different screen is given below.

Example of vulnerability when output values are not escaped

This example below is given only for reference; it should never be implemented.

Implementation of output screen

```
<!-- omitted -->
<tr>
    <td>Job</td>
    <td>${customerForm.job}</td>  <!-- (1) -->
</tr>
<!-- omitted -->
```

Sr. No.	Description
(1)	Job, which is a customerForm field, is output without escaping.

Enter <script> tag in Job field on input screen.



Figure.9.10 **Picture - Input HTML Tag**

It is recognized as <script> tag and dialog box is displayed.

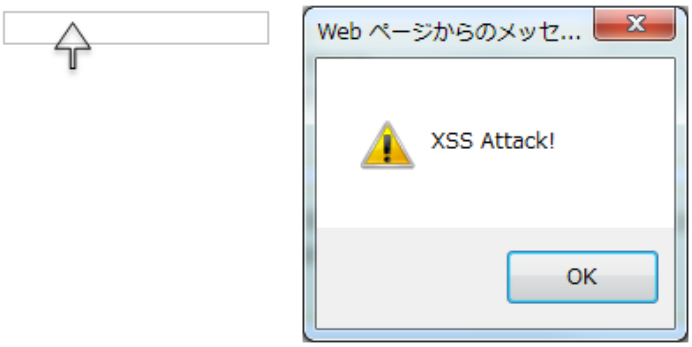


Figure.9.11 **Picture - No Escape Result**

Example of escaping output value using f:h() function

Implementation of output screen

```
<!-- omitted -->
<tr>
    <td>Job</td>
    <td>${f:h(customerForm.job)}</td>  <!-- (1) -->
</tr>
.<!-- omitted -->
```

Sr. No.	Description
(1)	EL function `f:h()` is used for escaping.

Enter <script> tag in Job field on input screen.

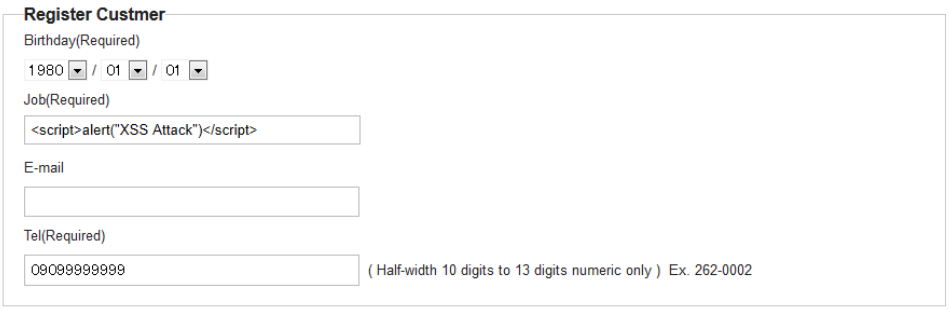


Figure.9.12 **Picture - Input HTML Tag**

By escaping special characters, input value is output as is without being recognized as <script> tag.

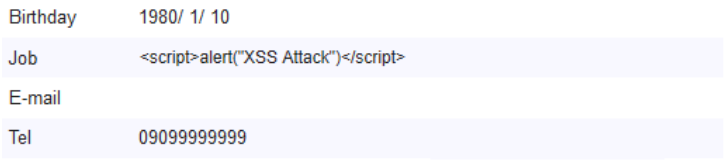


Figure.9.13 **Picture - Escape Result**

Output result

```
<!-- omitted -->
<tr>
    <td>Job</td>
    <td>&lt;script&gt;alert(&quot;XSS Attack&quot;)&lt;/script&gt;</td>
</tr>
<!-- omitted -->
```

Tip: java.util.Date subclass format

It is recommended that you use `<fmt:formatDate>` of JSTL to format and display java.util.Date subclasses. See the example below.

```
<fmt:formatDate value="${form.date}" pattern="yyyyMMdd" />
```

If `f:h()` is used for setting the value of "value" attribute, it gets converted into String and `javax.el.ELException` is thrown; hence `${form.date}` is used as is. However, it is safe from XSS attack since the value is in yyyyMMdd format.

Tip: String that can be parsed into java.lang.Number or subclass of java.lang.Number

It is recommended that you use `<fmt:formatNumber>` to format and display the string that can be parsed to java.lang.Number subclasses or java.lang.Number. See the example below.

```
<fmt:formatNumber value="${f:h(form.price)}" pattern="###,###" />
```

There is no problem even if the above is a String; hence when `<fmt:formatNumber>` tag is no longer used, `f:h()` is being used explicitly so that no one forgets to use `f:h()`.

JavaScript Escaping

Escaping JavaScript special characters is a fundamental countermeasure against XSS vulnerability. Escaping is must if it is required to dynamically generate JavaScript based on the outside input.

Example of JavaScript special characters that require escaping and example after escaping these characters are as follows:

Before escaping	After escaping
'	\'
"	\"
\	\\
/	\/
<	\x3c
>	\x3e
0x0D(Return)	\r
0x0A(Linefeed)	\n

Example of vulnerability when output values are not escaped

Example of occurrence of XSS problem is given below.

This example below is given only for reference; it should never be implemented.

```
<html>
  <script  type="text/javascript">
      var aaa = '<script>${warnCode}<\/script>';
      document.write(aaa);
  </script>
<html>
```

Attribute name	Value
warnCode	<script></script><script>alert('XSS Attack!');</script><\/script>

As shown in the above example, in order to dynamically generate JavaScript elements such as generating the code based on the user input, string literal gets terminated unintentionally leading to XSS vulnerability.

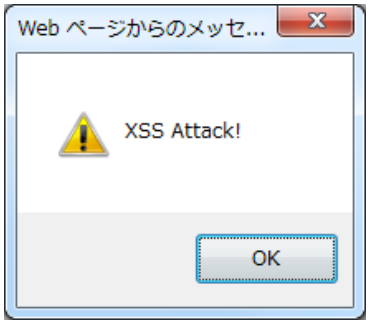


Figure.9.14 **Picture - No Escape Result**

Output result

```
<script type="text/javascript">
    var aaa = '<script><\/script><script>alert('XSS Attack!');<\/script><\/script>
↪';
    document.write(aaa);
</script>

.. tip::

Dynamically generated JavaScript code depending on user input carries a risk of␣
↪any script being inserted; hence an alternate way should be considered or it␣
↪should be avoided as much as possible unless there is a specific business␣
↪requirement.
```

Example of escaping output value using f:js() function

To prevent XSS, it is recommended that you use EL function `f:js()` for the value entered by user.

Usage example is shown below.

```
<script type="text/javascript">
  var message = '<script>${f:js(message)}<\/script>';  // (1)
  <!-- omitted -->
</script>
```

Sr. No.	Description
(1)	By using `f:js()` of EL function, the value is set as variable after escaping the value entered by user.

Output result

```
<script  type="text/javascript">
    var aaa = '<script>\x3c\/script\x3e\x3cscript\x3ealert(\'XSS Attack!\');\x3c\/
↪script\x3e<\/script>';
    document.write(aaa);
</script>
```

Event handler Escaping

To escape the value of event handler of javascript, `f:hjs()` should be used instead of `f:h()` or `f:js()`. It is equivalent to `${f:h(f:js())}`.

This is because, when `');alert("XSS Attack");// "` is specified as event handler value such as `<input type="submit" onclick="callback('xxxx');">`, different script gets inserted, after escaping the value in character reference format, escaping in HTML needs to be done.

Example of vulnerability when output values are not escaped

Example of occurrence of XSS problem is given below.

```
<input type="text" onmouseover="alert('output is ${warnCode}') . ">
```

Attribute name	Value
warnCode	`'); alert('XSS Attack!'); //` When the above values are set, string literal is terminated unintentionally leading to XSS attack.

XSS dialog box is displayed on mouse over.

Output result

```
<!-- omitted -->
<input type="text" onmouseover="alert('output is'); alert('XSS Attack!'); // .') ">
<!-- omitted -->
```

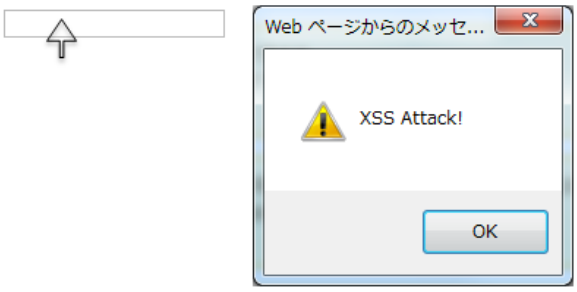


Figure.9.15 **Picture - No Escape Result**

Example of escaping output value using f:hjs() function

Example is shown below:

```
<input type="text" onmouseover="alert('output is ${f:hjs(warnCode)}') . ">  // (1)
```

Sr. No.	Description
(1)	Value after escaping by EL function `f:hjs()` is set as an argument of javascript event handler.

XSS dialog is not output on mouse over.



Figure.9.16 **Picture - Escape Result**

Output result

```
<!-- omitted -->
<input type="text" onmouseover="alert('output is \&#39;); alert(\&#39;XSS Attack!\&
↪#39;);\&quot; \/\/ .') ">
<!-- omitted -->
```

9.8 Encryption

9.8.1 Overview

Encryption is required for confidential information like personal data, passwords etc in the following cases.

- Sending/receiving confidential information through a network medium like Internet
- Storing confidential information in external resources like database, file etc

Spring Security also offers encryption related functions besides the major functions like “Authentication” and “Authorization”.
However, since functions offered by Spring Security are limited, the encryption methods which are not supported by Spring Security must be implemented independently.

The guideline explains following processes.

- Encryption and decryption by common key encryption methods which use class offered by Spring Security
- Generation of pseudo-random numbers which use class offered by Spring Security
- Encryption and decryption by public key encryption method which use JCA (Java Cryptography Architecture)
- Encryption and decryption by hybrid encryption method which use JCA

For details of Encryption function of Spring Security, refer Spring Security Reference -Spring Security Crypto Module-.

Encryption method

Encryption method is explained below.

Common key encryption method

It is a method which uses the same key during encryption and decryption.
Since the method shares a key used in decryption for the encryption, a different path to safely transfer the key to encryption side is required.

Public key encryption method

It is a method wherein encryption is performed by using a public key offered by decryption side and decryption is performed by using a secret key which is paired with the public key.

Secret key used while decrypting encrypted text is not published resulting in high security strength, however cost of encryption and decryption is likely to be high.

Hybrid encryption method

It is a method which combines merits of common key encryption method wherein processing cost is less and merits of public key encryption method wherein security strength is high due to easy management and distribution of key.

This method is used in SSL/TLS etc.

For example, in HTTPS communication, common key generated at the client side is sent after encrypting it with public key on the server side and common key is decrypted at the server by using secret key paired with public key. The subsequent communication is done by a common key encryption method which uses a shared common key.

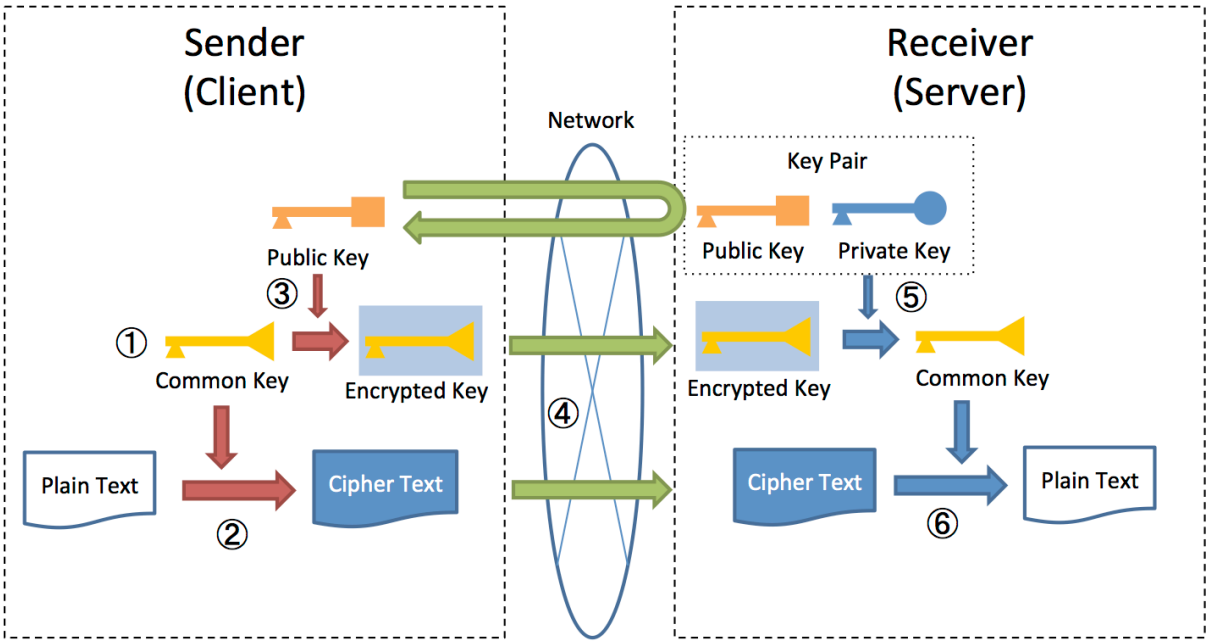
This method consists of

- Confidential information which is likely to be large in size is encrypted by a common key encryption method wherein the processing cost is less
- A common key with smaller size wherein safe distribution is required is encrypted by using a public key encryption method with high security strength

Since common key used while decrypting confidential information is protected by secret key, security strength of public key encryption method is retained while achieving a high-speed encryption and decryption process than public key encryption method.

Process flow of hybrid encryption method from encryption to decryption is shown in the following figure..

1. Sender generates a common key for encryption of plain text.
2. Plain text is encrypted by common key generated by sender.
3. Sender uses public key of receiving side to encrypt common key.
4. Sender sends encrypted text along with encrypted common key.
5. Receiving side decrypts encrypted common key by using secret key on receiving side.
6. Receiving side decrypts encrypted text by decrypted common key



Encryption algorithm

Encryption algorithm is explained.

DES / 3DES

DES (Data Encryption Standard) is an algorithm standardised in USA as an algorithm for common encryption method. Currently, it is not recommended since key length is short at 56 bits.

3DES (triple DES) is an encryption algorithm which repeats DES while changing the key.

AES

AES (Advanced Encryption Standard) is an algorithm for common key encryption method. It is an encryption standard established subsequent to DES and is used as a current default standard for encryption.

Also, ECB (Electronic Codebook), CBC (Cipher Block Chaining) and OFB (Output Feedback) are provided as encryption mode wherein a message longer than the block length is encrypted. Of these, CBC is widely used.

Note: AES with GCM

GCM (Galois/Counter Mode) is an encryption mode generally accepted for feasible parallel processing and excellent processing efficiency as compared to CEC and can be used in AES.

RSA

RSA is an algorithm of public key encryption method. Ability of a computing machine impacts the algorithm since it is based on difficulty of prime numbers factoring. It requires adequate key length as indicated in "Issues of encryption algorithm in 2010". At present, 2048 bits is used a standard length.

DSA / ECDSA

DSA (Digital Signature Algorithm) is a standard specification for digital signature. It is based on the difficulty of discrete logarithmic problem.

ECDSA (Elliptic Curve Digital Signature Algorithm) is a variant of DSA which uses Elliptic curve photography. Elliptic curve photography offers an advantage of reducing key length which is necessary for maintaining security level.

Pseudo-random number (Generator)

Random numbers are used in the generation of a key.

In this case, if the value of a number generated at random can be predicted, encryption security can no longer be maintained. Hence, a random number which cannot be predicted easily (pseudo-random number) must be used.

It is a pseudo-random number generator which is used to generate a pseudo-random number.

javax.crypto.Cipher class

`Cipher` class offers encryption and decryption functions, and specifies a combination of encryption algorithms like AES or RSA, encryption modes like ECB or CBC and padding methods like PKCS1.

Encryption mode is a mechanism to encrypt messages longer than block length, as explained in *AES*.

Also, padding method is a method of storage when encryption target that does not meet the required block length is to be encrypted.

Java application specifies a combination in the "`<Encryption algorithm>/<Encryption mode>/<Padding method>`" or "`<Encryption algorithm>`" format. For example, "`AES/CBC/PKCS5Padding`" or "`RSA`" are used. For details, refer to JavaDoc of Cipher class.

Encryption function in Spring Security

Spring Security offers an encryption and decryption function which uses a common key encryption method.

Encryption algorithm is 256-bit AES using PKCS #5's PBKDF2 (Password-Based Key Derivation Function #2).

Encryption mode is CBC and padding method is PKCS5Padding.

Components for encryption and decryption

Spring Security offers following interfaces as a function for encryption and decryption using common key encryption method.

- `org.springframework.security.crypto.encrypt.TextEncryptor` (For text)
- `org.springframework.security.crypto.encrypt.BytesEncryptor` (For byte array)

Also, following classes are offered as implementation classes for these interfaces and `Cipher` class is used internally.

- `org.springframework.security.crypto.encrypt.HexEncodingTextEncryptor` (For text)
- `org.springframework.security.crypto.encrypt.AesBytesEncryptor` (For byte array)

Components to generate random numbers

Spring Security offers following interfaces as functions to generate random numbers (key).

- `org.springframework.security.crypto.keygen.StringKeyGenerator` (for text)
- `org.springframework.security.crypto.keygen.BytesKeyGenerator` (for byte array)

Also, following classes are offered as implementation classes for these interfaces.

- `org.springframework.security.crypto.keygen.HexEncodingStringKeyGenerator` (for text)
- `org.springframework.security.crypto.keygen.SecureRandomBytesKeyGenerator` (for byte array. Generate a different key length by `generateKey` method and return)
- `org.springframework.security.crypto.keygen.SharedKeyGenerator` (for byte array. Return same key length set by constructor, using `generateKey` method)

Note: Spring Security RSA

spring-security-rsa offers API for public key encryption method and hybrid encryption method which use RSA as an encryption algorithm. spring-security-rsa is currently not managed as official repository of Spring <https://github.com/spring-projects>_. Later, how to use the repository will be explained in the guideline after moving it under official repository of Spring.

spring-security-rsa offers 2 classes given below.

- `org.springframework.security.crypto.encrypt.RsaRawEncryptor`

  A class that offers encryption and decryption functions which use public key encryption method.

- `org.springframework.security.crypto.encrypt.RsaSecretEncryptor`

  A class that offers encryption and decryption functions which use hybrid encryption method.

9.8.2 How to use

An unlimited strength JCE jurisdiction policy file must be applied for handling key length 256 bits of AES in some Java products like Oracle.

Note: JCE jurisdiction policy file

Default encryption algorithm strength is restricted in some Java products due to relation with import regulations. If a more powerful algorithm is to be used, an unlimited strength JCE jurisdiction policy file must be obtained and installed in JDK/JRE. For details, refer Java Cryptography Architecture Oracle Providers Documentation.

Download destination for JCE jurisdiction policy file

- For Oracle Java 8
- For Oracle Java 7

Common key encryption method

A method is explained wherein AES is used as an encryption algorithm.

Encryption of character string

- Encrypt text (string).

```
public static String encryptText(
    String secret, String salt, String plainText) {
    TextEncryptor encryptor = Encryptors.text(secret, salt); // (1)

    return encryptor.encrypt(plainText); // (2)
}
```

Sr. No.	Description
(1)	Call `Encryptors#text` method by specifying common key and salt, and generate instance of `TextEncryptor` class. Since initialization vector of generated instance is random, a varied result is returned at the time of encryption. It should be noted that CEC is used as an encryption mode. Common key and salt specified during encryption are also used at the time of decryption.
(2)	Encrypt plain text by using `encrypt` method.

Note: Regarding encryption results

Return value of `encrypt` method (encryption results) return a different value for each execution, however, if key and salt are identical, decryption process results will be similar as well (can be correctly decrypted).

- Fetch identical encryption results.

This method is used in the processes such as searching the database etc.using encrypted results. However, whether to use the method must be reviewed considering possible reduction in the security strength.

```
public static void encryptTextResult(
    String secret, String salt, String plainText) {
    TextEncryptor encryptor = Encryptors.queryableText(secret, salt); // (1)
    System.out.println(encryptor.encrypt(plainText)); // (2)
    System.out.println(encryptor.encrypt(plainText)); //
}
```

Sr.No.	Description
(1)	When identical value must be fetched as encryption results, generate an instance of `TextEncryptor` class by using `Encryptors#queryableText` method.
(2)	The instance generated by `Encryptors#queryableText` method returns identical values as the encryption results for `encrypt` method.

- Encrypt text (string) by using AES which uses GCM.

  AES using GCM can be used in Spring Security4.0.2 and subsequent versions. Processing efficiency is superior to CEC as explained in *AES*.

```
public static String encryptTextByAesWithGcm(String secret, String salt,␣
↪String plainText) {
    TextEncryptor aesTextEncryptor = Encryptors.delux(secret, salt); // (1)

    return aesTextEncryptor.encrypt(plainText); // (2)
}
```

Sr. No.	Description
(1)	Call `Encryptors#delux` method by specifying common key and salt, and generate an instance of `TextEncryptor` class. Common key and salt specified during encryption are also used at the time of decryption.
(2)	Encrypt plain text by using `encrypt` method.

Note: Java support for AES which uses GCM

AES using GCM can be used in Java SE8 and subsequent versions. For details, refer JDK 8 security enhancement.

Decryption of string

- Decrypt encryption text of text (string).

```
public static String decryptText(String secret, String salt, String␣
↪cipherText) {
    TextEncryptor decryptor = Encryptors.text(secret, salt); // (1)

    return decryptor.decrypt(cipherText); // (2)
}
```

Sr. No.	Description
(1)	Call `Encryptors#text` method by specifying common key and salt, and generate an instance of `TextEncryptor` class. Specify values used at the time of encryption as common key and salt.
(2)	Decrypt encrypted text by using `decrypt` method.

- Decrypt encrypted text of text (string) by using AES which uses GCM.

```
public static String decryptTextByAesWithGcm(String secret, String salt,␣
↪String cipherText) {
    TextEncryptor aesTextEncryptor = Encryptors.delux(secret, salt); // (1)

    return aesTextEncryptor.decrypt(cipherText); // (2)
}
```

Sr. No.	Description
(1)	Call `Encryptors#delux` method by specifying common key and salt, and generate an instance of `TextEncryptor` class. Specify values at the time of encryption as common key and salt.
(2)	Decrypt encrypted text by using `decrypt` method.

Encryption of byte array

- Encrypt byte array.

```
public static byte[] encryptBytes(String secret, String salt, byte[]␣
↪plainBytes) {
    BytesEncryptor encryptor = Encryptors.standard(secret, salt); // (1)

    return encryptor.encrypt(plainBytes); // (2)
}
```

Sr. No.	Description
(1)	Call `Encryptors#standard` method by specifying common key and salt, and generate an instance of `BytesEncryptor` class. Common key and salt specified during encryption are also used at the time of decryption.
(2)	Encrypt plain text of byte array by using `encrypt` method.

- Encrypt byte array by using AES which uses GCM.

```
public static byte[] encryptBytesByAesWithGcm(String secret, String salt,␣
↪byte[] plainBytes) {
```

```
    BytesEncryptor aesBytesEncryptor = Encryptors.stronger(secret, salt); //␣
↪(1)

    return aesBytesEncryptor.encrypt(plainBytes); // (2)
}
```

Sr. No.	Description
(1)	Call `Encryptors#stronger` method by specifying common key and salt, and generate an instance of `BytesEncryptor` class. Common key and salt specified during encryption are also used at the time of decryption.
(2)	Encrypt plain text of byte array by using `encrypt` method.

Decryption of byte array

Decrypt encrypted text of byte array.

```
public static byte[] decryptBytes(String secret, String salt, byte[]␣
↪cipherBytes) {
    BytesEncryptor decryptor = Encryptors.standard(secret, salt); // (1)

    return decryptor.decrypt(cipherBytes); // (2)
}
```

Sr. No.	Description
(1)	Call `Encryptors#standard` method by specifying common key and salt, and generate an instance of `BytesEncryptor` class. Specify values used at the time of encryption as common key and salt.
(2)	Decrypt encrypted text of byte array by using `decrypt` method.

- Decrypt byte array using AES which use GCM.

```
public static byte[] decryptBytesByAesWithGcm(String secret, String salt,␣
↪byte[] cipherBytes) {
    BytesEncryptor aesBytesEncryptor = Encryptors.stronger(secret, salt); //␣
↪(1)

    return aesBytesEncryptor.decrypt(cipherBytes); // (2)
}
```

Sr. No.	Description
(1)	Call `Encryptors#stronger` method by specifying common key and salt, and generate an instance of `BytesEncryptor` class. Specify values used at the time of encryption as common key and salt.
(2)	Decrypt encrypted text of byte array by using `decrypt` method.

Public key encryption method

Since functions related to public key encryption method are not offered by Spring Security, a method which uses JCA and OpenSSL is explained using a sample code.

Preliminary preparation (Generation of key pairs using JCA)

- Generate key pairs (a combination of public key / secret key) using JCA, perform encryption and decryption process by using public key and secret key respectively.

```
public void generateKeysByJCA() {
    try {
        KeyPairGenerator generator = KeyPairGenerator.getInstance("RSA"); //␣
↪(1)
        generator.initialize(2048); // (2)
        KeyPair keyPair = generator.generateKeyPair(); // (3)
        PublicKey publicKey = keyPair.getPublic();
        PrivateKey privateKey = keyPair.getPrivate();

        byte[] cipherBytes = encryptByPublicKey("Hello World!", publicKey);  /
↪/ (4)
```

```
        String plainText = decryptByPrivateKey(cipherBytes, privateKey); //␣
↪(5)
        System.out.println(plainText);
    } catch (NoSuchAlgorithmException e) {
        throw new SystemException("e.xx.xx.9002", "No Such setting error.",␣
↪e);
    }
}
```

Sr. No.	Description
(1)	Specify RSA algorithm and generate an instance of `KeyPairGenerator` class.
(2)	Specify 2048 bits as a key length.
(3)	Generate key pairs.
(4)	Use public key and perform encryption process. Processing details will be described later.
(5)	Use secret key and perform decryption process. Processing details will be described later.

Note: When the encrypted data is to be handled as string

When encrypted data is to be handled as string like in external system linkage etc, Base64 encoding can be listed as one of the measures. `java.util.Base64` of Java standard is used in case of subsequent versions of Java SE8. In the earlier versions, `org.springframework.security.crypto.codec.Base64` of Spring Security is used.

A method used for Base64 encoding and decoding is explained using `java.util.Base64` of Java standard.

- Base64 encoding

```
// omitted
byte[] cipherBytes = encryptByPublicKey("Hello World!", publicKey);  //␣
↪Encryption process
String cipherString = Base64.getEncoder().encodeToString(cipherBytes);  //␣
↪Convert encrypted text of byte array to string
```

```
// omitted
```

- Base64 decoding

```
// omitted
byte[] cipherBytes = Base64.getDecoder().decode(cipherString); // Convert␣
↪encrypted text of string to byte array
String plainText = decryptByPrivateKey(cipherBytes, privateKey); //␣
↪Decryption process
// omitted
```

Encryption

- Use public key and encrypt character string.

```
public byte[] encryptByPublicKey(String plainText, PublicKey publicKey) {
    try {
        Cipher cipher = Cipher.getInstance("RSA/ECB/PKCS1Padding"); // (1)
        cipher.init(Cipher.ENCRYPT_MODE, publicKey);                        //␣
↪(2)
        return cipher.doFinal(plainText.getBytes(StandardCharsets.UTF_8)); //
    } catch (NoSuchAlgorithmException | NoSuchPaddingException e) {
        throw new SystemException("e.xx.xx.9002", "No Such setting error.",␣
↪e);
    } catch (InvalidKeyException |
             IllegalBlockSizeException |
             BadPaddingException e) {
        throw new SystemException("e.xx.xx.9003", "Invalid setting error.",␣
↪e);
    }
}
```

Sr.No.	Description
(1)	Specify encryption algorithm, encryption mode and padding method, and generate an instance of `Cipher` class.
(2)	Execute encryption process.

Decryption

- Use secret key and decrypt byte array.

```
public String decryptByPrivateKey(byte[] cipherBytes, PrivateKey privateKey) {
    try {
        Cipher cipher = Cipher.getInstance("RSA/ECB/PKCS1Padding"); // (1)
        cipher.init(Cipher.DECRYPT_MODE, privateKey);           // (2)
        byte[] plainBytes = cipher.doFinal(cipherBytes); //
        return new String(plainBytes, StandardCharsets.UTF_8);
    } catch (NoSuchAlgorithmException | NoSuchPaddingException e) {
        throw new SystemException("e.xx.xx.9002", "No Such setting error.",␣
↪e);
    } catch (InvalidKeyException |
             IllegalBlockSizeException |
             BadPaddingException e) {
        throw new SystemException("e.xx.xx.9003", "Invalid setting error.",␣
↪e);
    }
}
```

Sr. No.	Description
(1)	Specify encryption algorithm, encryption mode and padding method, and generate an instance of `Cipher` class.
(2)	Execute decryption process.

OpenSSL

If Cipher is identical, a different method can be used for encryption and decryption for public key encryption method.

Here, key pairs are created in advance by using OpenSSL and encryption is performed by JCA, by using this public key. Hence, a method wherein decryption process is performed by OpenSSL, by using the secret key is explained.

Note: OpenSSL

The software must be installed for creating key pairs by OpenSSL. It can be downloaded from the following site.

Download destination for OpenSSL

- For Linux
- For Windows

- Create key pairs by OpenSSL as a preliminary preparation.

```
$ openssl genrsa -out private.pem 2048  # (1)

$ openssl pkcs8 -topk8 -nocrypt -in private.pem -out private.pk8 -outform DER
 # (2)

$ openssl rsa -pubout -in private.pem -out public.der -outform DER  # (3)
```

Sr. No.	Description
(1)	Generate secret key of 2048 bits (DER format) by OpenSSL..
(2)	Convert secret key to PKCS#8 format for reading it from Java application.
(3)	Generate public key (DER format) from secret key.

- Read public key created by OpenSSL in the application and perform encryption process by using public key that has been read.

```
public void useOpenSSLDecryption() {
    try {
        KeySpec publicKeySpec = new X509EncodedKeySpec(
```

```
                Files.readAllBytes(Paths.get("public.der"))); // (1)
        KeyFactory keyFactory = KeyFactory.getInstance("RSA");
        PublicKey publicKey = keyFactory.generatePublic(publicKeySpec); // (2)

        byte[] cipherBytes = encryptByPublicKey("Hello World!", publicKey); //
 (3)

        Files.write(Paths.get("encryptedByJCA.txt"), cipherBytes);
        System.out.println("Please execute the following command:");
        System.out
                .println("openssl rsautl -decrypt -inkey hoge.pem -in␣
↪encryptedByJCA.txt");
    } catch (IOException e) {
        throw new SystemException("e.xx.xx.9001", "input/output error.", e);
    } catch (NoSuchAlgorithmException e) {
        throw new SystemException("e.xx.xx.9002", "No Such setting error.",␣
↪e);
    } catch (InvalidKeySpecException e) {
        throw new SystemException("e.xx.xx.9003", "Invalid setting error.",␣
↪e);
    }
}
```

Sr.No.	Description
(1)	Read binary data from public key file.
(2)	Generate an instance of `PublicKey` class from binary data.
(3)	Perform encryption process by using public key.

- Check that details encrypted by JCA can be decrypted by OpenSSL.

```
$ openssl rsautl -decrypt -inkey private.pem -in encryptedByJCA.txt  # (1)
```

Sr. No.	Description
(1)	Decrypt by OpenSSL by using a secret key.

Further, a method wherein encryption and decryption are performed by OpenSSL and JCA respectively using key pairs created by OpenSSL is explained.

- Perform encryption process by using OpenSSL commands.

```
$ echo Hello | openssl rsautl -encrypt -keyform DER -pubin -inkey public.der -
↪out encryptedByOpenSSL.txt  # (1)
```

Sr. No.	Description
(1)	Encrypt by OpenSSL by using a public key.

- Read secret key created by OpenSSL in the application and perform decryption process by using a secret key that has been read.

```
public void useOpenSSLEncryption() {
    try {
        KeySpec privateKeySpec = new PKCS8EncodedKeySpec(
                Files.readAllBytes(Paths.get("private.pk8"))); // (1)
        KeyFactory keyFactory = KeyFactory.getInstance("RSA");
        PrivateKey privateKey = keyFactory.generatePrivate(privateKeySpec); //
 (2)

        String plainText = decryptByPrivateKey(
               Files.readAllBytes(Paths.get("encryptedByOpenSSL.txt")),
               privateKey); // (3)
        System.out.println(plainText);
    } catch (IOException e) {
        throw new SystemException("e.xx.xx.9001", "input/output error.", e);
    } catch (NoSuchAlgorithmException e) {
        throw new SystemException("e.xx.xx.9002", "No Such setting error.",␣
↪e);
```

```
    } catch (InvalidKeySpecException e) {
        throw new SystemException("e.xx.xx.9003", "Invalid setting error.",␣
↪e);
    }
}
```

Sr. No.	Description
(1)	Read binary data from secret key file of PKCS #8 format and generate an instance of `PKCS8EncodedKeySpec` class.
(2)	Generate an instance of `PrivateKey` class from `KeyFactory` class.
(3)	Perform decryption process by using a secret key.

Hybrid encryption method

Similar to public key encryption method, since functions related to hybrid encryption methods are not offered by Spring Security, it is explained using a sample code.

The sample code refers to RsaSecretEncryptor class of spring-security-rsa.

Encryption

```
public byte[] encrypt(byte[] plainBytes, PublicKey publicKey, String salt) {
    byte[] random = KeyGenerators.secureRandom(32).generateKey(); // (1)
    BytesEncryptor aes = Encryptors.standard(
            new String(Hex.encode(random)), salt); // (2)

    try (ByteArrayOutputStream result = new ByteArrayOutputStream()) {
        final Cipher cipher = Cipher.getInstance("RSA"); // (3)
        cipher.init(Cipher.ENCRYPT_MODE, publicKey); // (4)
        byte[] secret = cipher.doFinal(random); // (5)

        byte[] data = new byte[2]; // (6)
        data[0] = (byte) ((secret.length >> 8) & 0xFF); //
        data[1] = (byte) (secret.length & 0xFF); //
```

```
        result.write(data); //

        result.write(secret); // (7)
        result.write(aes.encrypt(plainBytes)); // (8)

        return result.toByteArray(); // (9)
    } catch (IOException e) {
        throw new SystemException("e.xx.xx.9001", "input/output error.", e);
    } catch (NoSuchAlgorithmException | NoSuchPaddingException e) {
        throw new SystemException("e.xx.xx.9002", "No Such setting error.",␣
↪e);
    } catch (InvalidKeyException | IllegalBlockSizeException |␣
↪BadPaddingException e) {
        throw new SystemException("e.xx.xx.9003", "Invalid setting error.",␣
↪e);
    }
}
```

Sr. No.	Description
(1)	Call `KeyGenerators#secureRandom` method by specifyinh 32 bytes as a key length and generate an instance of `BytesKeyGenerator` class. Call `BytesKeyGenerator#generateKey` method and generate a common key. For details, refer to *Random number generation*.
(2)	Specify generated common key and salt, and generate an instance of `BytesEncryptor` class.
(3)	specify RSA as an encryption algorithm and generate an instance of `Cipher` class.
(4)	Specify encryption mode constant and public key, and initialise an instance of `Cipher` class.
(5)	Execute encryption process of common key. The encryption is performed by using public key encryption process.
(6)	Store length of encrypted common key in encrypted text of byte array. Length of stored common key is used at the time of decryption.
(7)	Store encrypted common key in encrypted text of byte array.
(8)	Encrypt plain text and store in encrypted text of byte array. The encryption is performed by using common key encryption process.
(9)	Return encrypted text of byte array.

Decryption

```
public byte[] decrypt(byte[] cipherBytes, PrivateKey privateKey, String salt)
↪{

    try (ByteArrayInputStream input = new ByteArrayInputStream(cipherBytes);
            ByteArrayOutputStream output = new ByteArrayOutputStream()) {
        byte[] b = new byte[2]; // (1)
        input.read(b); //
        int length = ((b[0] & 0xFF) << 8) | (b[1] & 0xFF); //

        byte[] random = new byte[length]; // (2)
        input.read(random); //
        final Cipher cipher = Cipher.getInstance("RSA"); // (3)
        cipher.init(Cipher.DECRYPT_MODE, privateKey); // (4)
        String secret = new String(Hex.encode(cipher.doFinal(random))); // (5)
        byte[] buffer = new byte[cipherBytes.length - random.length - 2]; //␣
↪(6)
        input.read(buffer); //
        BytesEncryptor aes = Encryptors.standard(secret, salt); // (7)
        output.write(aes.decrypt(buffer)); // (8)

        return output.toByteArray(); // (9)
    } catch (IOException e) {
        throw new SystemException("e.xx.xx.9001", "input/output error.", e);
    } catch (NoSuchAlgorithmException | NoSuchPaddingException e) {
        throw new SystemException("e.xx.xx.9002", "No Such setting error.",␣
↪e);
    } catch (InvalidKeyException | IllegalBlockSizeException |␣
↪BadPaddingException e) {
        throw new SystemException("e.xx.xx.9003", "Invalid setting error.",␣
↪e);
    }
}
```

Sr. No.	Description
(1)	Fetch length of encrypted common key.
(2)	Fetch encrypted common key.
(3)	Specify RSA as an encryption algorithm and generate an instance of `Cipher` class.
(4)	Specify decryption mode constant and secret key, and initialise an instance of `Cipher` class.
(5)	Execute decryption process of common key. Decryption is performed by using public key encryption process.
(6)	Fetch decryption target.
(7)	Specify decrypted common key and salt, and generate an instance of `BytesEncryptor` class.
(8)	Execute decryption process. Decryption is performed by using common key encryption process.
(9)	Return plain text of decrypted byte array.

Random number generation

Generation of string type pseudo-random number

```
public static void createStringKey() {
    StringKeyGenerator generator = KeyGenerators.string(); // (1)
    System.out.println(generator.generateKey()); // (2)
    System.out.println(generator.generateKey()); //
}
```

Sr. No.	Description
(1)	Generate an instance of key (pseudo-random number) generator `StringKeyGenerator` class. If key is generated by this generator, a different value is obtained for each instance. Key length cannot be specified, a key of length 8 byte is always generated.
(2)	Generate a key (pseudo-random number) by using `generateKey` method.

Generation of byte array type pseudo-randon number

- Generate a different key.

```
public static void createDifferentBytesKey() {
    BytesKeyGenerator generator = KeyGenerators.secureRandom(); // (1)
    System.out.println(Arrays.toString(generator.generateKey())); // (2)
    System.out.println(Arrays.toString(generator.generateKey())); //
}
```

Sr. No.	Description
(1)	Call `KeyGenerators#secureRandom` method and generate an instance of key (pseudo-random number) generator `BytesKeyGenerator` class. If key is generated by this generator, a different value is obtained for each instance. When key length is not specified, a key of length - 8 bytes is generated by default.
(2)	Generate a key by using `generateKey` method.

- Generate identical key.

```
public static void createSameBytesKey() {
    BytesKeyGenerator generator = KeyGenerators.shared(32); // (1)
    System.out.println(Arrays.toString(generator.generateKey())); // (2)
    System.out.println(Arrays.toString(generator.generateKey())); //
}
```

Sr. No.	Description
(1)	Specify 32 bytes as key length, call `KeyGenerators#shared` method and generate an instance of key (pseudo-random number) generator `BytesKeyGenerator` class. If key is generated by this generator, same value is obtained for each instance. Specifying key length is mandatory.
(2)	Generate key by using `generateKey` method.

9.9 Implementation Example of Typical Security Requirements

9.9.1 Introduction

Topics described in this chapter

- Example of implementation method to meet the typical security requirements using TERASOLUNA Server Framework for Java (5.x)
- Implementation method and source code description using the sample application shown in *Description of application*

Warning:

- The implementation methods described in this chapter is just an example, and implementation must be carried out in the actual development as per the requirement.
- Since it does not guarantee an exhaustive implementation of security measures, additional measures should be considered if necessary

Target Readers

- Must understand the contents of *Application Development*
- Must understand the contents of *Spring Security Overview*, *Authentication*, *Authorization*
- Must complete implementation of *Spring Security Tutorial*

9.9.2 Description of application

This section explains about a specific implementation method for security measures by using a sample application that meets the typical security requirements.

The list of security requirements that describe the implementation in this section is shown below. The functions of the sample application used as a base and the specifications for authentication and authorization are also shown below.

Henceforth, this sample application will be referred to as 'the application'.

Security requirements

The list of security requirements fulfilled by the application is shown below. The implementation example is described in *Implementation method and code description* for each classification.

Sr. No.	Classification	Requirement	Overview
(1)	*Force/Prompt password change*	Force password change when using initial password	Forces password change when authentication is successful using initial password
(2)		Force to change expired password	Force password change when authentication is successful for the users who have not changed the password for a certain period In this application, it is intended only for Administrator
(3)		Display message prompting password change	Displays message prompting password change when authentication is successful for the users who have not changed the password for a certain period
(4)	*Check password strength*	Specify minimum password length	Specifies the minimum length that can be set for the password
(5)		Specify the type of characters for the password	Specifies the type of characters (uppercase letters, lowercase letters, numbers, symbols) that must be included in the password
(6)		Prohibit user name from being used as password	Prohibit user name of the account from being used in the password

次のページに続く

Table. 9.38 – 前のページからの続き

Sr. No.	Classification	Requirement	Overview
(7)		Prohibit reuse of administrator password	Prohibit reusing the password which has been recently used by the administrator
(8)	*Account lockout*	Account lockout	If authentication of a certain account has failed for more than a specific number of times within a short period, then that account is set to 'authentication disabled' state (lockout state)
(9)		Specify account lockout duration	Specifies the duration for account lockout state
(10)		Unlock by administrator	Administrator can unlock any account
(11)	*Display date and time of last login*	Display date and time of last login	After successful authentication of an account, displays the date and time of last successful authentication of that account on the top screen
(12)	*Create authentication information for password reissue*	Assign random string to the password reissue URL	In order to prevent unauthorized access, a string that is difficult to guess is assigned to URL which is used to access the password reissue screen

次のページに続く

Table. 9.38 – 前のページからの続き

Sr. No.	Classification	Requirement	Overview
(13)		Issue confidential information for password reissue	Create confidential information in advance (Random string) that is difficult to guess, in order to use for user verification at the time of reissuing password
(14)	*Distribution of authentication information for password reissue*	Send a mail for password reissue screen URL	Send the URL to access the password reissue screen to the registered e-mail address of the account
(15)		Separate distribution of the password reissue screen URL and confidential information	Distribute confidential information to the user using a mode other than e-mail as a precaution against leakage of password reissue screen URL
(16)	*Verification at the time of executing password reissue*	Set validity period for authentication information for password reissue	Set validity period for password reissue screen URL and confidential information, and disable password reissue screen URL and confidential information if the validity period has expired
(17)	*Set the maximum limit for password reissue failure*	Set the maximum limit for password reissue failure	Disable password reissue screen URL and confidential information when the authentication fails for a specific number of times at the time of password reissue

次のページに続く

Table. 9.38 – 前のページからの続き

Sr. No.	Classification	Requirement	Overview
(18)	*Input value check from security viewpoint*	Set commonly prohibited characters for request parameters	Set commonly prohibited characters in overall application, for the string included in the request parameters
(19)		Set commonly prohibited string for uploaded file name	Set commonly prohibited string in overall application, for the uploaded file name
(20)		Input check for control string	Check whether control character is included in the input value.
(21)		Input check for file extension	Check whether extension of the uploaded file is allowed in the application.
(22)		Input check for file name	Check whether the file name for the uploaded file matches with the pattern authorized in the application
(23)		Input check for URL domain	Check whether domain of input URL is authorized by the application.
(24)		Input check for mail address domain	Check whether domain of input mail address is authorized by the application
(25)	*Audit log output*	Audit log output	Output date and time, user name, operation details and operation results for each request in a log

Functions

The application consists of following functions in addition to the application created in *Spring Security Tutorial*.

Function name	Description
Create new account function	A function to create new account
Password change function	Function to enable logged-in users to change their account password
Account lockout function	Function to set an account that has failed to authenticate more than a specific number of times in a short period to the 'authentication disabled' state
Unlock function	Function to return the account which is in the 'authentication disabled' state due to the account lockout function, to the 'authentication enabled' state again
Password reissue function	Function that can set a new password if the user has forgotten the password, after the confirmation with the user

Note: Since this application is a sample of security measures, it is essentially required. Update function for registration information other than user registration function and password is not created.

Specifications for authentication/authorization

In this application, the specifications for authentication/authorization are shown below respectively.

Authentication

- Initial password to be used for authentication will be issued by the application

Authorization

- Authentication is required to access the screens other than login screen, screen used for creating an account and the screen used for password reissue
- There are two types of roles, "General user" and "Administrator"
  - A single account can have multiple roles
- Account unlock function can be used only by the account having administrator rights

Authentication at the time of reissuing password

- The following information created by the application is used for the password reissue authentication
  - URL for Password reissue screen
  - Confidential information for authentication

•URL of the password reissue screen generated by the application is in the following format:

–{baseUrl}/reissue/resetpassword?form&token={token}

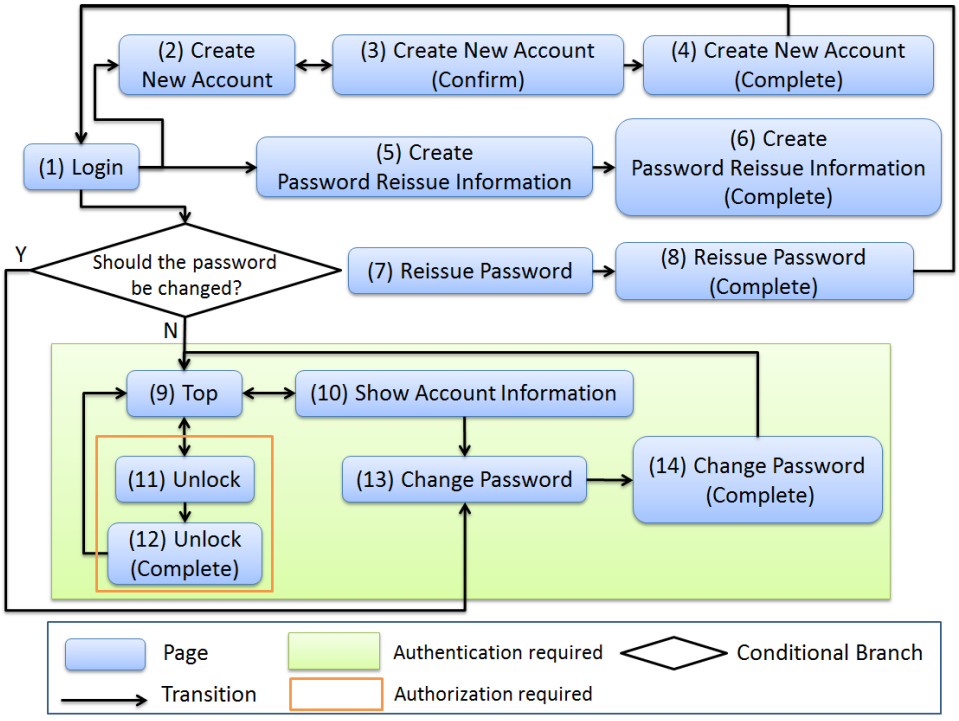
* {baseUrl} : Base URL of application
* {token} : UUID version4 format string（36 characters including hyphen, 128bit）

- A time-limit of 30 minutes is provided for the password reissue screen URL and authentication is possible only within the validity period

Design information

Page transition

Screen transition diagram is shown below. Screen transition in case of an error is omitted.



Sr. No.	Screen Name	Access control
(1)	Login screen	-

次のページに続く

Table. 9.39 – 前のページからの続き

Sr. No.	Screen Name	Access control
(2)	Create new account screen	-
(3)	A screen to confirm input for creating new account	-
(4)	New account creation complete screen	-
(5)	A screen to generate authentication information for password reissue	-
(6)	A screen to complete generation of authentication information for password reissue	-
(7)	Password reissue screen	-
(8)	Password reissue complete screen	-
(9)	Top screen	Authenticated users only
(10)	Account information display screen	Authenticated users only
(11)	Unlock screen	Administrator only
(12)	Unlock complete screen	Administrator only

次のページに続く

Table. 9.39 – 前のページからの続き

Sr. No.	Screen Name	Access control
(13)	Password change screen	Authenticated users only
(14)	Password change complete screen	Authenticated users only

URL List

URL list is shown below.

Sr. No.	Process name	HTTP method	URL	Description
1	Display login screen	GET	/login	Display login screen
2	Login	POST	/login	Authenticate using user name and password entered in the screen (performed by Spring Security)
3	Logout	POST	/logout	Log out (performed by Spring Security)
4	Display top screen	GET	/	Display top screen
5	Display account information	GET	/accounts	Display account information of login user
6	A screen to create new account	GET	/accounts/create?form	Display a screen to create new account
7	A screen to confirm input for creating new account	POST	/accounts/create?confirm	Display a screen to confirm input for creating new account
8	Create new account	POST	/accounts/create	Create a new account with input details
9	A screen to complete creation of new account	GET	/accounts/create?complete	Display a screen to complete new account creation
10	Display password change screen	GET	/password?form	Display password change screen
11	Password change	POST	/password	Change account password by using information input in the password change screen
12	Display password change complete screen	GET	/password?complete	Display password change complete screen
13	Display unlock screen	GET	/unlock?form	Display unlock screen
14	Unlock	POST	/unlock	Unlock an account by using information input in unlock screen
15	Display unlock complete screen	GET	/unlock?complete	Display unlock complete screen
16	Display a screen to generate authentication information for password reissue	GET	/reissue/create?form	Display a screen to generate authentication information for password reissue

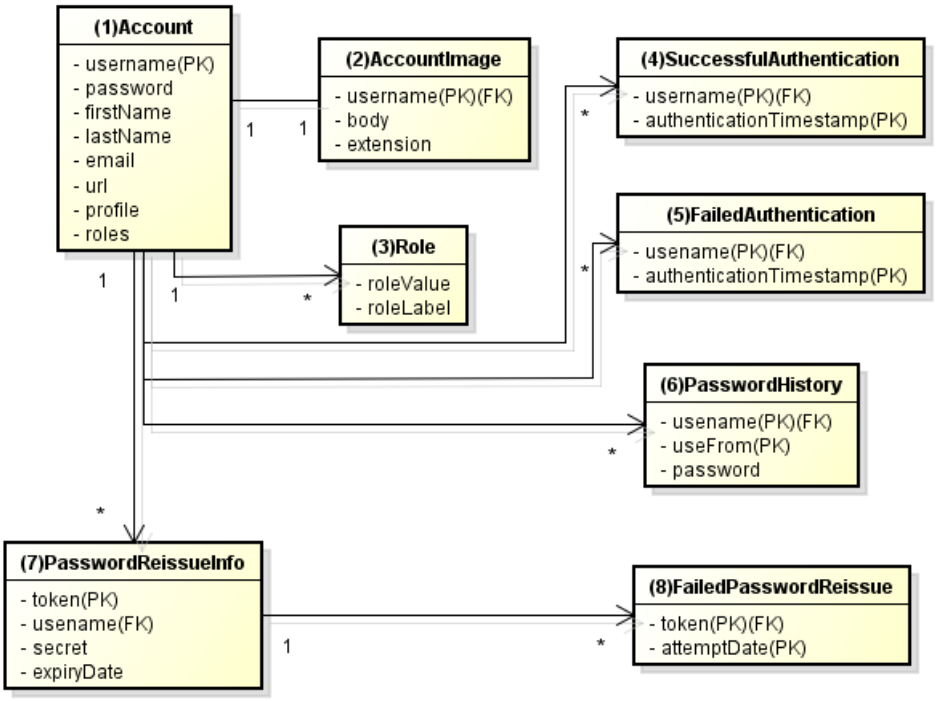
次のページに続く

Table. 9.40 – 前のページからの続き

Sr. No.	Process name	HTTP method	URL	Description
17	Generate authentication information for password reissue	POST	/reissue/create	Generate authentication information for password reissue
18	A screen to complete generation of authentication information for password reissue	GET	/reissue/create?complete	Display a screen to complete generation of authentication information for password reissue
19	Display password reissue screen	GET	/reissue/resetpassword?form&token={token}	Display password reissue screen by using two request parameters
20	Password reissue	POST	/reissue/resetpassword	Reissue password by using the information input in password reissue screen
21	Display password reissue complete screen	GET	/reissue/resetpassword?complete	Display password reissue complete screen

ER diagram

ER diagram in this application is shown below.



Sr. No.	Entity name	Description	Attribute
(1)	Account	Registered account information of user	username : User name password : Password（Hashed） firstName : First name lastName : Last name email : E-mail address url : Individual Web site and blog URL profile : Profile roles : Role(s)

次のページに続く

Table. 9.41 – 前のページからの続き

Sr. No.	Entity name	Description	Attribute
(2)	Account image	Image registered by the user for the account	username : User name for the account corresponding to the account image body : Binary image file extension : Extension of image file
(3)	Role	Rights to be used in authorization	roleValue : Identifier of role roleLabel : Display name of role
(4)	Authentication successful event	Information saved when authentication is successful in order to get the last login date and time of account	username : User name authenticationTimestamp : Date and time when authentication is successful
(5)	Authentication failed event	Information saved when authentication failed to be used by account lockout function	username : User name authenticationTimestamp : Date and time when authentication failed
(6)	Password change history	Information saved at the time of password change to be used to determine password expiration date	username : User name useFrom : Date and time when changed password is activated password : Changed password

次のページに続く

Table. 9.41 – 前のページからの続き

Sr. No.	Entity name	Description	Attribute
(7)	Authentication information for password reissue	Information to be used for user verification at the time of password reissue	token : String used to make a unique and difficult to guess password reissue screen URL username : User name secret : String to be used for user verification experyDate : Expiry date of authentication information for password reissue
(8)	Password reissue failed event	Information saved in password reissue failure to restrict the number of attempts for password reissue	token : token used when failed to reissue password attemptDate : Date and time when password reissue was attempted

Tip: In order to determine initial password and password expiration, a design can also be adopted wherein the information such as last modified date and time of password is provided by adding a field to the account entity. When implementation is done using this method, it is likely to lead to a situation where a column is added for determining various conditions in account table and entries are frequently updated.

In this application, table is maintained in a simple form. In order to fulfil the requirements by simply using Insert and Delete without unnecessary updates of the entries, a design using event entity such as authentication successful event entity has been adopted.

9.9.3 Implementation method and code description

Method of implementation in this application and the code are described for each classification of security requirements.

Only the minimum code required to fulfil the requirements for each classification is described here. Refer to GitHub for the complete code.

SQL for initial data registration to run this application is placed here.

Note: In this application, Lombok is used to eliminate boilerplate code. For Lombok, refer *Reducing Boilerplate Code (Lombok)*.

Force/Prompt password change

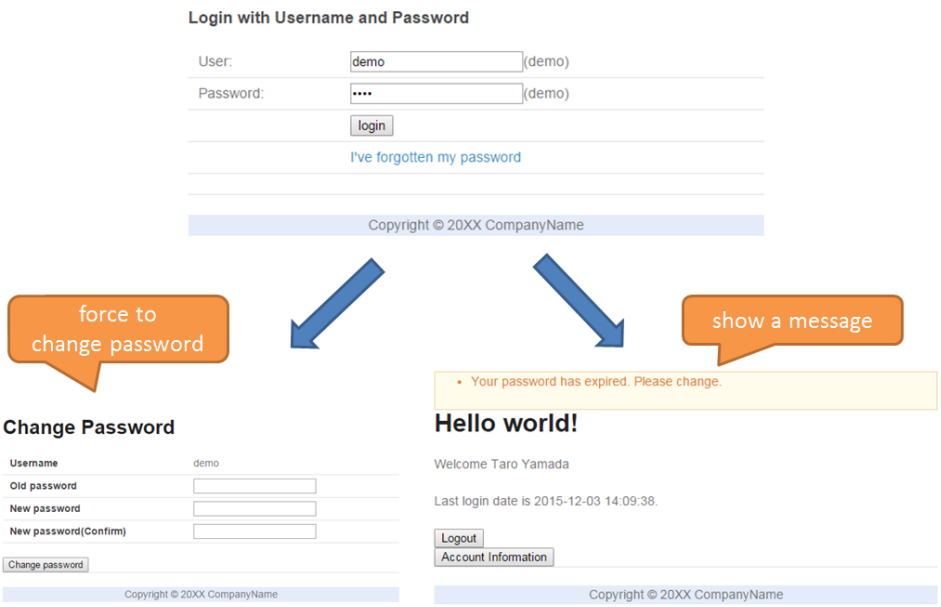
List of requirements to be implemented

- *Force password change when initial password is used*
- *Force to change expired administrator password*
- *Display message prompting password change*

Working image

Implementation method

In this application, the history when password is changed is stored as "Password change history" entity in the database. Using this password change history entity, the initial password and password expiration are determined.



Note that, redirecting to the password change screen and displaying message on the screen are controlled based on the determination result.

In particular, requirements are fulfilled by implementing and using the following process.

- Saving password change history entity

  When the password is changed, register password change history entity containing following information to the database.

  - User name of the account for which password is changed
  - Date and time when changed password is activated

- Determining initial password and password expiration

  After authentication, search the password change history entity of the authenticated account from the database. If even a single record is not found, consider that initial password is being used.

  Otherwise, get the latest password change history entity, calculate the difference between current date and time, and date and time when the password is activated and determine whether the password has expired.

- Forcible redirect to password change screen

  To force password change, the user is redirected to the password change screen in case a request is raised for a screen other than password change screen, when the conditions below are met.

  - When initial password is used by an authenticated user

  - When authenticated user is administrator and password has expired

Using `org.springframework.web.servlet.handler.HandlerInterceptor` , determine whether the above conditions are met before executing handler method of Controller.

Tip: There are other methods to redirect to the password change screen after authentication, however, depending on the method, it is likely that user gets access to a screen different from that of a password change screen by clicking the URL directly after redirecting. In the method that uses `HandlerInterceptor` , it cannot be avoided by a method wherein URL is directly clicked since the process is executed before executing handler method.

Tip: Servlet Filter can also be used instead of `HandlerInterceptor` . For both the descriptions, refer to *Implementing common logic to be executed before and after calling controller*. Here, `HandlerInterceptor` is used to perform processing for only the requests allowed by the application.

- Display message prompting password change

  Call the password expiration determination process described previously in the Controller. Pass the determination result to View, and switch show/hide message in View.

Code description

The code implemented according to the implementation method mentioned above is described sequentially.

- Saving password change history entity

  A series of implementations to register password change history entity in the database at the time of changing the password is shown below.

  - Implementation of Entity

    Implementation of password change history entity is as below.

```
package org.terasoluna.securelogin.domain.model;

// omitted

@Data
public class PasswordHistory {

    private String username; // (1)

    private String password; // (2)
```

```
    private DateTime useFrom; // (3)

}
```

Sr. No.	Description
(1)	User name of the account for which the password is changed
(2)	Password after change
(3)	Date and time of when changed password is activated

- Implementation of Repository

The Repository to register and search password change history entity to the database is shown below.

```
package org.terasoluna.securelogin.domain.repository.passwordhistory;

// omitted

public interface PasswordHistoryRepository {

    int create(PasswordHistory history); // (1)

    List<PasswordHistory> findByUseFrom(@Param("username") String username,
            @Param("useFrom") LocalDateTime useFrom); // (2)

    List<PasswordHistory> findLatest(
            @Param("username") String username, @Param("limit") int limit);
 ↪ // (3)

}
```

Sr. No.	Description
(1)	A method to register `PasswordHistory` object that has been assigned as an argument, as a record in the database
(2)	A method to get `PasswordHistory` object newer than the date specifying the date and time when the password is activated, in descending order (new order) by considering user name that has been assigned as an argument, as the key
(3)	A method to get specified number of `PasswordHistory` objects in new order by considering user name that has been assigned as an argument, as the key

Mapping file is as described below.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<mapper
    namespace="org.terasoluna.securelogin.domain.repository.
↪passwordhistory.PasswordHistoryRepository">

    <resultMap id="PasswordHistoryResultMap" type="PasswordHistory">
        <id property="username" column="username" />
        <id property="password" column="password" />
        <id property="useFrom" column="use_from" />
    </resultMap>

    <select id="findByUseFrom" resultMap="PasswordHistoryResultMap">
    <![CDATA[
        SELECT
            username,
            password,
            use_from
        FROM
            password_history
        WHERE
            username = #{username} AND
            use_from >= #{useFrom}
        ORDER BY use_from DESC
    ]]>
    </select>

    <select id="findLatest" resultMap="PasswordHistoryResultMap">
```

```
    <![CDATA[
        SELECT
            username,
            password,
            use_from
        FROM
            password_history
        WHERE
            username = #{username}
        ORDER BY use_from DESC
        LIMIT #{limit}
    ]]>
    </select>

    <insert id="create" parameterType="PasswordHistory">
    <![CDATA[
        INSERT INTO password_history (
            username,
            password,
            use_from
        ) VALUES (
            #{username},
            #{password},
            #{useFrom}
        )
    ]]>
    </insert>
</mapper>
```

– Service implementation

Password change history entity operations are also used in *Check password strength*. Therefore, call the Repository method from SharedService as shown below.

```
package org.terasoluna.securelogin.domain.service.passwordhistory;

// omitted

@Service
@Transactional
public class PasswordHistorySharedServiceImpl implements
        PasswordHistorySharedService {

    @Inject
    PasswordHistoryRepository passwordHistoryRepository;

    @Transactional(propagation = Propagation.REQUIRES_NEW)
    public int insert(PasswordHistory history) {
        return passwordHistoryRepository.create(history);
    }
```

```
    @Transactional(readOnly = true)
    public List<PasswordHistory> findHistoriesByUseFrom(String username,
            LocalDateTime useFrom) {
        return passwordHistoryRepository.findByUseFrom(username, useFrom);
    }

    @Override
    @Transactional(readOnly = true)
    public List<PasswordHistory> findLatest(String username, int limit) {
        return passwordHistoryRepository.findLatest(username, limit);
    }

}
```

Implementation of the process to save password change history entity in the database at the time of changing the password is shown below.

```
package org.terasoluna.securelogin.domain.service.account;

// omitted

@Service
@Transactional
public class AccountSharedServiceImpl implements AccountSharedService {

    @Inject
    ClassicDateFactory dateFactory;

    @Inject
    PasswordHistorySharedService passwordHistorySharedService;

    @Inject
    AccountRepository accountRepository;

    @Inject
    PasswordEncoder passwordEncoder;

    // omitted

    public boolean updatePassword(String username, String rawPassword) { //
 (1)
        String password = passwordEncoder.encode(rawPassword);
        boolean result = accountRepository.updatePassword(username,␣
↪password); // (2)

        LocalDateTime passwordChangeDate = dateFactory.newTimestamp().
↪toLocalDateTime();

        PasswordHistory passwordHistory = new PasswordHistory(); // (3)
```

```
        passwordHistory.setUsername(username);
        passwordHistory.setPassword(password);
        passwordHistory.setUseFrom(passwordChangeDate);
        passwordHistorySharedService.insert(passwordHistory); // (4)

        return result;
    }

    // omitted
}
```

Sr. No.	Description
(1)	A Method which is called while changing the password
(2)	Call the process to update the password in the database.
(3)	Create password change history entity and set user name, changed password, and date and time when changed password is activated.
(4)	Call the process to register the created password change history entity in the database.

- Determining initial password and password expiration

  Using the password change history entity registered in the database, implementation of the process to determine whether initial password is used and whether the password has expired is shown below.

```
package org.terasoluna.securelogin.domain.service.account;

// omitted

@Service
@Transactional
public class AccountSharedServiceImpl implements AccountSharedService {

    @Inject
    ClassicDateFactory dateFactory;

    @Inject
    PasswordHistorySharedService passwordHistorySharedService;
```

```
    @Value("${security.passwordLifeTimeSeconds}") // (1)
    int passwordLifeTimeSeconds;

    // omitted

   @Transactional(readOnly = true)
   @Override
   @Cacheable("isInitialPassword")
   public boolean isInitialPassword(String username) { // (2)
       List<PasswordHistory> passwordHistories = passwordHistorySharedService
               .findLatest(username, 1); // (3)
       return passwordHistories.isEmpty(); // (4)
   }

   @Transactional(readOnly = true)
   @Override
   @Cacheable("isCurrentPasswordExpired")
   public boolean isCurrentPasswordExpired(String username) { // (5)
       List<PasswordHistory> passwordHistories = passwordHistorySharedService
               .findLatest(username, 1); // (6)

       if (passwordHistories.isEmpty()) { // (7)
           return true;
       }

       if (passwordHistories
               .get(0)
               .getUseFrom()
               .isBefore(
                       dateFactory.newTimestamp().toLocalDateTime()
                               .minusSeconds(passwordLifeTimeSeconds))) { //␣
↪(8)
           return true;
       }

       return false;
   }

}
```

Sr. No.	Description
(1)	Fetch the length of the time period (in seconds) for which password is valid, from the property file and set.
(2)	A method which determines whether initial password is used, and returns true if it is used, or else returns false.
(3)	Call the process to fetch single record of the latest password change history entity from the database.
(4)	If password change history entity cannot be fetched from the database, determine that initial password is being used and return true. Otherwise, return false.
(5)	A method which determines whether the password currently being used has expired and returns true if it has expired, or else returns false.
(6)	Call the process to fetch single record of the latest password change history entity from the database.
(7)	If password change history entity cannot be fetched from the database, determine that the password has expired and return true.
(8)	If difference between the current date and time, and the date and time when the password fetched from the password change history entity is activated, is greater than the password validity period set in (1), determine that the password has expired and return true.
(9)	If any of the conditions of (7), (8) is not met, determine that the password is within the validity period and return false.

Tip: `@ Cacheable` assigned to isInitialPassword and isCurrentPasswordExpired is an annotation to use the Spring Cache Abstraction function. The result for method arguments can be cached by assigning `@Cacheable` annotation. Access to database during each initial password and password expiration determination is prevented by the use of the cache thereby preventing performance degradation. Refer to Official document for Cache Abstraction.

Further, while using cache, it should be noted that it is necessary to clear the cache as and when needed. In this application, at the time of changing the password or during logout, clear the cache to determine password expiration and determine initial password again.

Further, set cache TTL (Time to Live) as needed. Note that TTL is not set depending on the implementation of the cache to be used.

- Forcible redirect to password change screen

  In order to enforce password change, the implementation of the process to be redirected to the password change screen is shown below.

```
package org.terasoluna.securelogin.app.common.interceptor;

// omitted

public class PasswordExpirationCheckInterceptor extends
        HandlerInterceptorAdapter { // (1)

    @Inject
    AccountSharedService accountSharedService;

    @Override
    public boolean preHandle(HttpServletRequest request,
            HttpServletResponse response, Object handler) throws IOException
↪{ // (2)
        Authentication authentication = (Authentication) request
                .getUserPrincipal();

        if (authentication != null) {
            Object principal = authentication.getPrincipal();
            if (principal instanceof UserDetails) { // (3)
                LoggedInUser userDetails = (LoggedInUser) principal; // (4)
                if ((userDetails.getAccount().getRoles().contains(Role.ADMIN)␣
↪&& accountSharedService
                        .isCurrentPasswordExpired(userDetails.getUsername()))␣
↪// (5)
                        || accountSharedService.isInitialPassword(userDetails
                                .getUsername())) { // (6)
                    response.sendRedirect(request.getContextPath()
                            + "/password?form"); // (7)
```

```
                    return false; // (8)
                }
            }
        }

        return true;
    }
}
```

Sr. No.	Description
(1)	Inherit `org.springframework.web.servlet.handler.HandlerInterceptorAdapter` to include the process before the execution of handler method of Controller.
(2)	A method executed before the execution of handler method of Controller
(3)	Check whether the fetched user information is an object of `org.springframework.security.core.userdetails.UserDetails`.
(4)	Fetch `UserDetails` object . In this application, a class called `LoggedInUser` is created and used as the implementation of `UserDetails` .
(5)	Determine whether the user is an administrator by fetching the role from `UserDetails` object. Then, call the process to determine whether the password has expired. Perform logical AND (And) of these two results.
(6)	Call the process to determine whether initial password is being used.
(7)	If either (5) or (6) is true, redirect to the password change screen using `sendRedirect` method of `javax.servlet.http.HttpServletResponse` .
(8)	Return false to prevent handler method of Controller being executed continuously.

The settings to enable the redirect process described above are as described below.

spring-mvc.xml

```
<!-- omitted -->

<mvc:interceptors>
```

```
    <!-- omitted -->

    <mvc:interceptor>
        <mvc:mapping path="/**" /> <!-- (1) -->
        <mvc:exclude-mapping path="/password/**" /> <!-- (2) -->
        <mvc:exclude-mapping path="/reissue/**" /> <!-- (3) -->
        <mvc:exclude-mapping path="/resources/**" />
        <mvc:exclude-mapping path="/**/*.html" />
        <bean
            class="org.terasoluna.securelogin.app.common.interceptor.
↪PasswordExpirationCheckInterceptor" /> <!-- (4) -->
    </mvc:interceptor>

    <!-- omitted -->

</mvc:interceptors>

<!-- omitted -->
```

Sr. No.	Description
(1)	Use `HandlerInterceptor` to access all the paths under "/".
(2)	Exclude the paths under "/password" to prevent redirecting from password change screen to password change screen.
(3)	Exclude the paths under "/reissue" since it is not necessary to check password expiration at the time of reissuing password.
(4)	Specify the class of `HandlerInterceptor` .

- Display message prompting password change

  Implementation of Controller to display message prompting password change on top screen is shown below.

```
package org.terasoluna.securelogin.app.welcome;

// omitted

@Controller
```

```
public class HomeController {

    @Inject
    AccountSharedService accountSharedService;

    @RequestMapping(value = "/", method = { RequestMethod.GET,
            RequestMethod.POST })
    public String home(@AuthenticationPrincipal LoggedInUser userDetails, //␣
↪(1)
            Model model) {

        Account account = userDetails.getAccount(); // (2)

        model.addAttribute("account", account);

        if(accountSharedService.isCurrentPasswordExpired(account.
↪getUsername())){ // (3)
            ResultMessages messages = ResultMessages.warning().add(
                    "w.sl.pe.0001");
            model.addAttribute(messages);
        }

        // omitted

        return "welcome/home";

    }

}
```

Sr. No.	Description
(1)	Fetch object of `LoggedInUser` for which `UserDetails` is implemented by specifying `AuthenticationPrincipal` annotation.
(2)	Fetch account information retained by `LoggedInUser` .
(3)	Call the password expiration determination process by using the user name obtained from account information as an argument. If the result is true, fetch the message from the property file, set it in Model and pass it to View.

Implementation of View is as follows:

Top screen(home.jsp)

```
<!-- omitted -->

<body>
   <div id="wrapper">
       <span id="expiredMessage">
           <t:messagesPanel /> <!-- (1) -->
       </span>

       <!-- omitted -->

   </div>
</body>

<!-- omitted -->
```

Sr. No.	Description
(1)	Using messagesPanel tag, display the password expiration message passed from the Controller.

Check password strength

List of requirements to be implemented

- *Specify minimum number of characters for password*
- *Specify character type for password*
- *Prohibit password containing user name*
- *Prohibit reuse of administrator password*

Working image

Implementation method

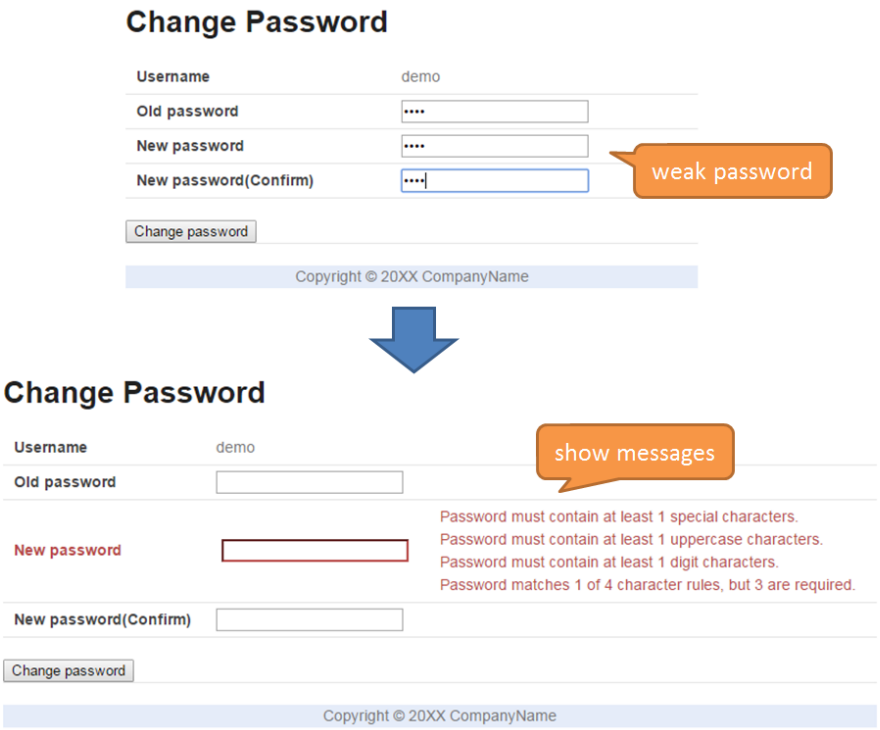
Input Validation function can be used to verify the strength of the password specified by the user at the time of password change. In this application, the strength of the password is verified by using Bean Validation.

Requirements for password strength are wide-ranging and differ depending on the application.

Use Passay as the library for password validation and create the required Bean Validation annotation.

Many functions that are commonly used in password validation have been provided in Passay. The functions that have not been provided can also be easily implemented by extending the standard functions.

Refer to *Appendix* for the overview of Passay.



In particular, describe the following settings and process and fulfil the requirements using them.

- Creating validation rules for Passay

  Create the following validation rules to be used to fulfil the requirements.

  - Validation rule wherein minimum password length is set
  - Validation rule wherein the character type that must be included in the password is set
  - Validation rule to check that the password does not contain user name
  - Validation rule to check that same password has not been used recently

- Creating Passay validator

  Create Passay validator wherein validation rules created above, are set.

- Creating Bean Validation annotation

  Create an annotation for password validation using Passay validator. All the validation rules can also be verified by one annotation, however, verifying various rules leads to complex process and reduced visibility. To avoid this, it should be implemented by dividing into two as shown below.

  - Annotation to validate the characteristics of the password

    Check the three validation rules, “Password is longer than the minimum string length”, “Password includes characters of the specified character type”, and “Password does not contain user name”

  - Annotation to compare with previous password

    Check that the recently used password is not reused by the administrator in recent period of time.

  Any annotation is a correlation input check rule using user name and new password. When a violation occurs in either of the inputs of both the rules, respective error message is displayed.

- Password validation

  Perform password validation using the created Bean Validation annotation.

Code description

The code implemented according to the implementation method mentioned above is described sequentially. Password validation using Passay is described in *Password validation*.

- Creating validation rules for Passay

  Most of the verification rules used in this application can be defined by using the class provided in Passay by default.
  However, in the class provided by Passay, validation rule to compare with the hashed previous password cannot be defined in `org.springframework.security.crypto.password.PasswordEncoder`.
  Therefore, it is necessary to create a class with individual validation rules by extending the class provided by Passay as shown below.

```
package org.terasoluna.securelogin.app.common.validation.rule;

// omitted

public class EncodedPasswordHistoryRule extends HistoryRule { // (1)

    PasswordEncoder passwordEncoder; // (2)

    public EncodedPasswordHistoryRule(PasswordEncoder passwordEncoder) {
        this.passwordEncoder = passwordEncoder;
    }

    @Override
    protected boolean matches(final String clearText,
            final PasswordData.Reference reference) { // (3)
        return passwordEncoder.matches(clearText, reference.getPassword()); //
 (4)
    }
}
```

Sr. No.	Description
(1)	Extend `org.passay.HistoryRule` to check that password is not a recently used password.
(2)	Inject `PasswordEncoder` used for password hashing.
(3)	Override the method to compare with previous password.
(4)	Compare with hashed password using `matches` method of `PasswordEncoder` .

Define a Bean for validation rules of Passay as shown below.

applicationContext.xml

```
<bean id="lengthRule" class="org.passay.LengthRule"> <!-- (1) -->
    <property name="minimumLength" value="${security.passwordMinimumLength}" /
↪>
</bean>
<bean id="upperCaseRule" class="org.passay.CharacterRule"> <!-- (2) -->
    <constructor-arg name="data">
        <util:constant static-field="org.passay.EnglishCharacterData.UpperCase
↪" />
    </constructor-arg>
    <constructor-arg name="num" value="1" />
</bean>
<bean id="lowerCaseRule" class="org.passay.CharacterRule"> <!-- (3) -->
    <constructor-arg name="data">
        <util:constant static-field="org.passay.EnglishCharacterData.LowerCase
↪" />
    </constructor-arg>
    <constructor-arg name="num" value="1" />
</bean>
<bean id="digitRule" class="org.passay.CharacterRule"> <!-- (4) -->
    <constructor-arg name="data">
        <util:constant static-field="org.passay.EnglishCharacterData.Digit" />
    </constructor-arg>
    <constructor-arg name="num" value="1" />
</bean>
<bean id="specialCharacterRule" class="org.passay.CharacterRule"> <!-- (5) -->
```

```
    <constructor-arg name="data">
        <util:constant static-field="org.passay.EnglishCharacterData.Special"␣
↪/>
    </constructor-arg>
    <constructor-arg name="num" value="1" />
</bean>
<bean id="characterCharacteristicsRule" class="org.passay.
↪CharacterCharacteristicsRule"> <!-- (6) -->
    <property name="rules">
        <list>
            <ref bean="upperCaseRule" />
            <ref bean="lowerCaseRule" />
            <ref bean="digitRule" />
            <ref bean="specialCharacterRule" />
        </list>
    </property>
    <property name="numberOfCharacteristics" value="3" />
</bean>
<bean id="usernameRule" class="org.passay.UsernameRule" /> <!-- (7) -->
<bean id="encodedPasswordHistoryRule"
    class="org.terasoluna.securelogin.app.common.validation.rule.
↪EncodedPasswordHistoryRule"> <!-- (8) -->
    <constructor-arg name="passwordEncoder" ref="passwordEncoder" />
</bean>
```

Sr. No.	Description
(1)	In `org.passay.LengthRule` property to check the password length, set the minimum length of the password fetched from the property file.
(2)	A validation rule to check that one or more single-byte upper-case letters are included. Set `org.passay.EnglishCharacterData.UpperCase` and numerical value 1 in `org.passay.CharacterRule` constructor to check the character type included in the password.
(3)	A validation rule to check that one or more single-byte lower-case letters are included. Set `org.passay.EnglishCharacterData.LowerCase` and numerical value 1 in `org.passay.CharacterRule` constructor to check the character type included in the password.
(4)	A validation rule to check that one or more single-byte digits are included. Set `org.passay.EnglishCharacterData.Digit` and numerical value 1 in `org.passay.CharacterRule` constructor to check for the character type included in the password.
(5)	A validation rule to check that one or more single-byte symbols are included. Set `org.passay.EnglishCharacterData.Special` and numerical value 1 in `org.passay.CharacterRule` constructor to check for the character type included in the password.
(6)	A validation rule to check that 3 out of the 4 validation rules from (2)-(5) are met. Set Bean list defined in (2)-(5) and numerical value 3 in `org.passay.CharacterCharacteristicsRule` property.
(7)	A validation rule to check that password does not contain user name
(8)	A validation rule to check that the password is not included in the passwords used in the past

- Creating Passay validator

  Using validation rules of Passay described above, Bean definition for validator to perform actual validation is shown below.

  applicationContext.xml

```
<bean id="characteristicPasswordValidator" class="org.passay.PasswordValidator
↪"> <!-- (1) -->
    <constructor-arg name="rules">
        <list>
            <ref bean="lengthRule" />
            <ref bean="characterCharacteristicsRule" />
            <ref bean="usernameRule" />
        </list>
    </constructor-arg>
</bean>
<bean id="encodedPasswordHistoryValidator" class="org.passay.PasswordValidator
↪"> <!-- (2) -->
    <constructor-arg name="rules">
        <list>
            <ref bean="encodedPasswordHistoryRule" />
        </list>
    </constructor-arg>
</bean>
```

Sr. No.	Description
(1)	A validator to validate the characteristics of the password. Set a Bean for `LengthRule` , `CharacterCharacteristicsRule` , `UsernameRule` as a property.
(2)	A validator to check the history of the passwords that were used in the past. Set a Bean for `EncodedPasswordHistoryRule` as a property.

- Creating Bean Validation annotation

  To fulfil the requirements, create two annotations that use the validator described above.

  - Annotation to validate the characteristics of the password

    The implementation of the annotation to check three validation rules - 'password should be longer than the minimum string length, it should contain characters of specified character type and it should not contain user name' is shown below.

```
package org.terasoluna.securelogin.app.common.validation;
```

```
// omitted

@Documented
@Constraint(validatedBy = { StrongPasswordValidator.class }) // (1)
@Target({ TYPE, ANNOTATION_TYPE })
@Retention(RUNTIME)
public @interface StrongPassword {
    String message() default "{org.terasoluna.securelogin.app.common.
↪validation.StrongPassword.message}";

    Class<?>[] groups() default {};

    String usernamePropertyName(); // (2)

    String newPasswordPropertyName(); // (3)

    @Target({ TYPE, ANNOTATION_TYPE })
    @Retention(RUNTIME)
    @Documented
    public @interface List {
        StrongPassword[] value();
    }

    Class<? extends Payload>[] payload() default {};
}
```

Sr. No.	Description
(1)	Specify `ConstraintValidator` to be used at the time of assigning annotation.
(2)	A property to specify property name of the user name.
(3)	A property to specify property name for the password.

```
package org.terasoluna.securelogin.app.common.validation;

// omitted

public class StrongPasswordValidator implements
        ConstraintValidator<StrongPassword, Object> {

    @Inject
    @Named("characteristicPasswordValidator") // (1)
    PasswordValidator characteristicPasswordValidator;
```

```
    private String usernamePropertyName;

    private String newPasswordPropertyName;

    @Override
    public void initialize(StrongPassword constraintAnnotation) {
        usernamePropertyName = constraintAnnotation.usernamePropertyName();
        newPasswordPropertyName = constraintAnnotation.
↪newPasswordPropertyName();
    }

    @Override
    public boolean isValid(Object value, ConstraintValidatorContext␣
↪context) {
        BeanWrapper beanWrapper = new BeanWrapperImpl(value);
        String username = (String) beanWrapper.
↪getPropertyValue(usernamePropertyName);
        String newPassword = (String) beanWrapper
                .getPropertyValue(newPasswordPropertyName);

        RuleResult result = characteristicPasswordValidator
                .validate(PasswordData.newInstance(newPassword, username,␣
↪null)); // (2)

        if (result.isValid()) { // (3)
            return true;
        } else {
            context.disableDefaultConstraintViolation();
            for (String message : characteristicPasswordValidator
                    .getMessages(result)) { // (4)
                context.buildConstraintViolationWithTemplate(message)
                        .addPropertyNode(newPasswordPropertyName)
                        .addConstraintViolation();
            }
            return false;
        }
    }
}
```

Sr. No.	Description
(1)	Inject validator of Passay.
(2)	Create an instance of `org.passay.PasswordData` wherein password and user name are specified and perform validation by the validator.
(3)	Confirm the check result, if it is OK, return true, else return false.
(4)	Fetch and set all the password validation error messages.

- Annotation to compare with password used in the past

Implementation of the annotation to check that the administrator does not reuse the password used earlier within a short period of time, is shown below.

Password change history entity is used to get the password used in the past. Refer to *Force/Prompt password change* for the password change history entity.

Note: In the setting of "Prevent reuse of password used before specified period", it is possible to reuse a password by repeating a password within a short period of time. In order to prevent this, check is performed in this application by setting "Prevent reuse of password used from a certain period onwards

---

```
package org.terasoluna.securelogin.app.common.validation;

@Documented
@Constraint(validatedBy = { NotReusedPasswordValidator.class }) // (1)
@Target({ TYPE, ANNOTATION_TYPE })
@Retention(RUNTIME)
public @interface NotReusedPassword {
    String message() default "{org.terasoluna.securelogin.app.common.
↪validation.NotReusedPassword.message}";

    Class<?>[] groups() default {};

    String usernamePropertyName(); // (2)
```

```
    String newPasswordPropertyName(); // (3)

    @Target({ TYPE, ANNOTATION_TYPE })
    @Retention(RUNTIME)
    @Documented
    public @interface List {
        NotReusedPassword[] value();
    }

    Class<? extends Payload>[] payload() default {};
}
```

Sr. No.	Description
(1)	Specify `ConstraintValidator` to be used while assigning an annotation.
(2)	A property to specify the property name of the user name. It is required to search the password used in the past, from the database.
(3)	A property to specify the property name of the password.

```
package org.terasoluna.securelogin.app.common.validation;

// omitted

public class NotReusedPasswordValidator implements
        ConstraintValidator<NotReusedPassword, Object> {

    @Inject
    ClassicDateFactory dateFactory;

    @Inject
    AccountSharedService accountSharedService;

    @Inject
    PasswordHistorySharedService passwordHistorySharedService;

    @Inject
    PasswordEncoder passwordEncoder;

    @Inject
    @Named("encodedPasswordHistoryValidator") // (1)
```

```
    PasswordValidator encodedPasswordHistoryValidator;

    @Value("${security.passwordHistoricalCheckingCount}") // (2)
    int passwordHistoricalCheckingCount;

    @Value("${security.passwordHistoricalCheckingPeriod}") // (3)
    int passwordHistoricalCheckingPeriod;

    private String usernamePropertyName;

    private String newPasswordPropertyName;

    private String message;

    @Override
    public void initialize(NotReusedPassword constraintAnnotation) {
        usernamePropertyName = constraintAnnotation.usernamePropertyName();
        newPasswordPropertyName = constraintAnnotation.
↪newPasswordPropertyName();
        message = constraintAnnotation.message();
    }

    @Override
    public boolean isValid(Object value, ConstraintValidatorContext␣
↪context) {
        BeanWrapper beanWrapper = new BeanWrapperImpl(value);
        String username = (String) beanWrapper.
↪getPropertyValue(usernamePropertyName);
        String newPassword = (String) beanWrapper
                .getPropertyValue(newPasswordPropertyName);

        Account account = accountSharedService.findOne(username);
        String currentPassword = account.getPassword();

        boolean result = checkNewPasswordDifferentFromCurrentPassword(
                newPassword, currentPassword, context); // (4)
        if (result && account.getRoles().contains(Role.ADMIN)) { // (5)
            result = checkHistoricalPassword(username, newPassword,␣
↪context);
        }

        return result;
    }

    private boolean checkNewPasswordDifferentFromCurrentPassword(
            String newPassword, String currentPassword,
            ConstraintValidatorContext context) {
        if (!passwordEncoder.matches(newPassword, currentPassword)) {
            return true;
        } else {
```

```
            context.disableDefaultConstraintViolation();
            context.buildConstraintViolationWithTemplate(message)
                    .addPropertyNode(newPasswordPropertyName).
↪addConstraintViolation();
            return false;
        }
    }

    private boolean checkHistoricalPassword(String username,
            String newPassword, ConstraintValidatorContext context) {
        LocalDateTime useFrom = dateFactory.newTimestamp().
↪toLocalDateTime()
                .minusMinutes(passwordHistoricalCheckingPeriod);
        List<PasswordHistory> historyByTime = passwordHistorySharedService
                .findHistoriesByUseFrom(username, useFrom);
        List<PasswordHistory> historyByCount = passwordHistorySharedService
                .findLatest(username, passwordHistoricalCheckingCount);
        List<PasswordHistory> history = historyByCount.size() >␣
↪historyByTime
                .size() ? historyByCount : historyByTime; // (6)

        List<PasswordData.Reference> historyData = new ArrayList<>();
        for (PasswordHistory h : history) {
            historyData.add(new PasswordData.HistoricalReference(h
                    .getPassword())); // (7)
        }

        PasswordData passwordData = PasswordData.newInstance(newPassword,
                username, historyData); // (8)
        RuleResult result = encodedPasswordHistoryValidator
                .validate(passwordData); // (9)

        if (result.isValid()) { // (10)
            return true;
        } else {
            context.disableDefaultConstraintViolation();
            context.buildConstraintViolationWithTemplate(
                    encodedPasswordHistoryValidator.getMessages(result).
↪get(0)) // (11)
                    .addPropertyNode(newPasswordPropertyName).
↪addConstraintViolation();
            return false;
        }
    }
}
```

Sr. No.	Description
(1)	Inject Passay validator.
(2)	Fetch the threshold to prohibit reuse of password up to a previous date, from the property file and inject it.
(3)	Fetch the threshold (in seconds) to prohibit the reuse of the password used from a date onwards, from the property file and inject it.
(4)	Call the process to check whether new password is different from the currently used password. Perform this check regardless of the general user / administrator.
(5)	In case of administrator, call the process to check that new password is not included in the previously used passwords.
(6)	Fetch the number of password change history entities specified in (2) and the password change history entities of the period specified in (3) and use the larger number of the two for the subsequent checks.
(7)	In order to make a comparison with the previous password using Passay validator, fetch the password from the password change history entity and create a list of `org.passay.PasswordData.HistoricalReference`.
(8)	Create an instance of `org.passay.PasswordData` which specifies password, user name and list of previous passwords.
(9)	Perform validation by using the validator.
(10)	Confirm the check result, if it is OK, return true, else return false.

次のページに続く

Table. 9.42 – 前のページからの続き

Sr. No.	Description
(11)	Fetch the password validation error messages.

- Password validation

  Perform password validation in the application layer which use Bean Validation annotation. Since input check other than Null check is covered by the annotation assigned to Form class, only `@NotNull` is assigned as a single item check.

```
package org.terasoluna.securelogin.app.passwordchange;

// omitted

import lombok.Data;

@Data
@Compare(source = "newPasssword", destination = "confirmNewPassword",␣
↪operator = Compare.Operator.EQUAL) // (1)
@StrongPassword(usernamePropertyName = "username", newPasswordPropertyName =
↪"newPassword") // (2)
@NotReusedPassword(usernamePropertyName = "username", newPasswordPropertyName␣
↪= "newPassword") // (3)
@ConfirmOldPassword(usernamePropertyName = "username",␣
↪oldPasswordPropertyName = "oldPassword") // (4)
public class PasswordChangeForm implements Serializable{

    private static final long serialVersionUID = 1L;

    @NotNull
    private String username;

    @NotNull
    private String oldPassword;

    @NotNull
    private String newPassword;

    @NotNull
    private String confirmNewPassword;

}
```

Sr. No.	Description
(1)	An annotation to check whether second input of new password is identical with the first input. Refer to *terasoluna-gfw-common check rules* for the details.
(2)	An annotation to verify the characteristic of the password, described above
(3)	An annotation to compare with the previous password
(4)	An annotation to check that the entered current password is correct. Definition will be omitted.

```
package org.terasoluna.securelogin.app.passwordchange;

// omitted

@Controller
@RequestMapping("password")
public class PasswordChangeController {

    @Inject
    PasswordChangeService passwordService;

    // omitted

    @RequestMapping(method = RequestMethod.POST)
    public String change(@AuthenticationPrincipal LoggedInUser userDetails,
            @Validated PasswordChangeForm form, BindingResult bindingResult, /
↪/ (1)
            Model model) {

        Account account = userDetails.getAccount();
        if (bindingResult.hasErrors() ||
                !account.getUsername().equals(form.getUsername())) { // (2)
            model.addAttribute(account);
            return "passwordchange/changeForm";
        }

        passwordService.updatePassword(form.getUsername(),
                form.getNewPassword());

        return "redirect:/password?complete";
    }
```

```
    // omitted

}
```

Sr. No.	Description
(1)	A handler method called at the time of changing the password. Perform validation by assigning `@Validated` annotation to Form in the parameter.
(2)	Confirm that the user name for password change and the user name of the logged-in account are identical. If the two users are different, the user is again taken to the password change screen.

Note: In this application, user name is fetched from the Form to perform password validation using the user name in Bean Validation. It is assumed that in View, the user name set in `Model` is retained as hidden, however, since there is a risk of tampering, user name obtained from the Form before password change is confirmed.

Account lock

List of requirements to be implemented

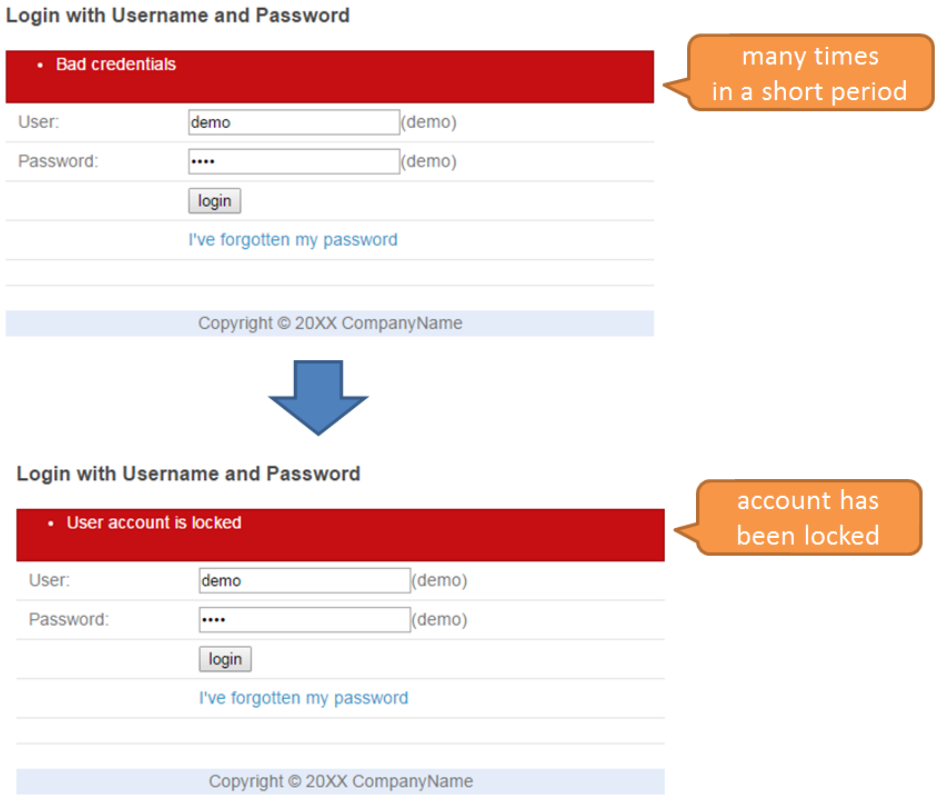
- *Account lock*
- *Specifying account lockout duration*
- *Unlocking by the administrator*

Working image

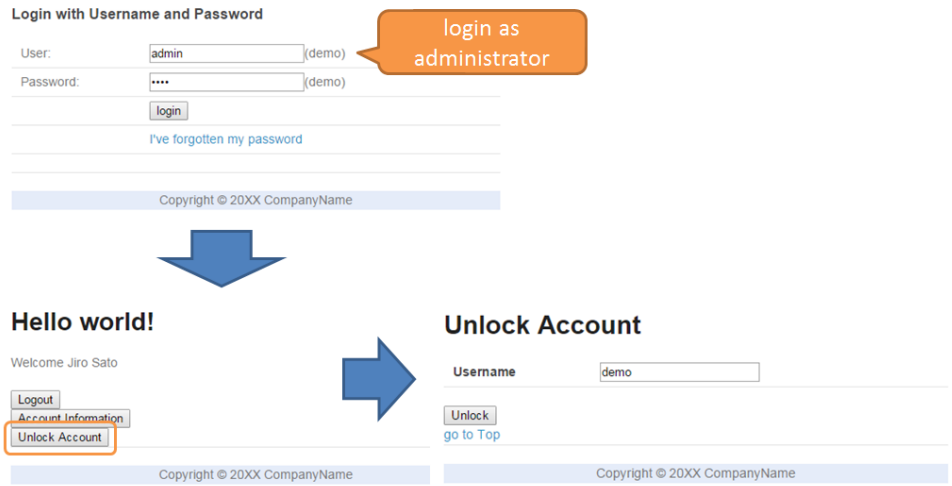
- Account lock

In the login form, if you try to authenticate a user name with an incorrect password for a certain number of times, successively in a short duration, then that user's account will be locked. Locked account is not authenticated even if a set of correct user name and password is entered.

Locked status is cancelled after a certain period of time or by unlocking it.



- Unlock



Unlock function can be used only when the user having administrator rights has logged in. If unlocking is carried out by entering the user name for which the locked status is to be resolved, then the account of that user returns to the status wherein authentication can be done again.

Implementation method

In Spring Security, an account lockout status can be set for `org.springframework.security.core.userdetails.UserDetails`.

If "Locked" is set, Spring Security reads that setting and throws `org.springframework.security.authentication.LockedException`.

By using this function, if only the process set in `UserDetails` is implemented by determining whether the account is locked, lockout function can be implemented.

In this application, the history of authentication failure is stored in the database as an "authentication failure event" entity, and the lockout status of the account is determined using this authentication failure event entity.

In particular, each requirement related to account lockout is fulfilled by implementing and using the following three processes.

- Storing authentication failure event entity

  In case of authentication failure due to invalid authentication information input, the events generated by Spring Security are handled and the user name used for authentication and the date and time when authentication was attempted are registered in the database as authentication failure event entity.

- Determining lockout status

  For some accounts, if a certain number of new authentication failure event entities at the current time are more than a certain fixed number, the corresponding account is determined to be locked. Call this determination process during authentication and set the determination results in the implementation class of `UserDetails`.

- Deleting authentication failure event entity

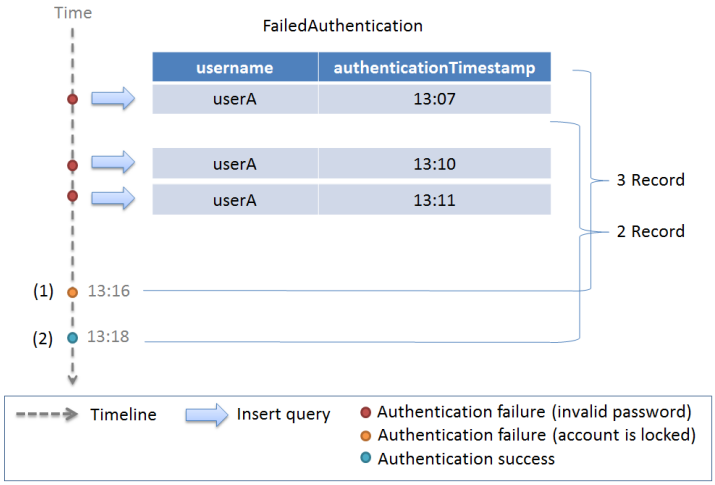
  Delete all authentication failure event entities for an account.

  Since an account is targeted for lockout only when it fails to authenticate continuously, delete the authentication failure event entity when authentication is successful.

  Also, since the lockout status of the account is determined using the authentication failure event entity, unlock function can be implemented by deleting the authentication failure event entity. Prevent account lockout from being executed by other than administrator using authorization function.

Warning: Since authentication failure event entity is intended only to determine lockout, it is deleted when it is no longer required. A separate log should be always saved when authentication log is required.

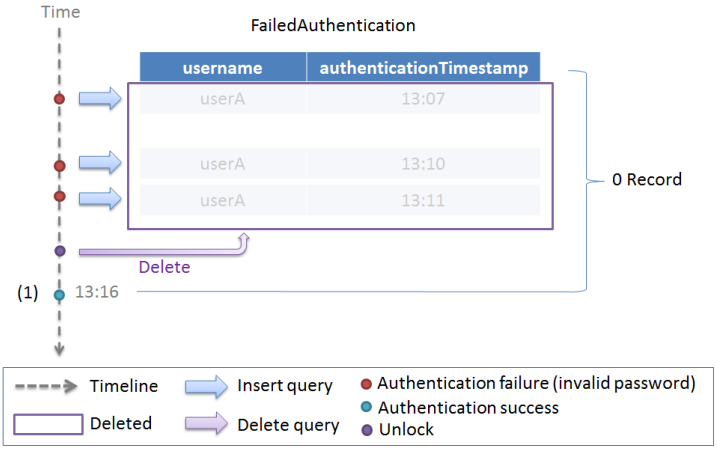
The working example of lockout function which uses authentication failure event entity is described with the help of the following figure. Lockout by authentication failure for 3 times and lockout duration of 10 minutes is considered as an example.



Sr. No.	Description
(1)	Authentication with incorrect password has been attempted three times in last 10 minutes, and authentication failure event entities for all the three occasions are stored in the database. Therefore, it is determined that the account is locked.
(2)	Authentication failure event entities for 3 occasions are stored in the database. However, since authentication failure event entities are only for the two occasions in last 10 minutes, the account is determined to be “not locked”.

Similarly, a working example for unlocking is described in the following figure.

Sr. No.	Description
(1)	Authentication with incorrect password has been attempted three times in last 10 minutes. Thereafter, since the authentication failure event entity is deleted, authentication failure event entity is not stored in the database and the account is determined as “not locked”.



Code description

- Common part

  In this application, registration, search and deletion of authentication failure event entity for the database is commonly required to implement the functions related to account lockout. Therefore, the implementation of domain layer / infrastructure layer related to the authentication failure event entity is shown first.

  - Implementation of Entity

    The implementation of authentication failure event entity with user name and date and time when authentication was attempted is shown below.

```
package org.terasoluna.securelogin.domain.model;

// omitted

@Data
public class FailedAuthentication implements Serializable {
  private static final long serialVersionUID = 1L;

  private String username; // (1)

  private LocalDateTime authenticationTimestamp; // (2)
}
```

Sr. No.	Description
(1)	User name used for authentication
(2)	Date and time when authentication was attempted

- Implementation of Repository

Repository to search, register and delete authentication failure event entity is shown below.

```
package org.terasoluna.securelogin.domain.repository.authenticationevent;

// omitted

public interface FailedAuthenticationRepository {

  int create(FailedAuthentication event); // (1)

  List<FailedAuthentication> findLatest(
                  @Param("username") String username, @Param("count") long␣
↪count); // (2)

  int deleteByUsername(@Param("username") String username); // (3)
}
```

Sr. No.	Description
(1)	A method to register `FailedAuthentication` object that is assigned as an argument, as a record in the database
(2)	A method to get specified number of `FailedAuthentication` objects in a new sequence by considering user name assigned as an argument, as the key
(3)	A method to delete the authentication failure event entity records collectively by considering user name assigned as an argument, as the key

Mapping file is as below.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<mapper
  namespace="org.terasoluna.securelogin.domain.repository.
↪authenticationevent.FailedAuthenticationRepository">

  <resultMap id="failedAuthenticationResultMap"
          type="FailedAuthentication">
          <id property="username" column="username" />
          <id property="authenticationTimestamp" column="authentication_
↪timestamp" />
```

```
  </resultMap>

  <insert id="create" parameterType="FailedAuthentication">
    <![CDATA[
        INSERT INTO failed_authentication (
            username,
            authentication_timestamp
        ) VALUES (
          #{username},
            #{authenticationTimestamp}
        )
    ]]>
  </insert>

  <select id="findLatest" resultMap="failedAuthenticationResultMap">
       <![CDATA[
            SELECT
                username,
                authentication_timestamp
            FROM
                failed_authentication
            WHERE
                username = #{username}
            ORDER BY authentication_timestamp DESC
            LIMIT #{count}
       ]]>
  </select>

  <delete id="deleteByUsername">
     <![CDATA[
          DELETE FROM
              failed_authentication
          WHERE
              username = #{username}
     ]]>
  </delete>
</mapper>
```

- Implementation of Service

The service to call the method of the created Repository is defined as below.

```
package org.terasoluna.securelogin.domain.service.authenticationevent;

// omitted

@Service
@Transactional
public class AuthenticationEventSharedServiceImpl implements
                AuthenticationEventSharedService {
```

```
    // omitted

    @Inject
    ClassicDateFactory dateFactory;

    @Inject
    FailedAuthenticationRepository failedAuthenticationRepository;

    @Inject
    AccountSharedService accountSharedService;

    @Transactional(readOnly = true)
    @Override
    public List<FailedAuthentication> findLatestFailureEvents(
                    String username, int count) {
            return failedAuthenticationRepository.
↪findLatestEvents(username, count);
    }


    @Transactional(propagation = Propagation.REQUIRES_NEW)
    @Override
    public void authenticationFailure(String username) { // (1)
         if (accountSharedService.exists(username)){
             FailedAuthentication failureEvents = new␣
↪FailedAuthentication();
             failureEvents.setUsername(username);
             failureEvents.setAuthenticationTimestamp(dateFactory.
↪newTimestamp()
                     .toLocalDateTime());

             failedAuthenticationRepository.create(failureEvents);
         }
     }

    @Override
    public int deleteFailureEventByUsername(String username) {
            return failedAuthenticationRepository.
↪deleteByUsername(username);
    }

    // omitted

}
```

Sr. No.	Description
(1)	A method to create authentication failure event entity and register in the database. If account of the user name received as an argument does not exist, skip the process of registration to the database since it violates the foreign key constraints of the database. Since it is likely that authentication failure event entity is not registered by the exception after executing this method, `REQUIRES_NEW` is specified in the propagation method of transaction.

The code implemented according to the implementation method is described below sequentially.

- Storing authentication failure event entity

  Use `@EventListener` annotation to execute the process by handling the event generated at the time of authentication failure. For handling of event by using `@EventListener` annotation, refer *Handling authentication event* .

```
package org.terasoluna.securelogin.domain.service.account;

// omitted

@Component
public class AccountAuthenticationFailureBadCredentialsEventListener{

    @Inject
    AuthenticationEventSharedService authenticationEventSharedService;

    @EventListener // (1)
    public void onApplicationEvent(
                    AuthenticationFailureBadCredentialsEvent event) {

        String username = (String) event.getAuthentication().getPrincipal(); //
↪(2)

        authenticationEventSharedService.authenticationFailure(username); //␣
↪(3)
    }

}
```

Sr. No.	Description
(1)	By assigning `@EventListener` annotation, when authentication fails due to invalid authentication information such as incorrect password etc., `onApplicationEvent` method is executed.
(2)	Fetch the user name used for authentication from `AuthenticationFailureBadCredentialsEvent` object.
(3)	Call the process to create authentication failure event entity and register in the database.

- Determining lockout status

The process to determine account lockout status using authentication failure event entity is described.

```
package org.terasoluna.securelogin.domain.service.account;

// omitted

@Service
@Transactional
public class AccountSharedServiceImpl implements AccountSharedService {

    // omitted

    @Inject
    ClassicDateFactory dateFactory;

    @Inject
    AuthenticationEventSharedService authenticationEventSharedService;

    @Value("${security.lockingDurationSeconds}") // (1)
    int lockingDurationSeconds;

    @Value("${security.lockingThreshold}") // (2)
    int lockingThreshold;

    @Transactional(readOnly = true)
    @Override
    public boolean isLocked(String username) {
        List<FailedAuthentication> failureEvents =␣
↪authenticationEventSharedService
                        .findLatestFailureEvents(username, lockingThreshold);␣
↪// (3)
```

```
        if (failureEvents.size() < lockingThreshold) { // (4)
            return false;
        }

        if (failureEvents
                .get(lockingThreshold - 1) // (5)
                .getAuthenticationTimestamp()
                .isBefore(
                        dateFactory.newTimestamp().toLocalDateTime()
                        .minusSeconds(lockingDurationSeconds))) {
            return false;
        }

        return true;
    }

    // omitted
}
```

Sr. No.	Description
(1)	Specify the lockout duration in seconds. The value defined in Property file is injected.
(2)	Specify the locking threshold. The account is locked when authentication fails only for the number of times specified here. The value defined in Property file is injected.
(3)	Fetch the authentication failure event entity in new sequence only for the number same as the locking threshold.
(4)	If the number of fetched authentication failure event entities is less than the locking threshold value, determine that the account is not locked.
(5)	If the difference between oldest authentication failure time from the fetched authentication failure event entities and current time is greater than the locking duration, determine that the account is not locked.

In `org.springframework.security.core.userdetails.User` which is the implementation class of `UserDetails` , lockout status can be passed to the constructor.

In this application, class that inherits `User` and class that implements `org.springframework.security.core.userdetails.UserDetailsService` are used as shown below.

```
package org.terasoluna.securelogin.domain.service.userdetails;

// omitted

public class LoggedInUser extends User {

   // omitted

   private final Account account;

   public LoggedInUser(Account account, boolean isLocked,
           LocalDateTime lastLoginDate,
           List<SimpleGrantedAuthority> authorities) {
       super(account.getUsername(), account.getPassword(), true, true, true,
                   !isLocked, authorities); // (1)
       this.account = account;

       // omitted
   }

    public Account getAccount() {
        return account;
    }

   // omitted
}
```

Sr. No.	Description
(1)	In the constructor of `User` which is the parent class, pass ** Whether the account is locked** in truth-value. Note that it is necessary to pass true if the account is not locked.

```
package org.terasoluna.securelogin.domain.service.userdetails;

// omitted

@Service
public class LoggedInUserDetailsService implements UserDetailsService {

    @Inject
```

```
    AccountSharedService accountSharedService;

    @Transactional(readOnly = true)
    @Override
    public UserDetails loadUserByUsername(String username)
            throws UsernameNotFoundException {
        try {
            Account account = accountSharedService.findOne(username);
            List<SimpleGrantedAuthority> authorities = new ArrayList<>();
            for (Role role : account.getRoles()) {
                authorities.add(new SimpleGrantedAuthority("ROLE_"
                        + role.getRoleValue()));
            }
            return new LoggedInUser(account,
                    accountSharedService.isLocked(username), // (1)
                    accountSharedService.getLastLoginDate(username),
                    authorities);
        } catch (ResourceNotFoundException e) {
            throw new UsernameNotFoundException("user not found", e);
        }
    }

}
```

Sr. No.	Description
(1)	In constructor of `LoggedInUser` , pass the determination result of lockout status using `isLocked` method.

Settings to use the created `UserDetailsService` are as follows:

spring-security.xml

```
<!-- omitted -->

<sec:authentication-manager>
    <sec:authentication-provider
        user-service-ref="loggedInUserDetailsService"> <!-- (1) -->
        <sec:password-encoder ref="passwordEncoder" />
    </sec:authentication-provider>
</sec:authentication-manager>

<!-- omitted -->
```

Sr. No.	Description
(1)	Specify Bean id for `UserDetailsService`.

- Deleting authentication failure event entity
  - Deleting authentication failure event entity when authentication is successful

    Since only consecutive authentication failures are used to determine lockout, delete the authentication failure event entity of the account when authentication is successful. Create the method to be executed when authentication is successful in the Service created as a common part.

```
package org.terasoluna.securelogin.domain.service.authenticationevent;

// omitted

@Service
@Transactional
public class AuthenticationEventSharedServiceImpl implements
                AuthenticationEventSharedService {

    // omitted

    @Transactional(propagation = Propagation.REQUIRES_NEW)
    @Override
    public void authenticationSuccess(String username) {

        // omitted

        deleteFailureEventByUsername(username); // (1)
    }

    // omitted

}
```

Sr. No.	Description
(1)	Delete the authentication failure event entity for the account of the user name passed as an argument.

Use `@EventListener` annotation to execute the process by handling the event generated when authentication is successful.

```
package org.terasoluna.securelogin.domain.service.account;
```

```
// omitted

@Component
public class AccountAuthenticationSuccessEventListener{

    @Inject
    AuthenticationEventSharedService authenticationEventSharedService;

    @EventListener // (1)
    public void onApplicationEvent(
                    AuthenticationSuccessEvent event) {

        LoggedInUser details = (LoggedInUser) event.getAuthentication()
                .getPrincipal();

        authenticationEventSharedService.authenticationSuccess(details.
↪getUsername()); // (2)

    }

}
```

Sr. No.	Description
(1)	By assigning `@EventListener` annotation, `onApplicationEvent` method is executed when authentication is successful.
(2)	Fetch user name from `AuthenticationSuccessEvent` and call the process to delete authentication failure event entity.

- Unlocking

Since authentication failure event entity is used to determine lockout status, an account can be unlocked by deleting the authentication failure event entity. Perform the authorization settings to restrict the usage of unlock function to the user having administrator rights and implement the domain layer / application layer.

* Authorization settings

Set the rights for the user who can unlock an account as below.

spring-security.xml

```
<!-- omitted -->
```

```
  <sec:http pattern="/resources/**" security="none" />
  <sec:http>

      <!-- omitted -->

      <sec:intercept-url pattern="/unlock/**" access="hasRole('ADMIN')
↪" /> <!-- (1) -->

      <!-- omitted -->

  </sec:http>

<!-- omitted -->
```

Sr. No.	Description
(1)	Restrict the access rights for URL under /unlock to the administrator.

* Implementation of Service

```
package org.terasoluna.securelogin.domain.service.unlock;

// omitted

@Transactional
@Service
public class UnlockServiceImpl implements UnlockService {

    @Inject
    AccountSharedService accountSharedService;

    @Inject
    AuthenticationEventSharedService authenticationEventSharedService;

    @Override
    public void unlock(String username) {
        authenticationEventSharedService.
↪deleteFailureEventByUsername(username); // (1)
    }

}
```

Sr. No.	Description
(1)	Unlock an account by deleting the authentication failure event entity.

* Implementation of Form

```
package org.terasoluna.securelogin.app.unlock;

@Data
public class UnlockForm implements Serializable {

    private static final long serialVersionUID = 1L;

    @NotEmpty
    private String username;
}
```

* Implementation of View

Top screen(home.jsp)

```
<!-- omitted -->

<body>
    <div id="wrapper">

        <!-- omitted -->

        <sec:authorize url="/unlock"> <!-- (1) -->
        <div>
            <a id="unlock" href="${f:h(pageContext.request.
↪contextPath)}/unlock?form">
                Unlock Account
            </a>
        </div>
        </sec:authorize>

        <!-- omitted -->

    </div>
</body>

<!-- omitted -->
```

Sr. No.	Description
(1)	Display only for the user who has access rights, under /unlock.

Unlock form(unlokcForm.jsp)

```
<!-- omitted -->

<body>
    <div id="wrapper">
```

```
        <h1>Unlock Account</h1>
        <t:messagesPanel />
        <form:form action="${f:h(pageContext.request.contextPath)}/
↪unlock"
            method="POST" modelAttribute="unlockForm">
            <table>
                <tr>
                    <th><form:label path="username" cssErrorClass=
↪"error-label">Username</form:label>
                    </th>
                    <td><form:input path="username" cssErrorClass=
↪"error-input" /></td>
                    <td><form:errors path="username" cssClass="error-
↪messages" /></td>
                </tr>
            </table>

            <input id="submit" type="submit" value="Unlock" />
        </form:form>
        <a href="${f:h(pageContext.request.contextPath)}/">go to Top</
↪a>
    </div>
</body>

<!-- omitted -->
```

Unlock completion screen(unlockComplete.jsp)

```
<!-- omitted -->

<body>
    <div id="wrapper">
          <h1>${f:h(username)}'s account was successfully unlocked.</
↪h1>
          <a href="${f:h(pageContext.request.contextPath)}/">go to Top
↪</a>
    </div>
</body>

<!-- omitted -->
```

* Implementation of Controller

```
package org.terasoluna.securelogin.app.unlock;

// omitted

@Controller
@RequestMapping("/unlock") // (1)
public class UnlockController {
```

```
    @Inject
    UnlockService unlockService;

    @RequestMapping(params = "form")
    public String showForm(UnlockForm form) {
        return "unlock/unlockForm";
    }

    @RequestMapping(method = RequestMethod.POST)
    public String unlock(@Validated UnlockForm form,
            BindingResult bindingResult, Model model,
            RedirectAttributes attributes) {
        if (bindingResult.hasErrors()) {
                return showForm(form);
        }

        try {
            unlockService.unlock(form.getUsername()); // (2)
            attributes.addFlashAttribute("username", form.
↪getUsername());
            return "redirect:/unlock?complete";
        } catch (BusinessException e) {
            model.addAttribute(e.getResultMessages());
            return showForm(form);
        }
    }

    @RequestMapping(method = RequestMethod.GET, params = "complete")
    public String unlockComplete() {
        return "unlock/unlockComplete";
    }

}
```

Sr. No.	Description
(1)	Map to the URL under /unlock. It can be accessed by administrator only depending on the authorization settings.
(2)	Call the process to unlock an account by considering the user name obtained from the Form, as an argument.

Display the date and time of last login

List of requirements to be implemented

- *Display of previous login date and time*

Working image



Implementation method

In this application, the history when authentication is successful is stored in the database as an "authentication successful event" entity, and date and time of previous login for the account is displayed on the top screen using this authentication successful event entity.
In particular, fulfil the requirements by implementing the following two processes.

- Storing authentication successful event entity

  Handle the event generated by Spring Security when authentication is successful and register the username used for authentication and the date and time when authentication was successful in the database, as an authentication successful event entity.

- Fetch and display date and time of previous login

  At the time of authentication, fetch the latest authentication successful event entity in the account from the database, fetch the authentication successful date and time from the event entity and set in `org.springframework.security.core.userdetails.UserDetails` . Format, pass and display the authentication successful date and time retained by `UserDetails` in jsp.

Code description

- Common part

In this application, the authentication successful event entity must be registered and searched for the database in order to display previous login date and time. Therefore, the implementation of domain layer / infrastructure layer related to the authentication successful event entity is described first.

- Implementation of Entity

  The implementation of authentication successful event entity with user name and date and time when authentication was successful is as below.

```
package org.terasoluna.securelogin.domain.model;

// omitted

@Data
public class SuccessfulAuthentication implements Serializable {

    private static final long serialVersionUID = 1L;

    private String username; // (1)

    private LocalDateTime authenticationTimestamp; // (2)

}
```

Sr. No.	Description
(1)	User name used for authentication
(2)	Date and time when authentication is attempted

- Implementation of Repository

  Repository to search and register authentication successful event entity is shown below.

```
package org.terasoluna.securelogin.domain.repository.authenticationevent;

// omitted

public interface SuccessfulAuthenticationRepository {

    int create(SuccessfulAuthentication event); // (1)

    List<SuccessfulAuthentication> findLatestEvents(
            @Param("username") String username, @Param("count") long count);
 ↪ // (2)
}
```

Sr. No.	Description
(1)	A method to register `SuccessfulAuthentication` object that is assigned as an argument, as a record in the database
(2)	A method to fetch specified number of `SuccessfulAuthentication` objects in new sequence by considering user name assigned as an argument, as a key

Mapping file is as below.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<mapper
    namespace="org.terasoluna.securelogin.domain.repository.
↪authenticationevent.SuccessfulAuthenticationRepository">

    <resultMap id="successfulAuthenticationResultMap"
            type="SuccessfulAuthentication">
        <id property="username" column="username" />
        <id property="authenticationTimestamp" column="authentication_
↪timestamp" />
    </resultMap>

    <insert id="create" parameterType="SuccessfulAuthentication">
    <![CDATA[
        INSERT INTO successful_authentication (
            username,
            authentication_timestamp
        ) VALUES (
            #{username},
            #{authenticationTimestamp}
        )
    ]]>
    </insert>

    <select id="findLatestEvents" resultMap=
↪"successfulAuthenticationResultMap">
    <![CDATA[
        SELECT
            username,
            authentication_timestamp
        FROM
            successful_authentication
        WHERE
```

```
            username = #{username}
        ORDER BY authentication_timestamp DESC
        LIMIT #{count}
    ]]>
    </select>
</mapper>
```

- Implementation of Service

The service to call the methods of the created Repository is shown below.

```
package org.terasoluna.securelogin.domain.service.authenticationevent;

// omitted

@Service
@Transactional
public class AuthenticationEventSharedServiceImpl implements
         AuthenticationEventSharedService {

    // omitted

    @Inject
    ClassicDateFactory dateFactory;

    @Inject
    SuccessfulAuthenticationRepository successAuthenticationRepository;

    @Transactional(readOnly = true)
    @Override
    public List<SuccessfulAuthentication> findLatestSuccessEvents(
                    String username, int count) {
        return successAuthenticationRepository.findLatestEvents(username,␣
↪count);
    }

    @Transactional(propagation = Propagation.REQUIRES_NEW)
    @Override
      public void authenticationSuccess(String username) {
          SuccessfulAuthentication successEvent = new␣
↪SuccessfulAuthentication();
          successEvent.setUsername(username);
          successEvent.setAuthenticationTimestamp(dateFactory.
↪newTimestamp()
                  .toLocalDateTime());

          successAuthenticationRepository.create(successEvent);
          deleteFailureEventByUsername(username);
      }
```

```
}
```

The code implemented according to the implementation method is described below sequentially.

- Storing authentication successful event entity

  Use `@EventListener` annotation to execute the process by handling the event generated when authentication is successful.

```
package org.terasoluna.securelogin.domain.service.account;

// omitted

@Component
public class AccountAuthenticationSuccessEventListener{

    @Inject
    AuthenticationEventSharedService authenticationEventSharedService;

    @EventListener // (1)
    public void onApplicationEvent(AuthenticationSuccessEvent event) {
        LoggedInUser details = (LoggedInUser) event.getAuthentication()
                        .getPrincipal(); // (2)

        authenticationEventSharedService.authenticationSuccess(details
                .getUsername()); // (3)
    }

}
```

Sr. No.	Description
(1)	By assigning `@EventListener` annotation, `onApplicationEvent` method is executed when authentication is successful.
(2)	Fetch implementation class of `UserDetails` from `AuthenticationSuccessEvent` object. This class is described later.
(3)	Call the process to create authentication successful event entity and register in the database.

- Fetch and display date and time of previous login

  The Service to fetch date and time of previous login from authentication successful event entity is shown

below.

```
package org.terasoluna.securelogin.domain.service.account;

// omitted

@Service
@Transactional
public class AccountSharedServiceImpl implements AccountSharedService {

    // omitted

    @Inject
    AuthenticationEventSharedService authenticationEventSharedService;

    @Transactional(readOnly = true)
    @Override
    public LocalDateTime getLastLoginDate(String username) {
        List<SuccessfulAuthentication> events =
↪authenticationEventSharedService
                    .findLatestSuccessEvents(username, 1); // (1)

        if (events.isEmpty()) {
            return null; // (2)
        } else {
            return events.get(0).getAuthenticationTimestamp(); // (3)
        }
    }

    // omitted

}
```

Sr. No.	Description
(1)	Fetch one record of the latest authentication successful event entity by considering the user name assigned as an argument, as the key.
(2)	Return null if even a single record of authentication successful event entity could not be fetched at the time of initial login.
(3)	Fetch and return authentication date and time from authentication successful event entity.

Create a class that inherits `User` and a class that implements `UserDetailsService` as shown below

to fetch the date and time of previous login and retain it in `UserDetails` at the time of login.

```
package org.terasoluna.securelogin.domain.service.userdetails;

// omitted

public class LoggedInUser extends User {

    private final Account account;

    private final LocalDateTime lastLoginDate; // (1)

    public LoggedInUser(Account account, boolean isLocked,
            LocalDateTime lastLoginDate,
            List<SimpleGrantedAuthority> authorities) {

        super(account.getUsername(), account.getPassword(), true, true, true,
                        !isLocked, authorities);
        this.account = account;
        this.lastLoginDate = lastLoginDate; // (2)
    }

    // omitted

    public LocalDateTime getLastLoginDate() { // (3)
        return lastLoginDate;
    }

}
```

Sr. No.	Description
(1)	Declare a field to retain the date and time of previous login.
(2)	Set the date and time of previous login assigned as an argument in the field.
(3)	A method to return the retained date and time of previous login

```
package org.terasoluna.securelogin.domain.service.userdetails;

// omitted

@Service
public class LoggedInUserDetailsService implements UserDetailsService {
```

```
    @Inject
    AccountSharedService accountSharedService;

    @Transactional(readOnly = true)
    @Override
    public UserDetails loadUserByUsername(String username)
                throws UsernameNotFoundException {
        try {
            Account account = accountSharedService.findOne(username);
            List<SimpleGrantedAuthority> authorities = new ArrayList<>();
            for (Role role : account.getRoles()) {
                    authorities.add(new SimpleGrantedAuthority("ROLE_"
                                    + role.getRoleValue()));
            }
            return new LoggedInUser(account,
                            accountSharedService.isLocked(username),
                            accountSharedService.getLastLoginDate(username), //
↪(1)
                            authorities);
        } catch (ResourceNotFoundException e) {
            throw new UsernameNotFoundException("user not found", e);
        }
    }

}
```

Sr. No.	Description
(1)	Fetch date and time of previous login by calling Service method and pass it to constructor of `LoggedInUser`.

Implement application layer to display date and time of previous login on the top screen.

```
package org.terasoluna.securelogin.app.welcome;

// omitted

@Controller
public class HomeController {

   @Inject
   AccountSharedService accountSharedService;

   @RequestMapping(value = "/", method = { RequestMethod.GET,
           RequestMethod.POST })
   public String home(@AuthenticationPrincipal LoggedInUser userDetails, //␣
↪(1)
```

```
            Model model) {

        // omitted

        LocalDateTime lastLoginDate = userDetails.getLastLoginDate(); // (2)
        if (lastLoginDate != null) {
            model.addAttribute("lastLoginDate", lastLoginDate
                    .format(DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss
↪")));  // (3)
        }

        return "welcome/home";

    }

}
```

Sr. No.	Description
(1)	Fetch UserDetails object by using `@AuthenticationPrincipal` .
(2)	Fetch date and time of last login from `LoggedInUserDetails` .
(3)	Format date and time of last login, set it in Model and pass to View.

Top screen(home.jsp)

```
<body>
  <div id="wrapper">

      <!-- omitted -->

      <c:if test="${!empty lastLoginDate}"> <!-- (1) -->
          <p id="lastLogin">
              Last login date is ${f:h(lastLoginDate)}. <!-- (2) -->
          </p>
      </c:if>

      <!-- omitted -->

  </div>
</body>
```

Sr. No.	Description
(1)	Do not display if date and time of previous login is null.
(2)	Display the date and time of previous login passed from the Controller.

Creating authentication information for password reissue

List of requirements to be implemented

- *Assign random string to the URL for password reissue*
- *Issue confidential information for password reissue*

Working image



Enter the user name for which password is to be reissued on the screen to generate authentication information for password reissue. At this time, the confidential information and token to be used for authentication during password reissue, are generated. Confidential information is displayed on the screen and the URL for password reissue screen containing the token is sent to the registered e-mail address of the user.

There is an expiry date to the URL sent by e-mail. Password can be changed by accessing the URL within the expiry date and entering the confidential information and the new password. If the URL sent by e-mail is accessed after the expiry date, the user is taken to the error screen.

Confidential information and token generation is described from the flow mentioned above.

Implementation method

While reissuing the password, an alternative to password is required to verify that the user is the owner of the account.

In this application, URL of password reissue screen and confidential information are used as the information to verify the user.

Create a random string and add it to URL to make the password reissue screen URL unique and difficult to guess. Create confidential information which is in the form of a random string and use it for authentication as a measure against accidental leakage of URL.

Create two random strings by different ways so that that it becomes impossible to guess a string from the other string.

In particular, fulfil the requirements by implementing the following process.

- Creating and saving authentication information for password reissue

  Store the following information as the authentication information for password reissue in the database.

  - User name: User name of the account for which password is to be reissued
  - Token: Random string generated to make the password reissue screen URL unique and difficult to guess
  - Confidential information: Random string generated for user input at the time of password reissue
  - Expiry date: Expiry date for authentication information for password reissue

  Use `randomUUID` method of `java.util.UUID` class for token generation and Password generation function of Passay for generating confidential information.

  Save the confidential information to the database by hashing similar to password. Expiry date settings and confirmation process are described in *Validation at the time of executing password reissue*. Refer to *Distribute authentication information for password reissue* for the method to distribute authentication information for password reissue to the user.

Code description

- Common part

In the implementation according to the implementation method mentioned above, the process to register and search the authentication information for password reissue in the database is commonly required. Therefore, implementation of Entity and Repository related to the authentication information for password reissue is described first.

- Creation of Entity

  Create an Entity of authentication information for password reissue.

```
package org.terasoluna.securelogin.domain.model;

// omitted

@Data
public class PasswordReissueInfo {

    private String username; // (1)

    private String token; // (2)

    private String secret; // (3)

    private LocalDateTime expiryDate; // (4)

}
```

Sr. No.	Description
(1)	User name for password reissue
(2)	String that is generated to be included in the URL for password reissue (Token)
(3)	String to verify the user at the time of password reissue (Confidential information)
(2)	Expiry date for authentication information for password reissue

- Implementation of Repository

  Repository to search, register and delete the authentication information for password reissue is shown below.

```
package org.terasoluna.securelogin.domain.repository.passwordreissue;

// omitted

public interface PasswordReissueInfoRepository {

    void create(PasswordReissueInfo info); // (1)

    PasswordReissueInfo findOne(@Param("token") String token); // (2)

    int delete(@Param("token") String token); // (3)

    // omitted

}
```

Sr. No.	Description
(1)	A method to register `PasswordReissueInfo` object that is assigned as an argument, as a record in the database
(2)	A method to search and fetch `PasswordReissueInfo` object by considering the token assigned as an argument, as the key
(3)	A method to delete `PasswordReissueInfo` object by considering the token assigned as an argument, as the key

Mapping file is as below.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<mapper
    namespace="org.terasoluna.securelogin.domain.repository.
↪passwordreissue.PasswordReissueInfoRepository">

    <resultMap id="PasswordReissueInfoResultMap" type="PasswordReissueInfo
↪">
        <id property="username" column="username" />
        <id property="token" column="token" />
        <id property="secret" column="secret" />
        <id property="expiryDate" column="expiry_date" />
```

```
    </resultMap>

    <select id="findOne" resultMap="PasswordReissueInfoResultMap">
    <![CDATA[
        SELECT
            username,
            token,
            secret,
            expiry_date
        FROM
            password_reissue_info
        WHERE
            token = #{token}
    ]]>
    </select>

    <insert id="create" parameterType="PasswordReissueInfo">
    <![CDATA[
        INSERT INTO password_reissue_info (
            username,
            token,
            secret,
            expiry_date
        ) VALUES (
            #{username},
            #{token},
            #{secret},
            #{expiryDate}
        )
    ]]>
    </insert>

    <delete id="delete">
    <![CDATA[
        DELETE FROM
            password_reissue_info
        WHERE
            token = #{token}
    ]]>
    </delete>

    <!-- omitted -->

</mapper>
```

The code implemented according to the implementation method is described below.

- Generating and storing authentication information for password reissue
  - Definition of password generator

The definition of password generator and generation rules to use the password generation function of Passay is shown below. Refer to *Password generation* for the password generator and generation rules.

Sr. No.	Description
(1)	Define a Bean for password generator to be used in password generation function of Passay
(2)	Define a Bean for password generation rules to be used in password generation function of Passay. Using the validation rules that were used in *Check password strength*, define generation rules for the password containing one or more characters of single-byte upper case letters, single-byte lower case letters and single-byte digits respectively.

applicationContext.xml

```
<bean id="passwordGenerator" class="org.passay.PasswordGenerator" /> <!--␣
↪(1) -->
<util:list id="passwordGenerationRules">
    <ref bean="upperCaseRule" />
    <ref bean="lowerCaseRule" />
    <ref bean="digitRule" />
</util:list>
```

– Implementation of Service

The implementation of the process to create authentication information for password reissue and store in the database is shown below. The authentication information generated in this process is sent by e-mail. Sending the information by e-mail is omitted here as it is described later.

```
package org.terasoluna.securelogin.domain.service.passwordreissue;

// omitted

@Service
@Transactional
public class PasswordReissueServiceImpl implements PasswordReissueService {

    @Inject
    ClassicDateFactory dateFactory;

    @Inject
    PasswordReissueInfoRepository passwordReissueInfoRepository;

    @Inject
    AccountSharedService accountSharedService;
```

```
    @Inject
    PasswordEncoder passwordEncoder;

    @Inject
    PasswordGenerator passwordGenerator; // (1)

    @Resource(name = "passwordGenerationRules")
    List<CharacterRule> passwordGenerationRules; //(2)

    @Value("${security.tokenLifeTimeSeconds}")
    int tokenLifeTimeSeconds; // (3)

    // omitted

    @Override
    public String createAndSendReissueInfo(String username) {

        String rowSecret = passwordGenerator.generatePassword(10,␣
↪passwordGenerationRules); // (4)

        if(!accountSharedService.exists(username)){ // (5)
            return rowSecret;
        }

        Account account= accountSharedService.findOne(username); // (6)

        String token = UUID.randomUUID().toString(); // (7)

        LocalDateTime expiryDate = dateFactory.newTimestamp().
↪toLocalDateTime()
                .plusSeconds(tokenLifeTimeSeconds); // (8)

        PasswordReissueInfo info = new PasswordReissueInfo(); // (9)
        info.setUsername(username);
        info.setToken(token);
        info.setSecret(passwordEncoder.encode(rowSecret)); // (10)
        info.setExpiryDate(expiryDate);

        passwordReissueInfoRepository.create(info); // (11)

        // omitted (Send E-Mail)

        return rowSecret; // (12)

    }

    // omitted

}
```

Sr. No.	Description
(1)	Inject a password generator to be used in password generation function of Passay.
(2)	Inject password generation rules to be used in password generation function of Passay.
(3)	Specify the length of the period for which authentication information for password reissue is valid, in seconds. The value defined in the property file is injected.
(4)	Create a random string of length 10 in accordance with the password generation rules using the password generation function of Passay to use as confidential information.
(5)	Check whether the account of user name that is passed as an argument, exists. If it does not exist, return dummy confidential information since non-existence of the user is not known.
(6)	Fetch the account information of the user name included in the authentication information for password reissue.
(7)	Create a random string using `randomUUID` method of `java.util.UUID` class to use as a token.
(8)	By adding the value of (3) to current time, calculate expiry date for the authentication information for password reissue.
(9)	Create authentication information for password reissue and set the user name, token, confidential information and expiry date.
(10)	Set confidential information in `PasswordReissueInfo` after hashing it.

次のページに続く

Table. 9.43 – 前のページからの続き

Sr. No.	Description
(11)	Register the authentication information for password reissue in the database.
(12)	Return the created confidential information.

- Implementation of Form

```
package org.terasoluna.securelogin.app.passwordreissue;

// omitted

@Data
public class CreateReissueInfoForm implements Serializable {

    private static final long serialVersionUID = 1L;

    @NotEmpty
    private String username;
}
```

- Implementation of View

Screen to create authentication information for password reissue(createReissueInfoForm.xml)

```
<!-- omitted -->

<body>
    <div id="wrapper">
        <h1>Reissue password</h1>
        <t:messagesPanel />
        <form:form
            action="${f:h(pageContext.request.contextPath)}/reissue/create"
            method="POST" modelAttribute="createReissueInfoForm">
            <table>
                <tr>
                    <th><form:label path="username" cssErrorClass="error-
↪label">Username</form:label>
                    </th>
                    <td><form:input path="username" cssErrorClass="error-
↪input" /></td>
                    <td><form:errors path="username" cssClass="error-
↪messages" /></td>
                </tr>
            </table>

            <input id="submit" type="submit" value="Reissue password" />
        </form:form>
    </div>
</body>

<!-- omitted -->
```

- Implementation of Controller

```
package org.terasoluna.securelogin.app.passwordreissue;

// omitted

@Controller
@RequestMapping("/reissue")
public class PasswordReissueController {

    @Inject
    PasswordReissueService passwordReissueService;

    @RequestMapping(value = "create", params = "form")
    public String showCreateReissueInfoForm(CreateReissueInfoForm form) {
        return "passwordreissue/createReissueInfoForm";
    }

    @RequestMapping(value = "create", method = RequestMethod.POST)
    public String createReissueInfo(@Validated CreateReissueInfoForm form,
            BindingResult bindingResult, Model model,
            RedirectAttributes attributes) {
        if (bindingResult.hasErrors()) {
            return showCreateReissueInfoForm(form);
        }

        String rawSecret = passwordReissueService.
↪createAndSendReissueInfo(form.getUsername()); // (1)
        attributes.addFlashAttribute("secret", rawSecret);
        return "redirect:/reissue/create?complete";
    }

    @RequestMapping(value = "create", params = "complete", method =␣
↪RequestMethod.GET)
    public String createReissueInfoComplete() {
        return "passwordreissue/createReissueInfoComplete";
    }

    // omitted

}
```

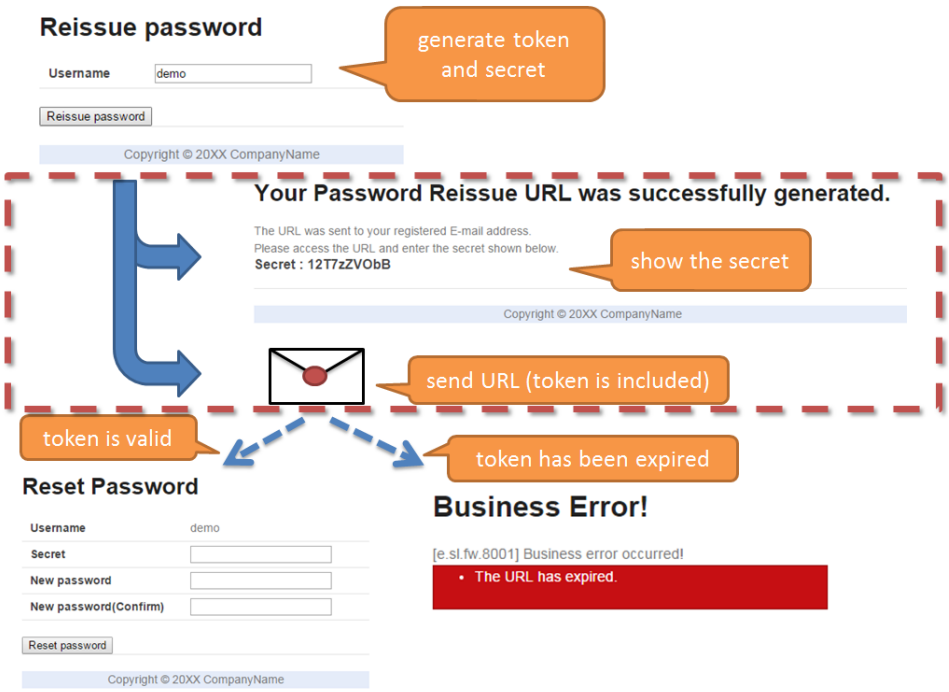
Sr. No.	Description
(1)	Create authentication information for password reissue from the user name fetched from Form and call the process registered in the database.

Distribution of authentication information for password reissue

List of requirements to be implemented

- *Separate distribution for password reissue screen URL and confidential information*
- *Send e-mail for URL of password reissue screen*

Working image



Creation of authentication information for password reissue is described in *Creating authentication information for password reissue*. The distribution of the created authentication information is described here.

Perform the authentication for password reissue using the password reissue screen URL and confidential information. Distribute the information to the user using different methods to prevent the leakage of the information at the same time. In this application, URL of the password reissue screen is sent to the registered e-mail address of the user, and confidential information is displayed on the screen.

Implementation method

Split the authentication information created using *Create authentication information for password reissue* and distribute to the user using separate methods.

Fulfil the requirements by implementing and using the following two processes.

- Display confidential information on screen

Distribute the confidential information before hashing that is created using *Create authentication information for password reissue* to the user by displaying it on the screen.

- Send an e-mail for URL of password reissue screen

  Send the URL of password reissue screen including token that is created using *Create authentication information for password reissue* by e-mail using the component for Spring Framework Mail linkage.

Code description

The code implemented according to the above implementation method is described sequentially.

- Displaying confidential information on screen

  A series of implementations to call the process to create confidential information from the Controller and display it in View is shown below.

```
package org.terasoluna.securelogin.app.passwordreissue;

// omitted

@Controller
@RequestMapping("/reissue")
public class PasswordReissueController {

    @Inject
    PasswordReissueService passwordReissueService;

    // omitted

    @RequestMapping(value = "create", method = RequestMethod.POST)
    public String createReissueInfo(@Validated CreateReissueInfoForm form,
            BindingResult bindingResult, Model model,
            RedirectAttributes attributes) {
        if (bindingResult.hasErrors()) {
            return showCreateReissueInfoForm(form);
        }

        String rawSecret = passwordReissueService.
↪createAndSendReissueInfo(form
                .getUsername()); // (1)
        attributes.addFlashAttribute("secret", rawSecret); // (2)
        return "redirect:/reissue/create?complete"; // (3)
    }

    @RequestMapping(value = "create", params = "complete", method =␣
↪RequestMethod.GET)
    public String createReissueInfoComplete() {
        return "passwordreissue/createReissueInfoComplete";
    }
```

```
    // omitted

}
```

Sr. No.	Description
(1)	Call the process to create confidential information.
(2)	Using RedirectAttributes, pass the confidential information to the redirect destination.
(3)	Redirect to completion screen of authentication information for password reissue.

Screen to complete creation of authentication information for password reissue(createReissueInfoComplete.jsp)

```
<!-- omitted -->

<body>
    <div id="wrapper">
        <h1>Your Password Reissue URL was successfully generated.</h1>
        The URL was sent to your registered E-mail address.<br /> Please
        access the URL and enter the secret shown below.
        <h3>Secret : <span id=secret>${f:h(secret)}</span></h3> <!-- (1) -->
    </div>
</body>

<!-- omitted -->
```

Sr. No.	Description
(1)	Display confidential information on the screen.

- Send a mail for URL of password reissue screen

  The implementation of the process to create URL of password reissue screen from the authentication information for password reissue screen and send it by e-mail is shown below. Refer to *Sending E-mail (SMTP)* for more information about how to add dependent libraries and how to fetch an e-mail session.

```
package org.terasoluna.securelogin.domain.service.mail;
```

```
// omitted

@Service
public class PasswordReissueMailSharedServiceImpl implements
        PasswordReissueMailSharedService {

    @Inject
    JavaMailSender mailSender; // (1)

    @Inject
    @Named("passwordReissueMessage")
    SimpleMailMessage templateMessage; // (2)

    // omitted

    @Override
    public void send(String to, String text) {
        SimpleMailMessage message = new SimpleMailMessage(templateMessage); //
 (3)
        message.setTo(to);
        message.setText(text);
        mailSender.send(message);
    }

}
```

Sr. No.	Description
(1)	Inject a Bean for `org.springframework.mail.javamail.JavaMailSender`.
(2)	Inject a Bean for `org.springframework.mail.SimpleMailMessage` in which email address of the source and e-mail title are set. In this application, only one Bean is defined for `SimpleMailMessage`, however, since multiple Beans are generally defined as a mail template, Bean name is specified by `@Named`.
(3)	Create an instance of `SimpleMailMessage` from the template, set the destination email address and text assigned as arguments and send.

```
package org.terasoluna.securelogin.domain.service.passwordreissue;

// omitted

@Service
```

```
@Transactional
public class PasswordReissueServiceImpl implements PasswordReissueService {

    @Inject
    ClassicDateFactory dateFactory;

    @Inject
    PasswordReissueMailSharedService mailSharedService;

    @Inject
    AccountSharedService accountSharedService;

    @Inject
    PasswordEncoder passwordEncoder;

    @Value("${security.tokenLifeTimeSeconds}")
    int tokenLifeTimeSeconds;

    @Value("${app.applicationBaseUrl}") // (1)
    String baseUrl;

    @Value("${app.passwordReissueProtocol}")
    String protocol;

    // omitted

    @Override
    public String createAndSendReissueInfo(String username) {

        String rowSecret = passwordGenerator.generatePassword(10,
                passwordGenerationRules);

        if(!accountSharedService.exists(username)){
            return rowSecret;
        }

        Account account= accountSharedService.findOne(username);

        String token = UUID.randomUUID().toString();

        LocalDateTime expiryDate = dateFactory.newTimestamp().
↪toLocalDateTime()
                .plusSeconds(tokenLifeTimeSeconds);

        PasswordReissueInfo info = new PasswordReissueInfo();
        info.setUsername(username);
        info.setToken(token);
        info.setSecret(passwordEncoder.encode(rowSecret));
        info.setExpiryDate(expiryDate);
```

```
        passwordReissueInfoRepository.create(info);

        UriComponentsBuilder uriBuilder = UriComponentsBuilder.
↪fromUriString(baseUrl);
        uriBuilder.pathSegment("reissue").pathSegment("resetpassword")
                .queryParam("form").queryParam("token", info.getToken());  //␣
↪(2)
        String passwordResetUrl = uriBuilder.build().encode().toUriString();

        mailSharedService.send(account.getEmail(), passwordResetUrl); // (3)

        return rowSecret;

    }

    // omitted

}
```

Sr. No.	Description
(1)	Fetch the base URL to be used in the password reissue screen URL from the property file.
(2)	Using the value fetched in (1) and the token included in the created authentication information for password reissue, create the URL of password reissue screen to be distributed to the user. Use `org.springframework.web.util.UriComponentsBuilder` to create the URL. `UriComponentsBuilder` is described in *Implementing hypermedia link*.
(3)	Send an email with the URL of the password reissue screen mentioned in the mail text to the registered e-mail address of the user.

Validation at the time of executing password reissue

List of requirements to be implemented

- *Setting validity period for authentication information for password reissue*



Working image

The distribution of authentication information for password reissue is described in *Distribution of authentication information for password reissue*. The process for using the distributed authentication information is described below.

As the authentication at the time of password reissue, verify the confidential information and URL of password reissue screen distributed separately in *Distribution of authentication information for password reissue*. Password is reissued only if the set of token and confidential information that is included in the URL is correct.

Moreover, set expiry date to the authentication information so that it is not valid for a long period of time unnecessarily as generally password is reissued immediately after the creation of authentication information. When the URL for password reissue screen is accessed, password reissue screen is displayed if the authentication information is within the expiry date and transits to error screen after the expiry date.

Implementation method

Token is included as a request parameter in the URL for password reissue screen sent by e-mail. Fetch the token when the password reissue screen is accessed and search the authentication information for password reissue from the database by considering this token as the key.

At the time of creating authentication information, set the expiry date in advance and check for expiration when it is obtained from the database. If it is within the expiry date, display the change password screen and accept the input for confidential information and new password.

If confidential information in the authentication information obtained from the database and the confidential

information entered by the user are identical, then authentication is successful and password is reissued.

In particular, fulfil the requirements by implementing the following three processes.

- Setting expiry date for authentication information for password reissue

  Set expiry date to the authentication information created in the process described in *Creating authentication information for password reissue*.

- Check expiry date for authentication information for password reissue

  When the password reissue screen is accessed, fetch the token included in the request parameter and search the authentication information for password reissue stored in the database considering the token as the key. Compare the expiry date included in authentication information with current time and after expiry date, transit to the error screen.

- User verification using authentication information for password reissue

  When reissuing the password, check whether the combination of user name, token and the confidential information provided by the user and the authentication information in the database are identical. If they are identical, reissue the password. If they are not identical, display the error message.

Code description

- Setting expiry date for authentication information for password reissue

  The settings for expiry date to the authentication information for password reissue are included in the process described in *Creating authentication information for password reissue*. Here, only relevant implementation points are shown again.

  - Implementation of Service

```
package org.terasoluna.securelogin.domain.service.passwordreissue;

// omitted

@Service
@Transactional
public class PasswordReissueServiceImpl implements PasswordReissueService {

    @Inject
    ClassicDateFactory dateFactory;

    @Inject
    PasswordReissueInfoRepository passwordReissueInfoRepository;

    @Value("${security.tokenLifeTimeSeconds}")
    int tokenLifeTimeSeconds; // (1)
```

```
    // omitted

    @Override
    public String createAndSendReissueInfo(String username) {

        // omitted

        LocalDateTime expiryDate = dateFactory.newTimestamp().
↪toLocalDateTime()
                .plusSeconds(tokenLifeTimeSeconds); // (2)

        PasswordReissueInfo info = new PasswordReissueInfo(); // (3)
        info.setUsername(username);
        info.setToken(token);
        info.setSecret(passwordEncoder.encode(rowSecret));
        info.setExpiryDate(expiryDate);

        passwordReissueInfoRepository.create(info); // (4)

        // omitted (Send E-Mail)

    }

    // omitted

}
```

Sr. No.	Description
(1)	Specify the length of the period for which authentication information for password reissue is valid, in seconds. The value defined in property file is injected.
(2)	By adding the value of (1) to current time, calculate expiry date for the authentication information for password reissue.
(3)	Create authentication information for password reissue and set the user name, token, confidential information and expiry date.
(4)	Register the authentication information for password reissue in the database.

- Check expiry date of authentication information for password reissue

  The implementation of the process to fetch authentication information for password reissue from the token included in the URL as a request parameter and check whether it is within the expiry date when the password reissue screen is accessed, is shown below. In this process, it is also checked whether the maximum limit for password reissue failure has exceeded. However, it is omitted here and will be described later.

  - Implementation of Service

```
package org.terasoluna.securelogin.domain.service.passwordreissue;

// omitted

@Service
@Transactional
public class PasswordReissueServiceImpl implements PasswordReissueService {

    @Inject
    ClassicDateFactory dateFactory;

    @Inject
    PasswordReissueInfoRepository passwordReissueInfoRepository;

    // omitted

    @Override
    @Transactional(readOnly = true)
    public PasswordReissueInfo findOne(String token) {
        PasswordReissueInfo info = passwordReissueInfoRepository.
↪findOne(token); // (1)

        if (info == null) {
            throw new ResourceNotFoundException(ResultMessages.error().add(
                    MessageKeys.E_SL_PR_5002, token));
        }

        if (dateFactory.newTimestamp().toLocalDateTime()
                .isAfter(info.getExpiryDate())) { // (2)
            throw new BusinessException(ResultMessages.error().add(
                    MessageKeys.E_SL_PR_2001));
        }

        // omitted (attempts exceeded upper bounds)

        return info;
    }

    // omitted

}
```

Sr. No.	Description
(1)	Fetch authentication information for password reissue from the database by considering the token assigned as an argument, as the key.
(2)	After the expiry date, throw `org.terasoluna.gfw.common.exception.BusinessException`.

- Implementation of Controller

```
package org.terasoluna.securelogin.app.passwordreissue;

// omitted

@Controller
@RequestMapping("/reissue")
public class PasswordReissueController {

    @Inject
    PasswordReissueService passwordReissueService;

    // omitted

    public String showPasswordResetForm(PasswordResetForm form, Model␣
↪model,
            @RequestParam("token") String token) { // (1)

        PasswordReissueInfo info = passwordReissueService.findOne(token); /
↪/ (3)

        form.setUsername(info.getUsername());
        form.setToken(token);
        model.addAttribute("passwordResetForm", form);
        return "passwordreissue/passwordResetForm";
    }

    // omitted

}
```

Sr. No.	Description
(1)	Fetch the token included as a request parameter in the URL for password reissue screen.
(2)	Call the Service method by passing the token in it. Authentication information is fetched from the database and expiry date is checked.

- User verification using authentication information for password reissue

  The implementation of the process to confirm whether the set of confidential information entered by the user on the password reissue screen and the token included in the URL of the password reissue screen is correct, is shown below. This confirmation process is a password reissue-specific logic. Since it is a check in which the results vary depending on the contents of the database, it is implemented in the Service without using Bean Validation and Spring Validator.

  - Implementation of Service

```
package org.terasoluna.securelogin.domain.service.passwordreissue;

// omitted

public interface PasswordReissueService {

    // omitted

    boolean resetPassword(String username, String token, String secret, //␣
↪(1)
            String rawPassword);

    // omitted

}
```

Sr. No.	Description
(1)	A method to set the new password after user verification using user name, token and confidential information assigned as arguments

```
package org.terasoluna.securelogin.domain.service.passwordreissue;

// omitted

@Service
@Transactional
```

```
public class PasswordReissueServiceImpl implements PasswordReissueService {

    @Inject
    PasswordReissueFailureSharedService␣
↪passwordReissueFailureSharedService;

    @Inject
    PasswordReissueInfoRepository passwordReissueInfoRepository;

    @Inject
    AccountSharedService accountSharedService;

    @Inject
    PasswordEncoder passwordEncoder;

    // omitted

    @Override
    public boolean resetPassword(String username, String token, String␣
↪secret,
            String rawPassword) {
        PasswordReissueInfo info = this.findOne(token); // (1)
        if (!passwordEncoder.matches(secret, info.getSecret())) { // (2)
            passwordReissueFailureSharedService.resetFailure(username,␣
↪token);
            throw new BusinessException(ResultMessages.error().add(
                MessageKeys.E_SL_PR_5003));
        }
        failedPasswordReissueRepository.deleteByToken(token);
        passwordReissueInfoRepository.delete(token); // (3)

        return accountSharedService.updatePassword(username, rawPassword);␣
↪// (4)

    }

    // omitted

}
```

Sr. No.	Description
(1)	Using the token assigned as an argument, fetch the authentication information for password reissue from the database. At this time, the expiry date is checked again.
(2)	Compare the hashed confidential information included in the authentication information for password reissue with the confidential information given as an argument. If they vary, throw `BusinessException` . Password reissue fails in this case.
(3)	Delete used authentication information from the database in order to prevent it from being reused.
(4)	Update the account password that has user name that was passed as an argument to the specified new password.

- Implementation of Form

Since input check other than Null check is covered depending on the annotation assigned to the class, only `@NotNull` is assigned as a single item check.

```
package org.terasoluna.securelogin.app.passwordreissue;

// omitted

@Data
@Compare(source = "newPasssword", destination = "confirmNewPassword",␣
↪operator = Compare.Operator.EQUAL)
@StrongPassword(usernamePropertyName = "username", newPasswordPropertyName␣
↪= "newPassword") // (1)
@NotReusedPassword(usernamePropertyName = "username",␣
↪newPasswordPropertyName = "newPassword") // (2)
public class PasswordResetForm implements Serializable{

    private static final long serialVersionUID = 1L;

    @NotNull
    private String username;

    @NotNull
    private String token;
```

```
    @NotNull
    private String secret;

    @NotNull
    private String newPassword;

    @NotNull
    private String confirmNewPassword;
}
```

Sr. No.	Description
(1)	An annotation to check the password strength. Refer to *Check password strength* for details.
(2)	An annotation to check reuse of password. Refer to *Check password strength* for details.

- Implementation of View

Password reissue screen(passwordResetForm.jsp)

```
<body>
    <div id="wrapper">
        <h1>Reset Password</h1>
        <t:messagesPanel />
        <form:form
            action="${f:h(pageContext.request.contextPath)}/reissue/
↪resetpassword"
            method="POST" modelAttribute="passwordResetForm">
            <table>
                <tr>
                    <th><form:label path="username">Username</form:label></
↪th>
                    <td>${f:h(passwordResetForm.username)} <form:hidden
                            path="username" value="${f:h(passwordResetForm.
↪username)}" />  <!-- (1) -->
                    </td>
                    <td></td>
                </tr>
                <form:hidden path="token" value="${f:h(passwordResetForm.
↪token)}" /> <!-- (2) -->
                <tr>
                    <th><form:label path="secret" cssErrorClass="error-
↪label">Secret</form:label>
                    </th>
                    <td><form:password path="secret" cssErrorClass="error-
↪input" /></td> <!-- (3) -->
```

```
                    <td><form:errors path="secret" cssClass="error-messages
↪" /></td>
                </tr>
                <tr>
                    <th><form:label path="newPassword" cssErrorClass=
↪"error-label">New password</form:label>
                    </th>
                    <td><form:password path="newPassword"
                            cssErrorClass="error-input" /></td>
                    <td><form:errors path="newPassword" cssClass="error-
↪messages"
                            htmlEscape="false" /></td>
                </tr>
                <tr>
                    <th><form:label path="confirmNewPassword"
                            cssErrorClass="error-label">New␣
↪password(Confirm)</form:label></th>
                    <td><form:password path="confirmNewPassword"
                            cssErrorClass="error-input" /></td>
                    <td><form:errors path="confirmNewPassword"
                            cssClass="error-messages" /></td>
                </tr>
            </table>

            <input id="submit" type="submit" value="Reset password" />
        </form:form>
    </div>
</body>
```

Sr. No.	Description
(1)	Retain user name as a hidden item.
(2)	Retain token as a hidden item.
(3)	Prompt the user to enter confidential information for user verification.

Password reissue screen(passwordResetComplete.jsp)

```
<body>
    <div id="wrapper">
        <h1>Your password was successfully reset.</h1>
        <a href="${f:h(pageContext.request.contextPath)}/">go to Top</a>
    </div>
```

```
</body>
```

- Implementation of Controller

```
package org.terasoluna.securelogin.app.passwordreissue;

// omitted

@Controller
@RequestMapping("/reissue")
public class PasswordReissueController {

    @Inject
    PasswordReissueService passwordReissueService;

    // omitted

    @RequestMapping(value = "resetpassword", method = RequestMethod.POST)
    public String resetPassword(@Validated PasswordResetForm form,
            BindingResult bindingResult, Model model) {
        if (bindingResult.hasErrors()) {
            return showPasswordResetForm(form, model, form.getToken());
        }

        try {
            passwordReissueService.resetPassword(form.getUsername(),
                    form.getToken(), form.getSecret(), form.
↪getNewPassword()); // (1)
            return "redirect:/reissue/resetpassword?complete";
        } catch (BusinessException e) {
            model.addAttribute(e.getResultMessages());
            return showPasswordResetForm(form, model, form.getToken());
        }
    }

    @RequestMapping(value = "resetpassword", params = "complete", method =␣
↪RequestMethod.GET)
    public String resetPasswordComplete() {
        return "passwordreissue/passwordResetComplete";
    }

    // omitted

}
```

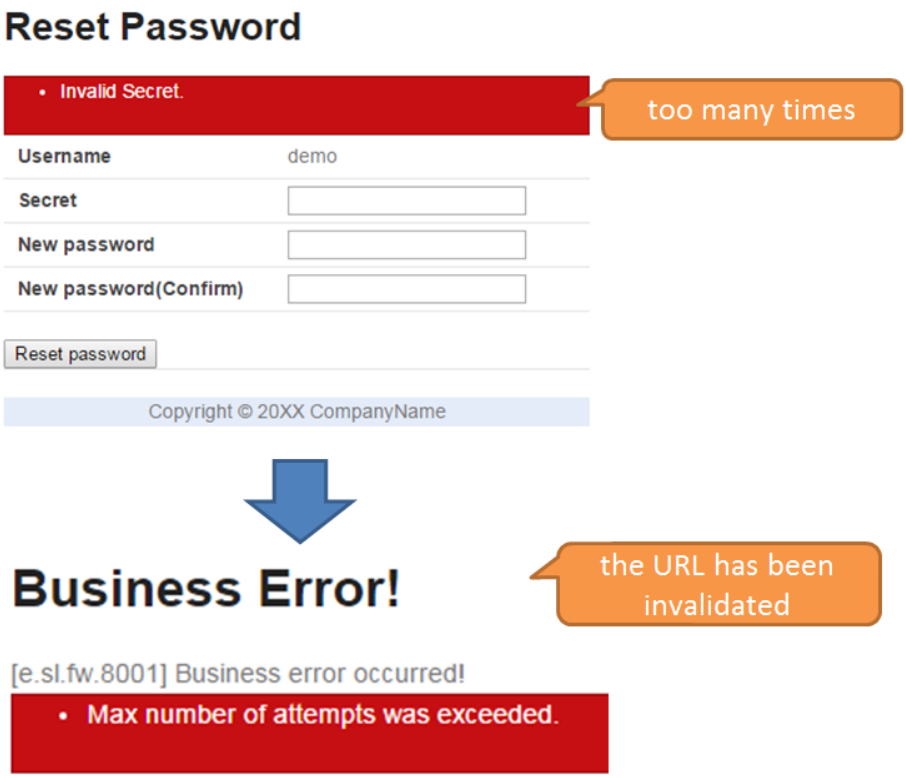
Sr. No.	Description
(1)	Pass the user name, token, confidential information and new password to the method of Service. If the combination of user name, token and confidential information is correct, it is updated to the new password.

Setting failure limit for the password reissue

List of requirements to be implemented

- *Setting failure limit for password reissue*

Working image



Even if URL for password reissue screen is leaked for some reason, password will not be reissued illegally if the confidential information is not leaked. Since a random value which cannot be guessed easily is used in the confidential information, it is highly unlikely to break the information easily. However, a maximum limit is set for the number of authentication failures to prevent brute force attack. When the authentication failures for password reissue exceeds the maximum limit, the password reissue for that URL (token) is disabled.

Implementation method

In this application, history of password reissue failure is stored in the database as "password reissue failure event" entity and number of failures for password reissue is measured by using the password reissue failure event entity.

When the number of failures is greater than the maximum value set in advance, an exception is thrown when the user tries to access password reissue screen.

In particular, fulfil the requirements by implementing and using following two processes.

- Storing password reissue failure event entity

  When a failure occurs in the user authentication, during the process "Confirmation of user using authentication information for password reissue" in *Validation at the time of executing password reissue*, set of the token used and failure date and time are registered in the database as password reissue failure event entity.

- Throwing an exception at the time of password reissue

  When authentication information is fetched from the database for password reissue, number of password reissue failure event entities is measured and an exception is thrown if the number is more than the maximum limit.

> **Warning:** Since password reissue failure event entity is only intended for measuring number of failures for password reissue, it is deleted when it is no longer required. When the log at the time of password reissue failure is required, a separate log must always be maintained.

Code description

- Common parts

  As a prerequisite, each process mentioned in *Validation at the time of executing password reissue* should be implemented. Other commonly required implementations related to registration, search and deletion of password reissue failure event entity for the database are shown below.

  - Implementation of Entity

    Implementation of password reissue failure event entity is shown below.

```
package org.terasoluna.securelogin.domain.model;

// omitted

@Data
public class FailedPasswordReissue {

    private String token; // (1)
```

```
    private LocalDateTime attemptDate; // (2)

}
```

Sr. No.	Description
(1)	A token used for password reissue
(2)	Date and time when password reissue is attempted

- Implementation of Repository

  Repository for search, registration and deletion of Entity is shown below.

```
package org.terasoluna.securelogin.domain.repository.passwordreissue;

// omitted

public interface FailedPasswordReissueRepository {

    int countByToken(@Param("token") String token); // (1)

    int create(FailedPasswordReissue event); // (2)

    int deleteByToken(@Param("token") String token); // (3)

    // omitted

}
```

Sr. No.	Description
(1)	A method which fetches number of `FailedPasswordReissue` objects considering token assigned as an argument, as a key.
(2)	A method which registers `FailedPasswordReissue` object assigned as an argument, as the records of the database.
(3)	A method which deletes `FailedPasswordReissue` object considering token assigned as an argument, as a key.

Mapping file is as given below.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<mapper
 namespace="org.terasoluna.securelogin.domain.repository.passwordreissue.
↪FailedPasswordReissueRepository">

 <select id="countByToken" resultType="_int">
    <![CDATA[
        SELECT
            COUNT(*)
        FROM
            failed_password_reissue
        WHERE
            token = #{token}
    ]]>
 </select>

 <insert id="create" parameterType="FailedPasswordReissue">
    <![CDATA[
        INSERT INTO failed_password_reissue (
            token,
            attempt_date
        ) VALUES (
         #{token},
            #{attemptDate}
        )
    ]]>
 </insert>
```

```
    <delete id="deleteByToken">
        <![CDATA[
            DELETE FROM
                    failed_password_reissue
            WHERE
                    token = #{token}
        ]]>
    </delete>

</mapper>
```

Code implemented in accordance with the implementation method is described here sequentially.

- Storing password reissue failure event entity

  A class which implements the process to be carried out at the time of password reissue failure is shown below.

```
package org.terasoluna.securelogin.domain.service.passwordreissue;

public interface PasswordReissueFailureSharedService {

    void resetFailure(String username, String token);

}
```

```
package org.terasoluna.securelogin.domain.service.passwordreissue;

// omitted

@Service
@Transactional
public class PasswordReissueFailureSharedServiceImpl implements
        PasswordReissueFailureSharedService {

    @Inject
    ClassicDateFactory dateFactory;

    @Inject
    FailedPasswordReissueRepository failedPasswordReissueRepository;

    // omitted

    @Transactional(propagation = Propagation.REQUIRES_NEW) // (1)
    @Override
    public void resetFailure(String username, String token) {
        FailedPasswordReissue event = new FailedPasswordReissue(); // (2)
        event.setToken(token);
        event.setAttemptDate(dateFactory.newTimestamp().toLocalDateTime());
```

```
            failedPasswordReissueRepository.create(event); // (3)
        }

}
```

Sr. No.	Description
(1)	It is a method which is called when a failure occurs during password reissue and has been designed to generate a run-time exception in the call source. Therefore, specify a propagation method in "REQUIRES_NEW" in order to perform transaction management separately from that of call source service.
(2)	Create password reissue failure event entity and specify token and, failure date and time.
(3)	Register password reissue failure event entity created in (2), in database.

Call process at the time of password reissue failure from "Confirmation of user using authentication information for password reissue" process of *Validation at the time of executing password reissue*.

```
package org.terasoluna.securelogin.domain.service.passwordreissue;

// omitted

@Service
@Transactional
public class PasswordReissueServiceImpl implements PasswordReissueService {

    @Inject
    PasswordReissueFailureSharedService passwordReissueFailureSharedService;

    @Inject
    PasswordReissueInfoRepository passwordReissueInfoRepository;

    @Inject
    AccountSharedService accountSharedService;

    @Inject
    PasswordEncoder passwordEncoder;

    // omitted

    @Override
    public boolean resetPassword(String username, String token, String secret,
```

```
            String rawPassword) {
        PasswordReissueInfo info = this.findOne(token); // (1)
        if (!passwordEncoder.matches(secret, info.getSecret())) { // (2)
            passwordReissueFailureSharedService.resetFailure(username, token);
↪ // (3)
            throw new BusinessException(ResultMessages.error().add(  // (4)
                MessageKeys.E_SL_PR_5003));
        }

        //omitted

    }

    // omitted

}
```

Sr. No.	Description
(1)	Fetch authentication information for password reissue from the database, using token assigned as an argument.
(2)	Compare hashed confidential information included in the authentication information for password reissue and confidential information assigned as an argument.
(3)	Call a method of SharedService which performs a process at the time of password reissue failure.
(4)	A run-time exception is thrown, however since the process at the time of password reissue failure is performed in different transaction, it does not leave any impact.

- Throwing an exception at the time of password reissue

  Fetching number of failures at the time of password reissue and implementation of process when the number of failures reach the maximum limit are shown below.

```
package org.terasoluna.securelogin.domain.service.passwordreissue;

// omitted

@Service
```

```
@Transactional
public class PasswordReissueServiceImpl implements PasswordReissueService {

    @Inject
    FailedPasswordReissueRepository failedPasswordReissueRepository;

    @Inject
    PasswordReissueInfoRepository passwordReissueInfoRepository;

    @Value("${security.tokenValidityThreshold}")
    int tokenValidityThreshold; // (1)

    // omitted

    @Override
    @Transactional(readOnly = true)
    public PasswordReissueInfo findOne(String token) {

        // omitted

        int count = failedPasswordReissueRepository // (2)
                .countByToken(token);
        if (count >= tokenValidityThreshold) { // (3)
            throw new BusinessException(ResultMessages.error().add(
                    MessageKeys.E_SL_PR_5004));
        }

        return info;
    }

    // omitted

}
```

Sr. No.	Description
(1)	Fetch and set maximum value for number of failures at the time of password reissue, from property file.
(2)	Fetch number of password reissue failure event entities from the database, considering token assigned as an argument, as a key.
(3)	Compare number of failure event entities at the time of password reissue that has been fetched and maximum limit for number of failures, and throw an exception if it exceeds the maximum limit.

Input check validation for security

List of requirements to be implemented

- *Configuration of common prohibited characters for request parameter*
- *Configuration of common prohibited strings for uploaded file name*
- *Input check for control characters*
- *Input check for file extensions*
- *Input check for file names*
- *Input check for URL domain*
- *Input check for mail address domain*

Working image

- Configuration of common prohibited strings
- Individual input check

Implementation method

Since the scope for performing a check greatly vary for the configuration of common prohibited characters of overall application and individual input checks, implementation is carried out separately.


Create Account
Username
ab&cd
prohibited character
Image
参照...
penguin.jpg
go to error page
Invalid Character Error!
Prohibited character detected!


Username
test user
First name
test
Last name
user
E-mail
testuser@test.org
E-mail(Confirmation)
testuser@test.org
URL
http://testuser.org
Image
参照...
めも.txt
Profile
test
control character
invalid domain
invalid file extension
invalid file name
Username
test user
Control characters are not allowed.
First name
test
Last name
user
E-mail
testuser@test.org
This domain is not allowed.
E-mail(Confirmation)
testuser@test.org
URL
http://testuser.org
This domain is not allowed.
Image
参照...
ファイルが選択されていません。
The file name is not allowed.
The file extension is not allowed.
Profile
test
show messages

For configuration of common prohibited strings, two methods described in *Implementing common logic to be executed before and after calling controller* can be used. In this application, implementation is carried out by using Servlet Filter in order to perform check regardless of whether mapping is done in the handler method of Controller. When an input error occurs, it signifies that the input values which were not predicted as the results of normal user operation are input and are described in the configuration file so as to transit to a common error screen without considering reduction in usability.

For individual input check, *Input Validation* function can be used. Input value check is performed in this application using Bean Validation. For the individual input error, implementation is done so as to encourage re-input by displaying an error message for input items corresponding to Bean Validation.

Code description

Code implemented in accordance with the implementation method above is explained sequentially.

- Implementation of Servlet Filter

  When the user input is to be used in the application, it may serve as a target for injection attacks like SQL injection and XSS directory traversal (path traversal).

  An implementation example of Servlet Filter to validate the input from the user in overall application is shown as a countermeasure for these attacks.

  In this application, it is verified that prohibited characters respectively configured for following items are not included.

Sr. No.	Items	Description
(1)	Request parameters	Since the request parameters are generally used for receiving the input from the user, the parameters act as a target for input check. Since both the parameter name and the value are entered by user, both the name and the value are checked.
(2)	Upload file name	Since file upload function is executed in this application, the file name of uploaded file entered by user acts as a target for input check.

  Further, Since `org.springframework.web.multipart.MultipartRequest` is used in the following code, using

`org.springframework.web.multipart.support.MultipartFilter` is assumed. For `MultipartFilter` , refer *File Upload*.

```
package org.terasoluna.securelogin.app.common.filter;

// omitted

public class InputValidationFilter extends OncePerRequestFilter { // (1)

    private final List<Character> prohibitedChars;

    private final List<Character> prohibitedCharsForFileName;

    public InputValidationFilter(char[] prohibitedChars,
            char[] prohibitedCharsForFileName) {
        this.prohibitedChars = Chars.asList(prohibitedChars); // (2)
        this.prohibitedCharsForFileName = Chars
                .asList(prohibitedCharsForFileName); // (3)
    }

    @Override
    protected void doFilterInternal(HttpServletRequest request,
            HttpServletResponse response, FilterChain filterChain)
            throws ServletException, IOException {
        if (request != null) {
            validateRequestParams(request); // (4)

            if (request instanceof MultipartRequest) {
                validateFileNames((MultipartRequest) request); // (5)
            }
        }

        filterChain.doFilter(request, response); // (6)
    }

    private void validateRequestParams(HttpServletRequest request) {
        Map<String, String[]> params = request.getParameterMap();
        for (Map.Entry<String, String[]> entry : params.entrySet()) {
            validate(entry.getKey(), prohibitedChars); // (7)
            for (String value : entry.getValue()) {
                validate(value, prohibitedChars); // (8)
            }
        }
    }

    private void validateFileNames(MultipartRequest request) {
        for (Map.Entry<String, MultipartFile> entry : request.getFileMap()
                .entrySet()) {
            String filename = new File(entry.getValue().getOriginalFilename())
```

```
                    .getName(); // (9)
            validate(filename, prohibitedCharsForFileName); // (10)
        }
    }

    private void validate(String target, List<Character> prohibited) {
        if (StringUtils.hasLength(target)) {
            List<Character> chars = Chars.asList(target.toCharArray());
            for(Character prohibitedChar : prohibited) { // (11)
                if (chars.contains(prohibitedChar)) {
                    throw new InvalidCharacterException(
                        "The request contains prohibited charcter.");
                }
            }
        }
    }

}
```

Sr. No.	Description
(1)	It is guaranteed that the process is executed only once for the request by inheriting `org.springframework.web.filter.OncePerRequestFilter`.
(2)	Receive the list of prohibited characters of request parameters as a string and store as a list of characters.
(3)	Receive the list of prohibited characters of file name as a string and store as a list of characters.
(4)	Call a method to perform input check of request parameters
(5)	In case of file upload request (when Content-Type is a POST request of `multipart/form-data`), the `request` object here acts as an instance of `StandardMultipartHttpServletRequest` which implements `MultipartRequest` , using `MultipartFilter` function. Fetch file name by using `MultipartRequest` API and call a method to perform input check.

次のページに続く

Table. 9.44 – 前のページからの続き

Sr. No.	Description
(6)	After completing input check, call a method to execute subsequent servlet filter process.
(7), (8)	Fetch the list of request parameters from `HttpServletRequest` and call a method to perform actual input check for each request parameter name and request parameter value.
(9)	Fetch the list of files uploaded from `MultipartRequest` and fetch actual file name. Since path is included in the file name according to browser or OS of the client and path delimiter tends to vary, the process is required to fetch only file name.
(10)	Call a method to perform actual input check for the file name of each uploaded file.
(11)	Check whether the string for input check is included in the prohibited characters - sequentially one character at a time and throw an exception if the prohibited character is included `InvalidCharacterException` is an exception created by inheriting `RuntimeException` . Code is omitted.

Tip: In this application, only request parameters and file names are checked for common prohibited characters. If required, similar check can also be implemented for HTTP header or cookies by fetching the value of HTTP header and Cookie using `getHeaders` and `getCookies` of `HttpServletRequest` .

- Configuration of Servlet Filter

Configure in web.xml to validate created `InputValidationFilter` .

Configuration is done by using `org.springframework.web.filter.DelegatingFilterProxy` by reading prohibited characters from the property file and defining a Bean for `InputValidationFilter` .

web.xml

```
<filter>
    <filter-name>MultipartFilter</filter-name>
    <filter-class>org.springframework.web.multipart.support.MultipartFilter</
↪filter-class>  <!-- (1) -->
</filter>
<filter-mapping>
    <filter-name>MultipartFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>

<filter>
    <filter-name>inputValidationFilter</filter-name>
    <filter-class>org.springframework.web.filter.DelegatingFilterProxy</
↪filter-class>  <!-- (2) -->
</filter>
<filter-mapping>
    <filter-name>inputValidationFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>

<!-- omitted -->

<error-page>
    <exception-type>org.terasoluna.securelogin.app.common.filter.exception.
↪InvalidCharacterException</exception-type>  <!-- (3) -->
    <location>/WEB-INF/views/common/error/invalidCharacterError.jsp</location>
</error-page>
```

Sr. No.	Description
(1)	Configure `MultipartFilter` which is a prerequisite for using `InputValidationFilter`. Note that `MultipartFilter` must be defined even prior to `InputValidationFilter`
(2)	Configure `InputValidationFilter` for which a Bean is defined, using `DelegatingFilterProxy`. Bean name must be specified in the `<filter-name>`.
(3)	Configure an error screen to be displayed when the `InvalidCharacterException` exception is thrown.

Note: Since `MultipartFilter` is used for input check of file name, configuration to validate upload function of Servlet 3.0 described in *Application settings* is required in addition to the details described here.

invalidCharacterError.jsp

```
<% response.setStatus(HttpServletResponse.SC_BAD_REQUEST); %>  <!-- (1) -->
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>Invalid Character Error!</title>

<!-- omitted -->
```

Sr. No.	Description
(1)	Since `InvalidCharacterException` is an exception which originates in the client input, set HTTP status code to `"400"` (Bad Request).

applicationContext.xml

```
<bean id="inputValidationFilter" class="org.terasoluna.securelogin.app.common.
↪filter.InputValidationFilter">
    <constructor-arg index="0" value="${app.security.prohibitedChars}"/>  <!--
↪ (1) -->
    <constructor-arg index="1" value="${app.security.
↪prohibitedCharsForFileName}"/>  <!-- (2) -->
```

```
</bean>
```

Sr. No.	Description
(1)	Fetch the list of prohibited characters of request parameter from property, as a string
(2)	Fetch the list of prohibited characters of file name from property, as a string

application.properties

```
## (1)
app.security.prohibitedChars=&\\!"<>*

## (2)
app.security.prohibitedCharsForFileName=&\\!"<>*;:
```

Sr. No.	Description
(1)	In this application, specify a list of characters which are not intended to be included in the request parameters, as a string
(2)	In this application, specify a list of characters which are not intended to be included in the uploaded file name, as a string

- Creating Bean Validation annotation

  An annotation of Bean Validation which validates the input value based on the requirements is created to alleviate the security risk posed by the input value which was not considered in the specifications of application.

  - Annotation to verify that the control characters are not included

    When the control characters are included in the input value, an unexpected issue is likely to occur in the application. Hence, it is checked that the control characters are not included for the input items which do not require input of control characters.

    Since only linefeed character of control characters is allowed in some cases at the time of text input, an annotation to allow linefeed code is created separately.

    It can be verified that control characters are not included by performing a check using normal

expressions.

```
package org.terasoluna.securelogin.app.common.validation;

// omitted
@Documented
@Constraint(validatedBy = {})
@Target({ FIELD })
@Retention(RUNTIME)
@ReportAsSingleViolation  // (1)
@Pattern(regexp = "^\\P{Cntrl}*$") // (2)
public @interface NotContainControlChars {
    String message() default "{org.terasoluna.securelogin.app.common.
↪validation.NotContainControlChars.message}";

    Class<?>[] groups() default {};

    @Target({ FIELD })
    @Retention(RUNTIME)
    @Documented
    public @interface List {
        NotContainControlChars[] value();
    }

    Class<? extends Payload>[] payload() default {};
}
```

Sr. No.	Description
(1)	Assign `@ReportAsSingleViolation` annotation to output only the message specified in the annotation, as an error message.
(2)	Assign `@Pattern` annotation to perform input check using normal expressions Since `\P{Cntrl}` signifies “characters other than control characters” in the normal expressions of Java, `^\\P{Cntrl}*$` matches only with the string which does not include a control character from beginning to end

Similarly, implementation example for the annotation which allows linefeed code is shown below.

```
package org.terasoluna.securelogin.app.common.validation;

// omitted
@Documented
@Constraint(validatedBy = {})
@Target({ FIELD })
@Retention(RUNTIME)
@ReportAsSingleViolation
@Pattern(regexp = "^[\\r\\n\\P{Cntrl}]*$") // (1)
public @interface NotContainControlCharsExceptNewlines {
    String message() default "{org.terasoluna.securelogin.app.common.
↪validation.NotContainControlCharsExceptNewlines.message}";

    Class<?>[] groups() default {};

    @Target({ FIELD })
    @Retention(RUNTIME)
    @Documented
    public @interface List {
        NotContainControlCharsExceptNewlines[] value();
    }

    Class<? extends Payload>[] payload() default {};
}
```

Sr. No.	Description
(1)	Specify normal expressions to match the string consisting of only “characters other than control character (`\P{Cntrl}`)” and linefeed code (`\r` , `\n`).

– Annotation to verify that extensions of the uploaded file are allowed

When a file uploaded by the user is to be received, the format of the file to be received is likely to be restricted. In such cases, a list of extensions for the files allowed is set and it is checked whether the extension of the uploaded file is included in the list.

It is a specification which can be changed for whether the upper case and lower case extensions are to be differentiated.

Warning: Since extension of the file can be disguised easily, a file format should not be trusted unconditionally even after performing the extension check.

```
package org.terasoluna.securelogin.app.common.validation;

// omitted

@Documented
@Constraint(validatedBy = { FileExtensionValidator.class })
@Target({ ElementType.FIELD })
@Retention(RetentionPolicy.RUNTIME)
public @interface FileExtension {
    String message() default "{org.terasoluna.securelogin.app.common.
↪validation.FileExtension.message}";

    Class<?>[] groups() default {};

    Class<? extends Payload>[] payload() default {};

    String[] extensions();  // (1)

    boolean ignoreCase() default true;  // (2)

    @Target({ ElementType.FIELD })
    @Retention(RetentionPolicy.RUNTIME)
    @Documented
    public @interface List {
        FileExtension[] value();
    }
}
```

Sr. No.	Description
(1)	Specify list of extensions to be allowed.
(2)	Whether to ignore the differentiation between uppercase and lowercase characters. Default value is `true` (ignore)

```
package org.terasoluna.securelogin.app.common.validation;

// omitted

public class FileExtensionValidator implements
        ConstraintValidator<FileExtension, MultipartFile> {

    private Set<String> extensions;

    private boolean ignoreCase;
```

```
    @Override
    public void initialize(FileExtension constraintAnnotation) {
        this.extensions = new HashSet<String>(
                Arrays.asList(constraintAnnotation.extensions()));
        this.ignoreCase = constraintAnnotation.ignoreCase();
    }

    @Override
    public boolean isValid(MultipartFile value,
            ConstraintValidatorContext context) {
        if (value == null) {  // (1)
            return true;
        }

        String fileNameExtension = StringUtils.getFilenameExtension(value
                .getOriginalFilename());  // (2)
        if (!StringUtils.hasLength(fileNameExtension)) {  // (3)
            return false;
        }

        for (String extension : extensions) {  // (4)
            if (fileNameExtension.equals(extension) || ignoreCase
                    && fileNameExtension.equalsIgnoreCase(extension)) {
                return true;
            }
        }
        return false;
    }

}
```

Sr. No.	Description
(1)	Return `true` in case of `null` since `null` check is performed by using another annotation.
(2)	Fetch extension from the file name by using `getFilenameExtension` method of `org.springframework.util.StringUtils`
(3)	Return `false` for the file without an extension since it is not permissible.
(4)	Fetch extensions one by one from the list of extensions to be allowed and compare with the string fetched from the file name. Return `true` if even one of the extensions show a match and return `false` if it does not match with any of the extensions in the list.

- Annotation that verifies that file name of the uploaded file matches with the patterns that are allowed

  When a file uploaded by the user is to be received, the format of the file to be received is likely to be restricted. In such cases, pattern for the file name to be allowed is set in normal expression and it is checked that file name of uploaded file matches with the pattern.

```
package org.terasoluna.securelogin.app.common.validation;

// omitted

@Documented
@Constraint(validatedBy = { FileNamePatternValidator.class })
@Target({ ElementType.FIELD })
@Retention(RetentionPolicy.RUNTIME)
public @interface FileNamePattern {

    String message() default "{org.terasoluna.securelogin.app.common.
↪validation.FileNamePattern.message}";

    Class<?>[] groups() default {};

    Class<? extends Payload>[] payload() default {};

    String pattern() default "";  // (1)

    @Target({ ElementType.FIELD })
    @Retention(RetentionPolicy.RUNTIME)
```

```
    @Documented
    public @interface List {
        FileNamePattern[] value();
    }

}
```

Sr. No.	Description
(1)	Pattern for the file name to be allowed

```
package org.terasoluna.securelogin.app.common.validation;

// omitted

public class FileNamePatternValidator implements
        ConstraintValidator<FileNamePattern, MultipartFile> {

    private Pattern pattern;

    @Override
    public void initialize(FileNamePattern constraintAnnotation) {
        this.pattern = Pattern.compile(constraintAnnotation.pattern());  //
 (1)
    }

    @Override
    public boolean isValid(MultipartFile value,
            ConstraintValidatorContext context) {
        if (value == null) {  // (2)
            return true;
        }

        String filename = new File(value.getOriginalFilename()).getName();
 // (3)
        return pattern.matcher(filename).matches();  // (4)
    }

}
```

Sr. No.	Description
(1)	Create and retain a pattern of normal expressions to be checked, as a `Pattern` .
(2)	Return `true` in case of `null` since `null` check is performed by using another annotation.
(3)	Fetch actual file name from `MultipartRequest` . Since path is included in the file name in accordance with browser and OS of client and path delimiter varies, the process is required for fetching only file name.
(4)	Return `true` when `Pattern` created in (1) and file name match otherwise return `false` .

- Annotation to verify that domain of input URL is allowed

When a URL input by the user is to be received, the domain that are allowed are likely to be restricted. In such cases, list of domains allowed is set and it is checked whether the domain of input URL is a domain included in the list or is a subdomain.

Since URL format is checked at the same time, implementation is done combining with `org.hibernate.validator.constraints.URL` .

```
package org.terasoluna.securelogin.app.common.validation;

// omitted

@Documented
@Constraint(validatedBy = { DomainRestrictedURLValidator.class })
@Target({ FIELD })
@Retention(RUNTIME)
@URL  // (1)
public @interface DomainRestrictedURL {

    String message() default "{org.terasoluna.securelogin.app.common.
↪validation.DomainRestrictedURL.message}";

    Class<?>[] groups() default {};

    String[] allowedDomains() default {};  // (2)
```

```
    @Target({ FIELD })
    @Retention(RUNTIME)
    @Documented
    public @interface List {
        DomainRestrictedURL[] value();
    }

    Class<? extends Payload>[] payload() default {};

}
```

Sr. No.	Description
(1)	Assign `@URL` to check URL format.
(2)	List of domains to be allowed

```
package org.terasoluna.securelogin.app.common.validation;

// omitted

public class DomainRestrictedURLValidator implements
        ConstraintValidator<DomainRestrictedURL, String> {

    private static final Pattern URL_REGEX = Pattern  // (1)
        .compile( "(?i)^(?:[a-z](?:[-a-z0-9\\+\\.])*)" + // protocol
                  ":(?:\\/\\/([^\\/:]+)" + // auth+host/ip
                  "(?::([0-9]*))?" + // port
                  "(?:\\/.*)*)$"
         );

    private Set<String> allowedDomains;

    @Override
    public void initialize(DomainRestrictedURL constraintAnnotation) {
        allowedDomains = new HashSet<String>(Arrays.
↪asList(constraintAnnotation
                .allowedDomains()));  // (2)
    }

    @Override
    public boolean isValid(String value, ConstraintValidatorContext␣
↪context) {
        Matcher urlMatcher = URL_REGEX.matcher(value);
        if (urlMatcher.matches()) {  // (3)
```

```
            String host = urlMatcher.group(1);
            for(String domain : allowedDomains) {  // (4)
                if (StringUtils.hasLength(host) && host.endsWith(".
↪"+domain)) {
                    return true;
                }
            }
            return false;
        } else {
            return true;
        }
    }

}
```

Sr. No.	Description
(1)	Create and retain a pattern of normal expressions as a `Pattern` for fetching URL domain Since whether the URL format is correct is verified by `@URL` , only minimum required expressions are specified here.
(2)	Fetch and retain the list of domains to be allowed
(3)	Check whether the URL format is correct. If the format is invalid, return `true` here since a validation error occurs in `URL` annotation used in combination
(4)	Fetch the domain one by one from the list of domains to be allowed and check whether it matches with the end part of host name of URL. Return `true` if even one of the values match and return `false` if it does not match with any of the values

- Annotation to verify that domain of input mail address is allowed

When a mail address input by the user is to be received, the domains that are allowed are likely to be restricted. In such cases, list of allowed domains is set and it is checked that domain of input mail address is included in the list. It is a specification which can be changed to include whether to allow the subdomain of domain.

Since the mail address format is checked at the same time, implementation is done combining with

`org.hibernate.validator.constraints.Email`.

```
package org.terasoluna.securelogin.app.common.validation;

// omitted

@Documented
@Constraint(validatedBy = { DomainRestrictedEmailValidator.class })
@Target({ FIELD })
@Retention(RUNTIME)
@Email  // (1)
public @interface DomainRestrictedEmail {
    String message() default "{org.terasoluna.securelogin.app.common.
↪validation.DomainRestrictedEmail.message}";

    Class<?>[] groups() default {};

    String[] allowedDomains() default {};  // (2)

    boolean allowSubDomain() default false;  // (3)

    @Target({ FIELD })
    @Retention(RUNTIME)
    @Documented
    public @interface List {
        DomainRestrictedEmail[] value();
    }

    Class<? extends Payload>[] payload() default {};
}
```

Sr. No.	Description
(1)	Assign `@Email` to check the mail address format
(2)	List of domains to be allowed
(3)	Whether to allow a subdomain. Default value is `false` (do not allow)

```
package org.terasoluna.securelogin.app.common.validation;

// omitted
```

```
public class DomainRestrictedEmailValidator implements
        ConstraintValidator<DomainRestrictedEmail, CharSequence> {

    private Set<String> allowedDomains;

    private boolean allowSubDomain;

    @Override
    public void initialize(DomainRestrictedEmail constraintAnnotation) {
        allowedDomains = new HashSet<String>(Arrays.
↪asList(constraintAnnotation
                .allowedDomains()));  // (1)
        allowSubDomain = constraintAnnotation.allowSubDomain();  // (2)
    }

    @Override
    public boolean isValid(CharSequence value,
            ConstraintValidatorContext context) {
        if (value == null) {  // (3)
            return true;
        }

        for (String domain : allowedDomains) {  // (4)
            if (value.toString().endsWith("@" + domain)
                    || (allowSubDomain && value.toString().endsWith(
                            "." + domain))) {
                return true;
            }
        }
        return false;
    }

}
```

Sr. No.	Description
(1)	Fetch and retain the list of domains to be allowed.
(2)	Fetch and retain the truth value which indicates whether the subdomain is allowed
(3)	Return `true` in case of `null` since `null` check is performed by another annotation
(4)	Fetch domain one by one from the list of domains to be allowed and check whether it matches with the domain part of the mail address. Return `true` even if one of the domains show a match and return `false` of it does not match any of the domains.

- Assigning annotation to Form class

  Annotation created is assigned to the field of Form class of new account creation.

```
package org.terasoluna.securelogin.app.account;

// omitted

public class AccountCreateForm implements Serializable {

    // omitted

    @NotNull
    @NotContainControlChars  // (1)
    @Size(min=4, max=128)
    private String username;

    // omitted

    @NotNull
    @NotContainControlChars
    @Size(min=1, max=128)
    @DomainRestrictedEmail(allowedDomains={ "domainexample.co.jp",
             "somedomainexample.co.jp" }, allowSubDomain=true)  // (2)
    private String email;

    // omitted

    @NotNull
    @NotContainControlChars
```

```
    @DomainRestrictedURL(allowedDomains={ "jp" })  // (3)
    private String url;

    @UploadFileRequired
    @UploadFileNotEmpty
    @UploadFileMaxSize
    @FileExtension(extensions = { "jpg", "png", "gif" })  // (4)
    @FileNamePattern(pattern = "[a-zA-Z0-9_-]+\\.[a-zA-Z]{3}")  // (5)
    private MultipartFile image;

    @NotNull
    @NotContainControlCharsExceptNewlines  // (6)
    private String profile;

}
```

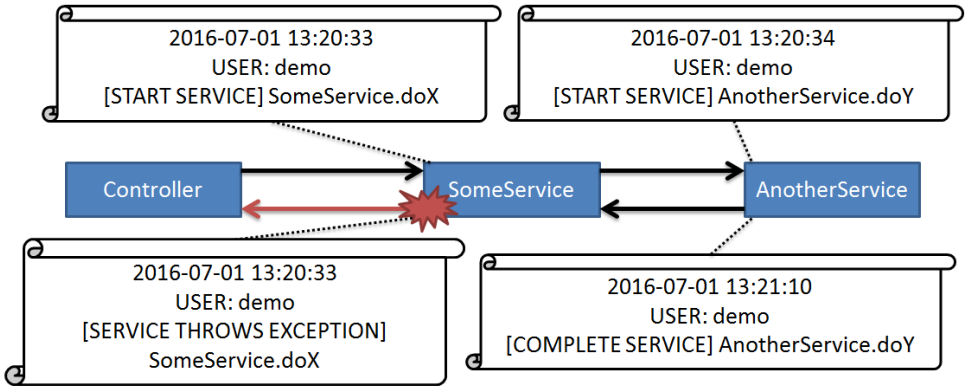
Sr. No.	Description
(1)	Check that control characters are not included.
(2)	Check that the mail address domain is “ntt.co.jp”, “nttdata.co.jp” or its subdomain
(3)	Check that URL domain is “jp” or its subdomain
(4)	Check that file extemsions are one of the “jpg”, “png” and “gif” extensions
(5)	Check that the file name is of “repeating 1 or more characters of Alphanumeric, “_”, “-” ” + ” ”.” ” + “single byte alphabet 3 characters” pattern
(6)	Check that control characters other than linefeed character are not included

Audit log output

List of requirements to be implemented

- *Audit log output*

Working image



While calling a service class method, date and time of calling, name of the user calling and method name are output in a log for the audit in order to check the information regarding the application like the user who has performed the operation, date and time at which the operation is performed, what kind of operation is performed etc. Further, for the results of method execution, log is output as "operation successful" if exception does not occur during the operation and "operation failed" if exception occurs during the process.

Log format is as below..

`{date and time of method calling} {Thread} {Name of user calling} {X-Track} {Log level} {Logger} {Message}`

Sr. No.	Description
Date and time of method calling	Date and time at which service class method is called is output in "yyyy-MM-dd HH:mm:ss" format
Thread	A thread which outputs the log
Name of user calling	Output user name of Spring Security which calls service class method Kept blank in case of no login
X-Track	An ID which is set for each request to enhance traceability For details, refer *Logging*
Log level	Level of log which has been output Output in `info` level in this application
Logger	A logger which outputs a log
Message	Time at which a method is called: "[START SERVICE] ServiceClassName.methodName" Time at which method is successfully terminated: "[COMPLETE SERVICE] ServiceClassName.methodName" Exception occurrence time: "[SERVICE THROWS EXCEPTION] ExceptionClassName.methodName"

How to implement

User name is fetched from authentication information of Spring Security in order to output user name in the log. Fetching user name and log output are implemented by using `org.terasoluna.gfw.security.web.logging.UserIdMDCPutFilter` described in *Logging* .

Further, operation details and operation results for the request are defined as below and output in a log, for this application.

- Operation details: Name of the method of service class that has been called
- Operation results: Whether the exceptions have occurred in the results of method processing

AOP(Aspect Oriented Programming) function offered by Spring can be used to achieve a cross-sectional function like log output for calling of all the methods of service class

AOP offered by Spring offers many implementation methods. This application puts an emphasis on combining with implementation of logging related components offered by common library <https://github.com/terasolunaorg/terasoluna-gfw>`_ and adopts a method to implement `org.aopalliance.intercept.MethodInterceptor`.

Requirements are fulfilled by implementation and configuration given below.

- Configure `UserIdMDCPutFilter`
- Create an advice which outputs a log at the time of calling a method and after execution
- Configure to apply the advice defined above for the class assigned with `@Service`

Note: Advice indicates a process which is executed for AOP within specified timing. Further, the place where advice can be embedded is called a joining point whereas set of join points where advice is embedded is called as a point cut For AOP function offered by Spring, refer Official document - AOP <http://docs.spring.io/spring/docs/4.3.5.RELEASE/spring-framework-reference/html/aop.html>`_ .

Code description

Code implemented in accordance with the implementation method above is sequentially explained.

- Configure `UserIdMDCPutFilter`

  Configuration to output authenticated user name of Spring Security in log is shown below.

  spring-security.xml

```
<!-- omitted -->

<sec:http>
    <!-- omitted -->
    <sec:custom-filter ref="userIdMDCPutFilter" after="ANONYMOUS_FILTER" />
    <!-- omitted -->
```

```
</sec:http>

<!-- omitted -->
<bean id="userIdMDCPutFilter" class="org.terasoluna.gfw.security.web.logging.
↪UserIdMDCPutFilter">
</bean>
```

Sr. No.	Description
(1)	Configure `UserIdMDCPutFilter` in Filter Chain of Spring Security to output a log immediately after generating user information By configuring `UserIdMDCPutFilter` , authenticated user name is added to MDC with a key called `USER` .

logback.xml

```
<!-- omitted -->

<appender name="AUDIT_LOG_FILE"
    class="ch.qos.logback.core.rolling.RollingFileAppender">  <!-- (1) -->
    <file>log/security-audit.log</file>
    <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
        <fileNamePattern>log/security-audit-%d{yyyyMMdd}.log</fileNamePattern>
        <maxHistory>7</maxHistory>
    </rollingPolicy>
    <encoder>
        <charset>UTF-8</charset>
        <pattern><![CDATA[date:%d{yyyy-MM-dd HH:mm:ss}\tthread:%thread\tUSER:
↪%X{USER}\tX-Track:%X{X-Track}\tlevel:%-5level\tlogger:%-48logger{48}
↪\tmessage:%msg%n]]></pattern>  <!-- (2) -->
    </encoder>
</appender>

<!-- omitted -->

<logger
   name="org.terasoluna.securelogin.domain.common.interceptor.
↪ServiceCallLoggingInterceptor"
   additivity="false">  <!-- (3) -->
   <level value="info" />
   <appender-ref ref="AUDIT_LOG_FILE" />
</logger>

<!-- omitted -->
```

Sr. No.	Description
(1)	Define an appender for audit log output.
(2)	Describe “USER:%X{USER}” in the pattern definition
(3)	Define a logger for audit log output Implementation of `org.terasoluna.securelogin.domain.common.interceptor.ServiceCallLoggingIn` will be explained later.

- Create an advice which outputs a log at the time of calling a method and after execution

  Implementation and configuration to output a log for operation details and operation results are shown below.

```
package org.terasoluna.securelogin.domain.common.interceptor;

// omitted

public class ServiceCallLoggingInterceptor implements MethodInterceptor {  //
↪ (1)

  private static final Logger logger = LoggerFactory
          .getLogger(ServiceCallLoggingInterceptor.class);

  @Override
  public Object invoke(MethodInvocation invocation) throws Throwable {  //␣
↪ (2)
      String methodName = invocation.getMethod().getName();
      String className = invocation.getMethod().getDeclaringClass()
              .getSimpleName();
      logger.info("[START SERVICE]{}.{}", className, methodName);  // (3)
      try {
          Object result = invocation.proceed();  // (4)
          logger.info("[COMPLETE SERVICE]{}.{}", className, methodName);  //
 (5)
          return result;  // (6)
      } catch (Throwable e) {
          logger.info("[SERVICE THROWS EXCEPTION]{}.{}", className,  // (7)
                  methodName);
          logger.info(Exception : {}, Message : {}, e.getClass().getName(),
                  e.getMessage()); // (8)
          throw e;  // (9)
```

```
        }
    }
}
```

Sr. No.	Description
(1)	Implement `MethodInterceptor` to describe the processes to be carried out before and after calling a method
(2)	Override `invoke` method defined in `MethodInterceptor` Information like name of the method being called can be fetched from `org.aopalliance.intercept.MethodInvocation` object of argument
(3)	Output name of the method to be called in a log before calling a method
(4)	Perform actual method calling and fetch results
(5)	Output a “operation successful” log message if an exception does not occur in the results of calling a method
(6)	Return a object for results of method calling
(7)	Output a “operation failed” message when an exception occurs in the results of calling a method
(8)	Output exception class and exception message thus occurred Since it is for the auditing, a stack trace is not output to avoid redundancy in the log
(9)	Throw an exception object thus occurred

Tip: Detailed contents can also be output by using arguments at the time of calling a method, to further extend the example of this application. However, in such a case, a password without hashing is likely to be output in a log. Hence, countermeasures like masking may be required.

- Configure for applying advice for a class assigned with `@Service` .

Configuration of point cut for the advice is shown below.

secure-login-domain.xml

```
<!-- omitted -->

<bean id="serviceCallLoggingInterceptor"
    class="org.terasoluna.securelogin.domain.common.interceptor.
↪ServiceCallLoggingInterceptor" />  <!-- (1) -->
<aop:config>
    <aop:advisor advice-ref="serviceCallLoggingInterceptor"
        pointcut="@within(org.springframework.stereotype.Service)" />  <!--␣
↪(2) -->
</aop:config>

<!-- omitted -->
```

Sr. No.	Description
(1)	Define a Bean for advice class created above (implementation class of `MethodInterceptor`)
(2)	Set Bean which implements the advice, in `advice-ref` attribute of `aop:advisor` tag and set point cuts in `pointcut` attribute respectively. By defining a point cut `@within(org.springframework.stereotype.Service)` , the method calling of the class assigned with `@Service` is used as a target for advice.

Note: Spring AOP adopts a proxy method wherein a proxy class which has been auto-created handles the method calling. Note that, as a constraint of AOP of Proxy method, advice is not executed for the method calling of visibility other than `public` or the method calling within the same class. For details, refer Official document .

Output results of log are as below.

```
date:2016-08-18 13:45:42   thread:tomcat-http--7   USER:demo   X-Track:
↪f514cc4159324ba28d8393f2c3062d89    level:INFO  logger:o.t.s.d.c.i.
↪ServiceCallLoggingInterceptor        message:[START␣
↪SERVICE]AccountSharedService.isInitialPassword
date:2016-08-18 13:45:42   thread:tomcat-http--7   USER:demo   X-Track:
↪f514cc4159324ba28d8393f2c3062d89    level:INFO  logger:o.t.s.d.c.i.
↪ServiceCallLoggingInterceptor        message:[START␣
↪SERVICE]PasswordHistorySharedService.findLatest
date:2016-08-18 13:45:42   thread:tomcat-http--7   USER:demo   X-Track:
↪f514cc4159324ba28d8393f2c3062d89    level:INFO  logger:o.t.s.d.c.i.
↪ServiceCallLoggingInterceptor        message:[COMPLETE␣
↪SERVICE]PasswordHistorySharedService.findLatest
date:2016-08-18 13:45:42   thread:tomcat-http--7   USER:demo   X-Track:
↪f514cc4159324ba28d8393f2c3062d89    level:INFO  logger:o.t.s.d.c.i.
↪ServiceCallLoggingInterceptor        message:[COMPLETE␣
↪SERVICE]AccountSharedService.isInitialPassword
```

Log at the time occurrence of exception is as below. Since the log is for the process performed before login, the user name is not displayed.

```
date:2016-08-18 13:52:32   thread:tomcat-http--10  USER:   X-Track:
↪1a37a9a280014216a300b61e2f4bbb66    level:INFO  logger:o.t.s.d.c.i.
↪ServiceCallLoggingInterceptor        message:[SERVICE THROWS␣
↪EXCEPTION]AccountSharedService.findOne
date:2016-08-18 13:52:32   thread:tomcat-http--10  USER:   X-Track:
↪1a37a9a280014216a300b61e2f4bbb66    level:INFO  logger:o.t.s.d.c.i.
↪ServiceCallLoggingInterceptor        message:[SERVICE THROWS␣
↪EXCEPTION]UserDetailsService.loadUserByUsername
date:2016-08-18 13:52:32   thread:tomcat-http--10  USER:   X-Track:
↪1a37a9a280014216a300b61e2f4bbb66    level:INFO  logger:o.t.s.d.c.i.
↪ServiceCallLoggingInterceptor        message:user not found
```

9.9.4 Conclusion

This chapter explains about an example of implementation method for security measures using a sample application.

Since it is likely that implementation method in this application cannot be used as it is in the actual development, a different customised method must be considered according to the requirements, using details of this chapter as a reference.

9.9.5 Appendix

Passay

Passay is a library which offers a password validation function and a password generation function. Passay API consists of following three key components.

- Validation rules

  It defines the conditions which must be fulfilled by password. Validation rules which are used widely like length of the password and characters types to be included can be created easily by using the class offered by library. Besides, the necessary validation rules can also be defined by the user.

- Validator

  A component which performs password validation based on validation rules. Various validation rules can be specified in a single validator.

- Generator

  A component which generates a password in conformance with the validation rules for the assigned character type.

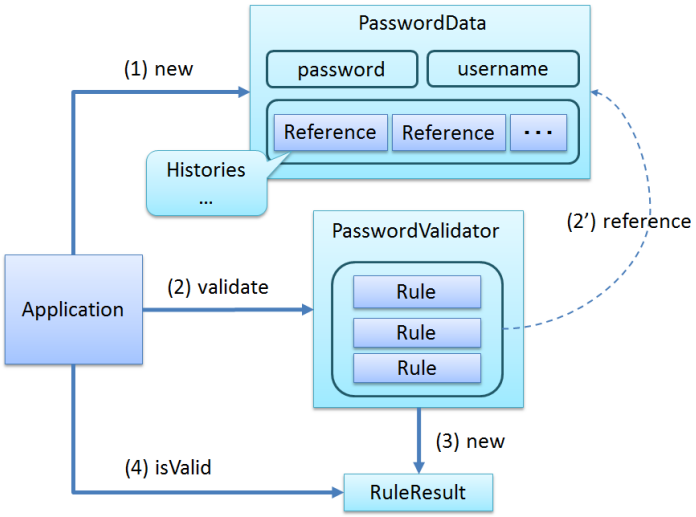
When Passay function is to be used, following definition must be added to pom.xml.

```
<dependencies>
  <dependency>
      <groupId>org.passay</groupId>
      <artifactId>passay</artifactId>
      <version>1.1.0</version>
  </dependency>
<dependencies>
```

Password validation

Overview

A schematic diagram of the password validation flow in Passay is shown below.



Sr. No.	Description
(1)	Create an instance of `org.passay.PasswordData` and specify information related to password to be validated. `PasswordData` can include list of passwords used in the past as a property in addition to password and user name. Passwords used in the past are retained as an instance of `org.passay.PasswordData.Reference` .
(2)	Perform validation for `PasswordData` using a validator, in accordance with the validation rules. Validation rules are created as an instance of implementation class of `org.passay.Rule` . A validator is an instance of `org.passay.PasswordValidator` and can include multiple validation rules as properties.
(3)	An instance of `org.passay.RuleResult` is created as validation result using a validator.

次のページに続く

Table. 9.45 – 前のページからの続き

Sr. No.	Description
(4)	Password validation results can be fetched from `RuleResult` as a `boolean` value. Also, an error message can be fetched from `RuleResult` by using a validator.

Some of the classes of validation rules offered by Passay are shown in the table below.

Class Name	Description	Key properties
`LengthRule`	A class of validation rules to specify minimum and maximum value for password length.	`minimuxLength` : Minimum value for password length (`int`). Specify in constructor or setter. `maximumLength` : Maximum value for password length (`int`)。 Specify in constructor or setter.
`CharacterRule`	A class of validation rules to specify character types that must be included in the password and the minimum number of characters of that character type.	`characterData`: Character type (`org.passay.CharacterData`). Specify in constructor. `numberOfCharacters` : Minimum number of characters (`int`). Specify in constructor or setter.
`CharacterCharacteristicsRule`	A class of validation rules which specify the number of rules that should be fulfilled, from multiple `CharacterRule` .	`rules`: List of validation rules related to character types (`List<CharacterRule>`). Specify in setter. `numberOfCharacteristics` : Minimum number of rules that should be fulfilled (`int`). Specify in setter.
`HistoryRule`	A class of validation rules which checks the pasword does not match with the password used previously.	None
`UsernameRule`	A class of validation rules to check that password should not contain the user name.	`matchBackwards` : Also check whether the user name has been used in the reverse (`boolean`). Specify in constructor or setter. `ignoreCase` : Not case-sensitive (`boolean`). Specify in constructor or setter.

In addition, classes of validation rules to check whether a specific character is to be included or to check using a regular expression are also provided. For details, refer http://www.passay.org/.

How to use

A validator can be created by passing the list of `org.passay.Rule` instance to the constructor of `PasswordValidator` . DI can be applied by defining a Bean for validator as below which specifies the validation rules. Note that, when a Bean is to be defined for multiple validation rules, DI must be applied using a Bean name, by combining `@Inject` and `@Named` .

```
<!-- Password Rules. -->
<bean id="upperCaseRule" class="org.passay.CharacterRule"> <!-- (1) -->
    <constructor-arg name="data">
        <util:constant static-field="org.passay.EnglishCharacterData.UpperCase" />
↪<!-- (2) -->
    </constructor-arg>
    <constructor-arg name="num" value="1" /> <!-- (3) -->
</bean>
<bean id="lowerCaseRule" class="org.passay.CharacterRule"> <!-- (4) -->
    <constructor-arg name="data">
        <util:constant static-field="org.passay.EnglishCharacterData.LowerCase" />
    </constructor-arg>
    <constructor-arg name="num" value="1" />
</bean>
<bean id="digitRule" class="org.passay.CharacterRule"> <!-- (5) -->
    <constructor-arg name="data">
        <util:constant static-field="org.passay.EnglishCharacterData.Digit" />
    </constructor-arg>
    <constructor-arg name="num" value="1" />
</bean>

<!-- Password Validator. -->
<bean id="characterPasswordValidator" class="org.passay.PasswordValidator"> <!--␣
↪(6) -->
    <constructor-arg name="rules">
        <list>
            <ref bean="upperCaseRule" />
            <ref bean="lowerCaseRule" />
            <ref bean="digitRule" />
        </list>
    </constructor-arg>
</bean>
```

Sr. No.	Description
(1)	Define a Bean for validation rules for specifying character type that must be included in the password and minimum number of characters for the character type.
(2)	Specify character type. Since `org.passay.EnglishCharacterData.UpperCase` is passed, validation rules are set for single byte uppercase characters.
(3)	Specify number of characters. Since “1” is passed, validation rules are set to check whether one or more single byte uppercase characters are included.
(4)	It is similar to (1)-(3), however since `org.passay.EnglishCharacterData.UpperCase` is passed as a character type, a bean is defined for the validation rules to check whether one or more single byte lowercase letters are included.
(5)	It is similar to (1)-(3), however, since `org.passay.EnglishCharacterData.Digit` is passed as a character type, a bean is defined for the validation rules to check whether one or more single byte digit is included.
(6)	Define a Bean for validator. Pass a list of validation rules to the constructor.

Perform password validation by using created validator.

```
@Inject
PasswordValidator characterPasswordValidator;

// omitted
```

```
public void validatePassword(String password){

    PasswordData pd = new PasswordData(password); // (1)
    RuleResult result = characterPasswordValidator.validate(pd); // (2)
    if (result.isValid()) { // (3)
       logger.info("Password is valid");
    } else {
       logger.error("Invalid password:");
       for (String msg : characterPasswordValidator.getMessages(result)) { // (4)
           logger.error(msg);
       }
    }

}
```

Sr. No.	Description
(1)	Pass the password to be validated to the constructor of `PasswordData` and create an instance.
(2)	Pass `PasswordData` in the `validate` method of `PasswordValidator` as an argument and implement password validation.
(3)	Fetch password validation results in truth value by using `isValid` method of `RuleResult` .
(4)	Pass `RuleResult` in `getMessages` method of `PasswordValidator` as an argument and fetch error message.

Password generation

Overview

Password generator and generation rules are used in the password generation function for Passay. A generator is an instance of `org.passay.PasswordGenerator` and the generation rules is a list of validation rules related to character type (`org.passay.CharacterRule`).

By assigning the length of the password to be generated and generation rules to the method of generator as arguments, a password which fulfils the generation rules is generated.

How to use

A method of creation of validation rules related to character type which is included in the generation rules is similar to *Password validation*. DI can be applied by defining a bean for generation rules and generator as given below.

```
<!-- Password Rules. -->
<bean id="upperCaseRule" class="org.passay.CharacterRule"> <!-- (1) -->
    <constructor-arg name="data">
        <util:constant static-field="org.passay.EnglishCharacterData.UpperCase" />
↪<!-- (2) -->
    </constructor-arg>
    <constructor-arg name="num" value="1" /> <!-- (3) -->
</bean>
<bean id="lowerCaseRule" class="org.passay.CharacterRule"> <!-- (4) -->
    <constructor-arg name="data">
        <util:constant static-field="org.passay.EnglishCharacterData.LowerCase" />
    </constructor-arg>
    <constructor-arg name="num" value="1" />
</bean>
<bean id="digitRule" class="org.passay.CharacterRule"> <!-- (5) -->
    <constructor-arg name="data">
        <util:constant static-field="org.passay.EnglishCharacterData.Digit" />
    </constructor-arg>
    <constructor-arg name="num" value="1" />
</bean>

 <!-- Password Generator. -->
 <bean id="passwordGenerator" class="org.passay.PasswordGenerator" /> <!-- (6) -->
 <util:list id="passwordGenerationRules"> <!-- (7) -->
     <ref bean="upperCaseRule" />
     <ref bean="lowerCaseRule" />
     <ref bean="digitRule" />
 </util:list>
```

Sr. No.	Description
(1)	Define a Bean for validation rules for specifying the character type that should be included in the password, and minimum number of characters for the character type.
(2)	Specify character type. Since `org.passay.EnglishCharacterData.UpperCase` is passed, validation rules are set for single byte uppercase characters.
(3)	Specify number of characters. Since "1" is passed, validation rules are set to check whether one or more single byte uppercase characters are included.
(4)	It is similar to (1)-(3), however since `org.passay.EnglishCharacterData.UpperCase` is passed as a character type, a bean is defined for the validation rules to check whether one or more single byte lowercase letters are included.
(5)	It is similar to (1)-(3), however, since `org.passay.EnglishCharacterData.Digit` is passed as a character type, a bean is defined for the validation rules to check whether one or more single byte digit is included.
(6)	Define a bean for generator.
(7)	Define a bean for generation rules. It is defined as a list of validation rules related to character type, defined in (1)-(5).

Create a password by using created generator and generation rules.

```
@Inject
PasswordGenerator passwordGenerator;

@Resource(name = "passwordGenerationRules")
List<CharacterRule> passwordGenerationRules;

// omitted
```

```
public void generatePassword(){

    String password = passwordGenerator.generatePassword(10,␣
↪passwordGenerationRules); // (1)

}
```

Sr. No.	Description
(1)	If length of the password to be generated and the generation rules are passed to `generatePassword` method of `PasswordGenerator` as arguments, a password which fulfils the generation rules is created.

Tip: When DI is to be applied to a collection for which a Bean is defined, an expected operation is not performed by `@Inject` + `@Named` . Therefore, DI is applied by Bean name using `@Resource` instead.

9.10 OAuth

9.10.1 Overview

This section explains the overview of OAuth 2.0 and the method to implement authorization control function as per the specifications of OAuth 2.0, by using Spring Security OAuth which is one of the Spring project.

Tip: ** Reference of Spring Security OAuth **

Spring Security OAuth also provides the functions which are not introduced in this guideline. Refer to OAuth 2 Developers Guide , for the details of Spring Security OAuth.

About OAuth 2.0

OAuth 2.0 is the authorization framework where access range can be specified for the resources protected on server, when HTTP service is used in third-party application.

OAuth 2.0 is specified as RFC, and it consists of various related technical specifications.

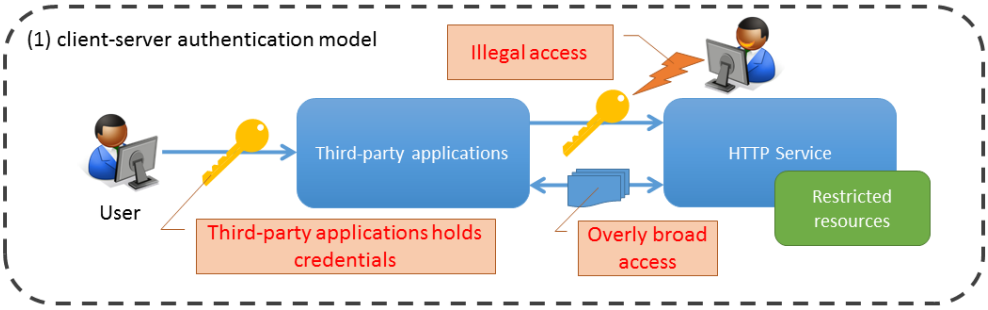
Important specifications of OAuth 2.0 are as follows.

Table.9.46 ** OAuth 2.0 important specifications **

RFC	Overview	Description
RFC 6749	The OAuth 2.0 Authorization Framework	Technical specifications describing the most fundamental contents of OAuth 2.0, such as terminology and authorization methods.
RFC 6750	Bearer Token Usage	Technical specifications related to method of transferring the “Access token without signature” (Hereafter, referred as access token) within servers, which is used for authorization control described in RFC 6749. Information about access token is described later.
RFC 6819	Threat Model and Security Considerations	Technical specifications about the security requirements that need to be considered while using OAuth 2.0 In these guidelines, the concrete explanation about the study items is omitted.
RFC 7519	JSON Web Token (JWT)	Technical specifications related to “JSON Web Token (JWT)” which is a token including JSON that can be signed.
RFC 7523	JSON Web Token (JWT) Profile for OAuth 2.0 Client Authentication and Authorization Grants	Technical specifications related to method of using JWT prescribed in RFC 6819 as an access token to be used for authorization control described in RFC 6749.
RFC 7009	OAuth 2.0 Token Revocation	Technical specifications related to additional end point invalidating the token.

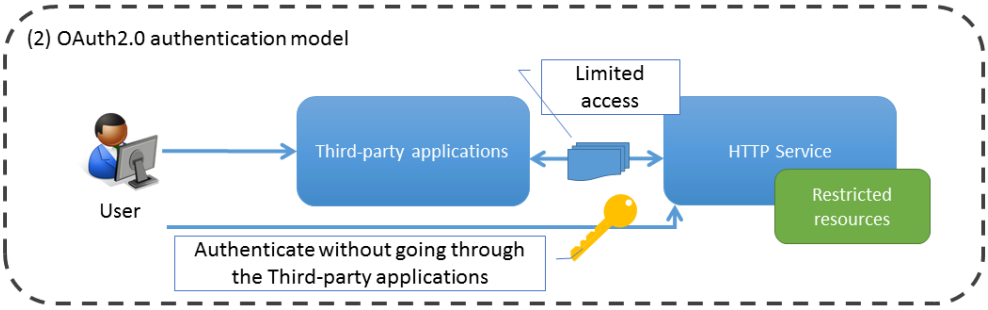
In conventional client-server type authentication model, the third-party application performs authentication by using user authentication information (user name and password etc.) in order to access the resources protected on server.

In other words, user needs to share the authentication information with third party in order to provide the rights to third party applications for accessing the resources; however it has the risks such as unintended access and information leakage by the user, if there are defects and malicious operations in third party application.



On the other hand, in OAuth 2.0, authentication can be directly performed by the user, and third party applications can access resources without sharing authentication information to third parties by issuing information for authenticated requests called as “access token”.

Moreover, more flexible access control is achieved as compared with conventional client-server type authentication model, by enabling the specification of access range (scope) of resources when access token is issued.



Architecture of OAuth 2.0

This section explains the roles, scope, authorization grant, and protocol flow defined in OAuth 2.0. In OAuth 2.0, concepts such as scope and authorization grant are defined, and the specifications of authorization are prescribed

by using these concepts.

Roles

Following 4 roles are defined in OAuth 2.0.

Table.9.47: ** Roles in OAuth 2.0 **

Role name	Description
Resource owner	Role permitting the access to protected resources. User (End user) etc.
Resource server	Role providing the protected resources. Web server.
Authorization server	Role authenticating resource owner and issuing the access token (Information necessary for client to access the resource server). Web server.

次のページに続く

Table. 9.47 – 前のページからの続き

Role name	Description
Client	Role getting the authorization of resource owner and making the request for protected resources on behalf of resource owner. Web application etc. Client information is registered in the authorization server in-advance and managed by client ID which is unique in the authorization server. OAuth 2.0 defines following 2 client types based on the ability to maintain confidentiality of client credentials (client authentication information). (1) Confidential Client that can maintain the confidentiality of client credential. (2) Public Client that cannot maintain the confidentiality of client credentials and cannot perform secure client authentication by using other means like the client executed on the device of resource owner. Moreover, OAuth 2.0 is being designed by considering the following examples as a client. (1) Web application (web application) Client executed on Web server (Confidential). (2) User agent-based application (user-agent-based application) Client executed in user agents of resource owner, by downloading the client code from Web server (public) . Such as Javascript application. (3) Native application (native application) Client installed and executed on the device of resource owner (Public).

Scope

The concept “Scope” is being used as a method of controlling access for resources protected in OAuth 2.0.

In response to the request from client, the authorization server can specify the access rights (read, write permissions etc.) for the protected resources, including the scope in access token based on the policy of authorization server or the instructions of resource owner.

Protocol flow

In OAuth 2.0, resources are accessed in the following flow.

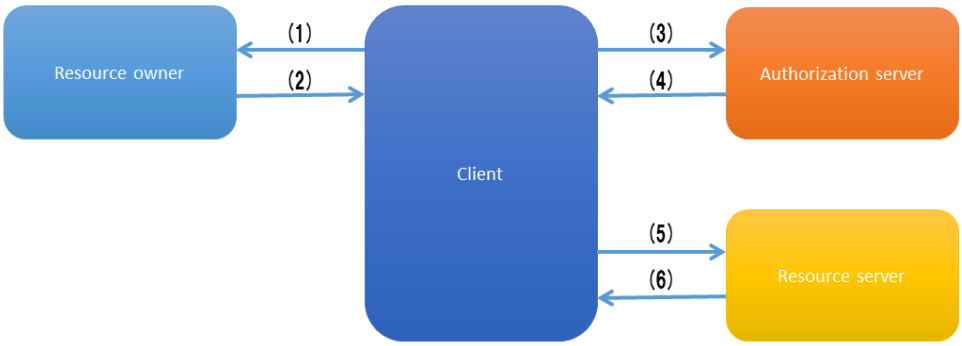


Table.9.48 ** Protocol flow of OAuth 2.0 **

Sr.No.	Description
(1)	Client requests authorization for the resource owner. As shown in the above figure, request is directly made to the resource owner, but it is recommended to request through the authorization server. Request is made through authorization server for the authorization code grant and implicit grant among the grant types described later.
(2)	Client receives an authorization grant (described later) as the credentials representing authorization from resource owner.
(3)	Client requests for access token by presenting own authentication information and authorization grant given by resource owner to the authorization server.
(4)	Authorization server authenticates the client and confirms the validity of authorization grant. It issues the access token if authorization grant is valid.
(5)	Client requests to the resource protected on the resource server and authenticates with the issued access token.
(6)	Resource server checks the validity of access token and accepts the request if it is valid.

Note: In order to simplify the complicated mechanism of signature and token exchange which was not popular in OAuth 1.0, the request handling access token needs to be made by HTTPS communication in OAuth 2.0. (Using

HTTPS communication prevents eavesdropping of access token)

Authorization grant

Authorization grant represents authorization from resource owner and it is used when client fetches the access token. In OAuth 2.0, following 4 grant types are defined, however these can be individually extended such as adding the credential items etc.

Client requests the access token to authorization server from any of grant types and accesses the resource server with the fetched access token. Authorization server must define 1 or more supporting grant types, and determine the grant type to be used among these as per the authorization request from client.

Table.9.49 **Authorization grant in OAuth 2.0**

Grant type	Description
Authorization code grant	In the flow of authorization code grant, the authorization server issues authorization code to client as an intermediary between client and resource owner, and the client issues the access token by passing the authorization code to authorization server. It is not necessary to share the credentials of resource owner to the client in order to issue the access token by using the authorization code issued by authorization server. Authorization code grant is used, when confidential client uses OAuth 2.0 similar to a Web application.
Implicit grant	In the flow of implicit grant, authorization server acts as an intermediary similar to authorization code grant, however it issues the access token directly instead of an authorization code. Since access token is encoded in the URL, it may get leaked to a resource owner or other applications on the same device. Further, since the client is not authenticated, a risk of spoofing attack due to unauthorized usage of access token issued to other clients is also likely. Implicit grant should be used only when client type is public, such as client implemented in Javascript.
Resource owner password credential grant	In the flow of resource owner password credential grant, client uses the authentication information of resource owner as authorization grant and issues the access token directly. Since the credentials of resource owner must be shared with client, a risk of unauthorized usage or leakage of credentials is likely if client's reliability is low. Resource owner password credential grant should be used only when there is high reliability between resource owner and client and when other grant types cannot be used.
Client credential grant	In the flow of client credential grant, authentication information of client is used as an authorization grant, and access token is issued directly. It is used when client is a resource owner.

Flow of authorization code grant is as follows.

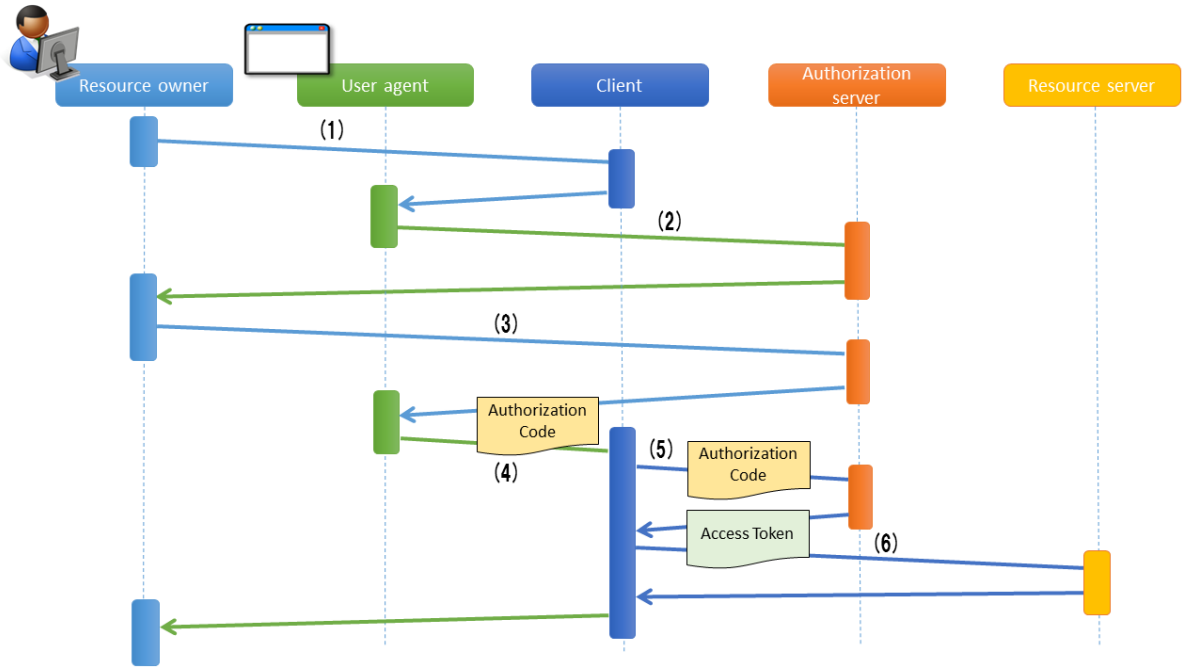


Table.9.50: **Authorization code grant flow**

Sr. No.	Description
(1)	Resource owner accesses the pages required for resources which are protected on resource server provided by the client through user agent. Client gives access to authorization end point on authorization server for the user agent of resource owner, in order to fetch authorization from resource owner. In this case, client includes the “Client ID for own identification”, “Scope requested to the resource as an option”, “Redirect URL for returning the user agent after authorization process performed by authorization server” and “State” in the request parameters. state is a random value associated with user agent and is used to ensure that series of flows is executed by the same user agent (CSRF countermeasures)

次のページに続く

Table. 9.50 – 前のページからの続き

Sr. No.	Description
(2)	User agent accesses the authorization end point of authorization server indicated to the client. Authorization server authenticates the resource owner through user agent and checks the validity of parameters by comparing it with the registered client information, based on the client ID, scope and redirect URL of request parameter. After confirmation, resource owner is asked to approve/ deny the access request.
(3)	Resource owner sends approval / denial of access request to the authorization server. When resource owner grants access, the authorization end point of authorization server issues authorization code and gives the instructions to redirect the user agent to client by using the redirect URL of request parameter. In such case, authorization code is assigned to redirect URL, as the request parameter of redirect URL.
(4)	User agent accesses the redirect URL with assigned authorization code. When processing of client is completed, the response is returned to resource owner.
(5)	Client sends the authorization code to the token end point on authorization server, in order to request the access token. Token end point of authorization server authenticates the client and verifies the validity of authorization code, and issues the access token and optional refresh token if it is valid. Refresh token is used to issue new access token when the access token is disabled or expired.
(6)	Client accesses the resource server with the fetched access token. Resource server checks the validity of access token, and if it is valid, it processes the request and returns the response to client.

Flow of implicit grant is as follows.

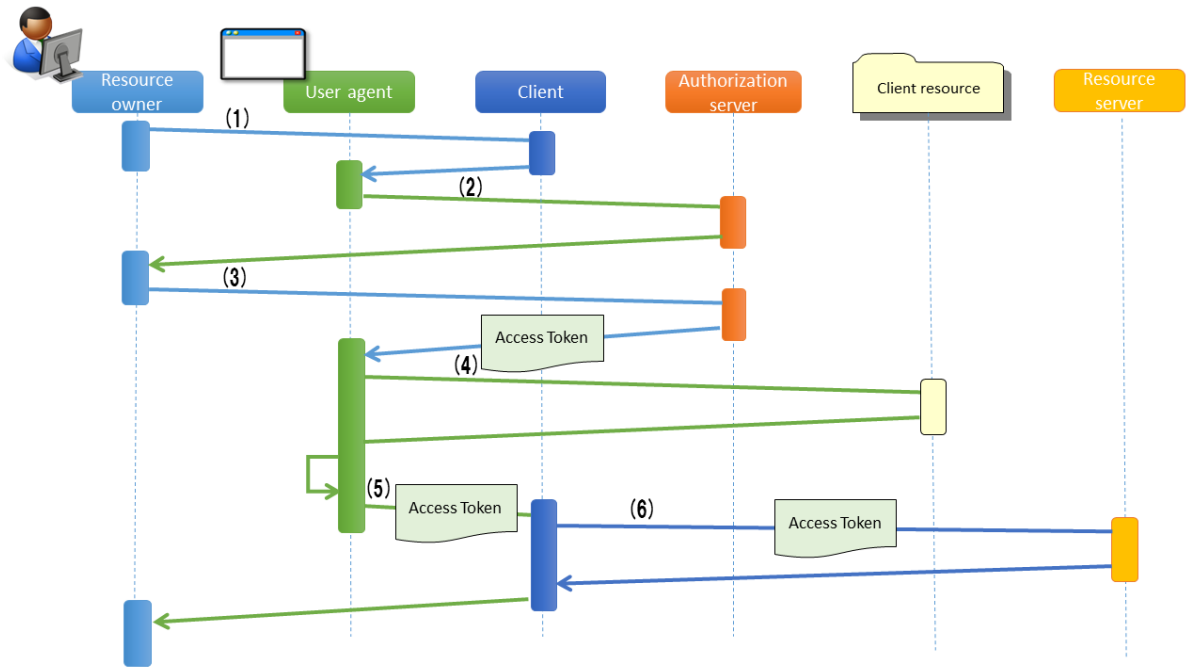


Table.9.51: **Flow of implicit grant**

Sr. No.	Description
(1)	Resource owner accesses the pages required for resources which are protected on resource server provided by the client through user agent. Client gives access to authorization end point on authorization server for the user agent of resource owner, in order to fetch authorization from resource owner and issue the access token. In this case, client includes the “Client ID for own identification”, “Scope requested to the resource as an option”, “Redirect URL for returning the user agent after performing authorization process by authorization server” and “State”, in the request parameters. state is a random value associated with user agent and is used to ensure that series of flows is executed by the same user agent (CSRF countermeasures)
(2)	User agent accesses the authorization end point of authorization server indicated to the client. Authorization server authenticates the resource owner through user agent and checks the validity of parameters by comparing it with the registered client information, based on the client ID, scope and redirect URL of request parameter. After confirmation, resource owner is asked to approve/ deny the access request.
	次のページに続く

Table. 9.51 – 前のページからの続き

Sr. No.	Description
(3)	Resource owner sends approval / denial of access request to the authorization server. When resource owner grants access, the authorization end point of authorization server gives the instructions to redirect the user agent to client resource by using the redirect URL of request parameter. In such case, access token is assigned to URL fragment of redirect URL.
(4)	User agent sends the request to client resource as per the redirect instructions. In such case, information of URL fragment is saved locally, and URL fragment is not sent in case of redirect. When client resource is accessed, the web page (usually, HTML document including the embedded script) is returned. User agent executes the script included in web page and extracts the access token from the locally saved URL fragment.
(5)	User agent passes the access token to client. When processing of client is completed, the response is returned to resource owner.
(6)	Client accesses the resource server with the fetched access token. Resource server checks the validity of access token, and if it is valid, it processes the request and returns the response to client

Flow of resource owner password credential grant is as follows.

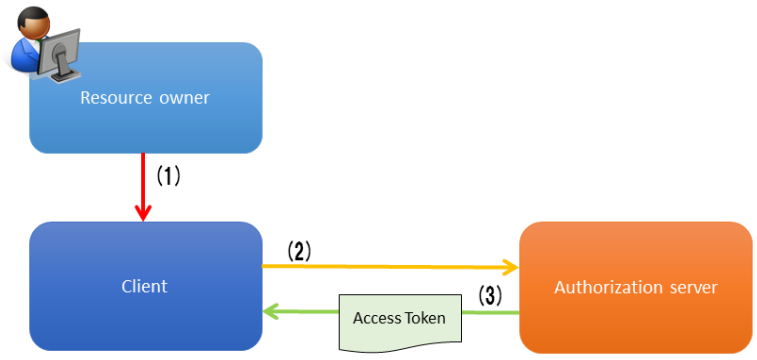


Table.9.52 **Flow of resource owner password credential grant**

Sr. No.	Description
(1)	Resource owner provides the credentials (user name, password) to the client.
(2)	Client accesses the token end point of authorization server, in order to request the access token. In this case, client includes the credentials specified from resource owner and the scope requested to resource, in the request parameter.
(3)	Token end point of authorization server authenticates the client and verifies the credentials of resource owner. If it is valid, it issues the access token.

Flow of client credential grant is as follows.

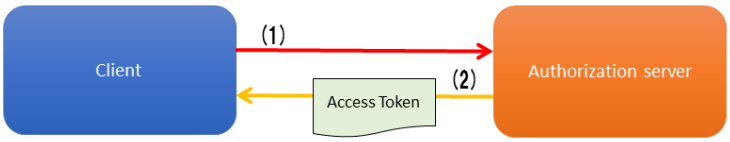


Table.9.53 **Flow of client credential grant**

Sr. No.	Description
(1)	Client accesses the token end point of authorization server, in order to request the access token. In this case, client requests for access token including the client's own credentials.
(2)	Token end point of authorization server authenticates the client and issues the access token when authentication is successful.

Life cycle of access token

Access token is issued by authorization server, by confirming the validity of authorization grant presented by the client. For an issued access token, the scope is assigned based on the policy of authority server or the instructions of resource owner and access is obtained for the protected resource. Expiry period is set when the access token is issued, and if it expires, the access rights of protected resource are invalidated.

Flow from issue of access token till its invalidation is as follows.

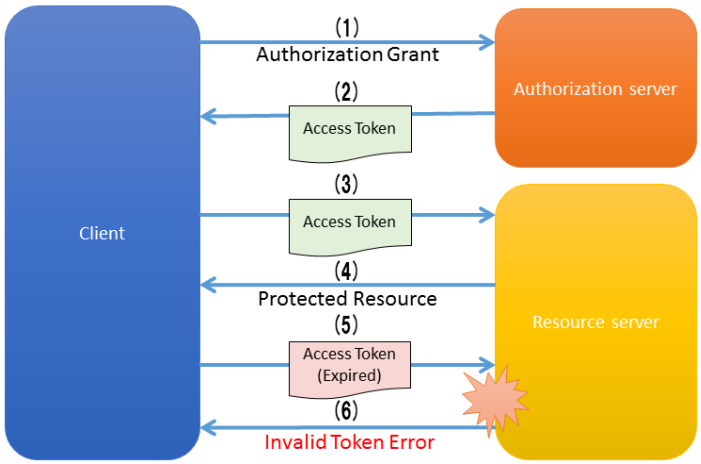


Table.9.54: **Flow from issue of access token till its invalidation**

Sr. No.	Description
(1)	Client presents the authorization grant and requests for access token.
(2)	Authorization server confirms the authorization grant presented by client and issues the access token.
(3)	Client presents the access token and requests for resource protected on resource server.
(4)	Resource server verifies the validity of access token presented by client and if it is valid, performs processing for the resource protected on resource server.

次のページに続く

Table. 9.54 – 前のページからの続き

Sr. No.	Description
(5)	Client presents the access token (expired), and requests for the resource protected on resource server.
(6)	Resource server verifies the validity of access token presented by client, and returns error if the access token has expired.

When access token expires, the access rights of protected resources are invalidated, but access rights of protected resources can also be invalidated by disabling the access token before expiry of access token.

When access token is to be disabled before its expiry, the client requests authorization server to cancel the token. Access rights of protected resources are invalidated for the disabled access token.

When access token expires, authorization grant should be presented to authorization server again and validity should be re-confirmed by authorization server, in order to re-acquire the access token by client. Therefore, when short validity term of access token is set, the usability decreases. On the other hand, when long validity term of access token is set, a high risk of disclosure or access token, and misuse is likely if it is disclosed.

Refresh token can be used to reduce the risk of disclosure without decreasing the usability of the token. Refresh token can be used to get the new access token without re-submitting the authorization grant when access token is invalidated or expired. When validity term is set at the time of issuing the refresh token and when refresh token expires, the access token cannot be re-issued.

Risk of disclosure of access token and misuse at the time of disclosure can also be controlled while maintaining the usability with the re-issue of access token in short cycle, by setting the short validity term of access token and setting the long validity term of refresh token.

Issuing the refresh token is optional and is to be determined by the authorization server.

Flow of re-issuing the access token as per the refresh token is as follows.

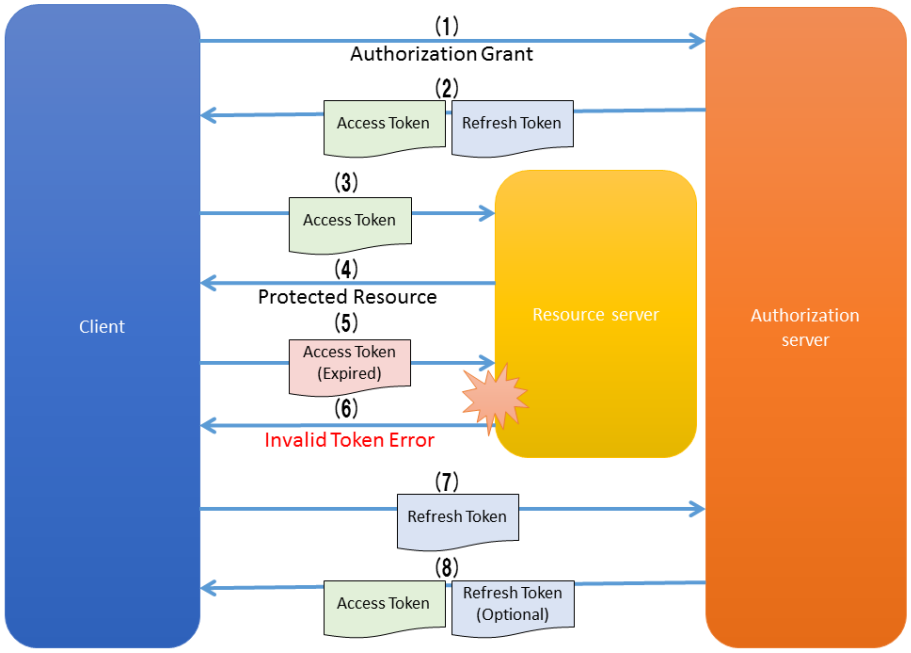


Table.9.55 **Flow from issuing the access token till its re-issue**

Sr. No.	Description
(1)	Client presents the authorization grant and requests for access token.
(2)	Authorization server confirms the authorization grant presented by client and issues the access token and refresh token.
(3)	Client presents the access token and requests for resource protected on resource server.
(4)	Resource server verifies the validity of access token presented by client, and if it is valid, it processes the resource protected on resource server.
(5)	Client presents the access token (expired), and requests for the resource protected on resource server.
(6)	Resource server verifies the validity of access token presented by client, and returns error if the access token has expired.
(7)	When access token expiration error is returned from resource server, the client requests a new access token by presenting the refresh token.

Architecture of Spring Security OAuth

Spring Security OAuth is a library that provides functions necessary while building 3 roles such as Authorization Server, Resource Server, and Client as Spring applications among the roles defined in OAuth 2.0. Spring Security OAuth is the technique that works by linking with the functions provided by Spring Framework (Spring MVC) and Spring Security, and it can build the authorization server, resource server and client by appropriate configuration (Bean definition) of default package provided by Spring Security OAuth. Further, many extension points are provided similar to Spring Framework and Spring Security in order to incorporate the requirements that cannot be implemented in default package provided by Spring Security OAuth.

Further, refer to *Authentication* and *Authorization* for the details, when the functions provided by Spring Security are to be used for authentication/ authorization of requests within each roles.

When authorization server, resource server, client are built by using Spring Security OAuth, process is performed with the flow given below.

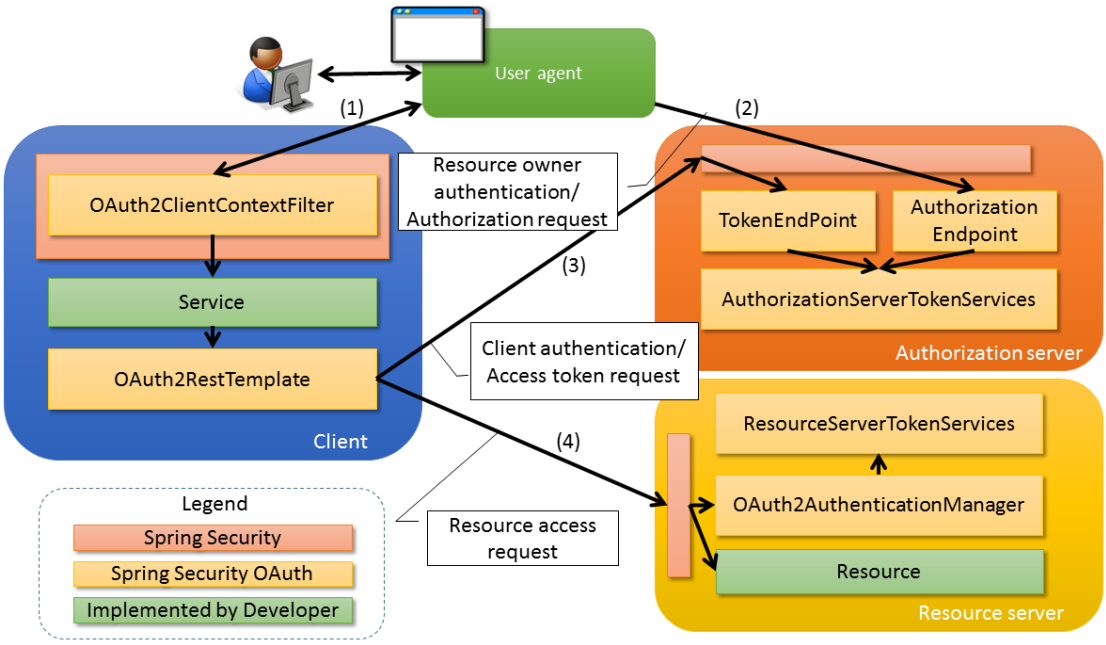


Table.9.56 **Flow of Spring Security OAuth**

Sr. No.	Description
(1)	Resource owner accesses the client through user agent. Client calls `OAuth2RestTemplate` from service. In case of authorization grant accessing the authorization end point, instructions are given to user agent to redirect to authorization end point of authorization server.
(2)	User agent accesses the `AuthorizationEndpoint` which is an authorization end point of authorization server. `AuthorizationEndpoint` displays the screen that demands the authorization to the resource owner. Resource owner accesses `AuthorizationEndpoint` by authorizing the scope for client. `AuthorizationEndpoint` issues the authorization code if authorization grant is authorization code grant and it issues the access token if it is the implicit grant. Issued authorization code or access token is passed to the client through user agent by using the redirect.
(3)	Client accesses `TokenEndpoint` which is the token end point of authorization server from `OAuth2RestTemplate`. `TokenEndpoint` calls `AuthorizationServerTokenService` and issues the access token. Issued access token is passed to client as a response. If authorization grant is authorization code grant, the authorization code is presented to authorization server. Validity of authorization code is verified by `TokenEndpoint` before issuing the access token.
(4)	Client specifies the access token fetched in (2) or (3) and accesses the resource server from `OAuth2RestTemplate`. Resource server calls the `OAuth2AuthenticationManager` and fetches the authentication information associated with access token through `ResourceServerTokenServices`. Further, it verifies the access token when authentication information is fetched. When access token verification is successful, the resource corresponding to request from the client is returned.

Note: As mentioned earlier, usage of HTTPS communication at each end point is assumed in OAuth 2.0, however there may be the cases when HTTPS communication is used in SSL accelerator or web server, or when distributed in multiple AP servers by using load balancer. When authorization code or redirect URL is embedded in client for linking the access token after authorization by resource owner, the redirect URL indicating the SSL accelerator, Web server and load balancer should be embedded.

Spring (Spring Security OAuth) provides the technique for assembling the redirect URL by using any of the following headers.

- Forwarded header
- X-Forwarded-Host header, X-Forwarded-Port header, X-Forwarded-Proto header

Setting should be done in order to assign the above mentioned headers at SSL accelerator and Web server, load balancer side, so that an appropriate redirect URL can be created in Spring(Spring Security OAuth). If this is not performed, verification of request parameter (redirect URL) performed in the authorization code grant or implicit grant is likely to fail.

Tip: End point provided by Spring Security OAuth is implemented by extending the Spring MVC function. `@FrameworkEndpoint` annotation is set in class at end point provided by Spring Security OAuth. This is because `@Controller` annotation does not conflict with the class registered by developer as a component. Further, end point registered in `@FrameworkEndpoint` annotation as component handles the `@FrameworkEndpoint` method conflicting with URL as Handler class, after reading `@RequestMapping` annotation of end point by `FrameworkEndpointHandlerMapping` which is an extension class of `RequestMappingHandlerMapping`.

Authorization server

Authorization server authenticates the resource owner, fetches the authorization from the resource owner after authentication, and then issues the access token.

OAuth 2.0 supports the "Scope" concept as an expression for specifying the access scope. Client specifies the scope while sending the authorization request, and when the resource owner authorizes the specified scope or when it matches with the scope registered previously in authorization server, that scope is authorized for the client by authorization server. Authorization can be used together in accordance with the scope of client and roles of Spring Security explained in *Authorization* section.

Spring Security OAuth provides the function to get an authorization from resource owner in `AuthorizationEndpoint`, and provides the function to issue access token of client in `AuthorizationEndpoint` or `TokenEndpoint`. `AuthorizationEndpoint` and `TokenEndpoint` issues the access token by `DefaultTokenServices` which is a default package of `AuthorizationServerTokenService`.

When access token is issued, the client information registered in authorization server through `ClientDetailsService` is fetched, and the same is used for verifying the possibility of access token issue. In OAuth 2.0 specifications, the registration procedure of clients that uses authorization server is not prescribed, and the client registration procedure is not supported in Spring Security OAuth as well.

Hence, when client registration interface is to be provided in the application, it must be implemented independently.

Spring Security authentication function is used for resource owner authentication. Refer to *Authentication* for the details.

Configuration of authorization server is shown below.

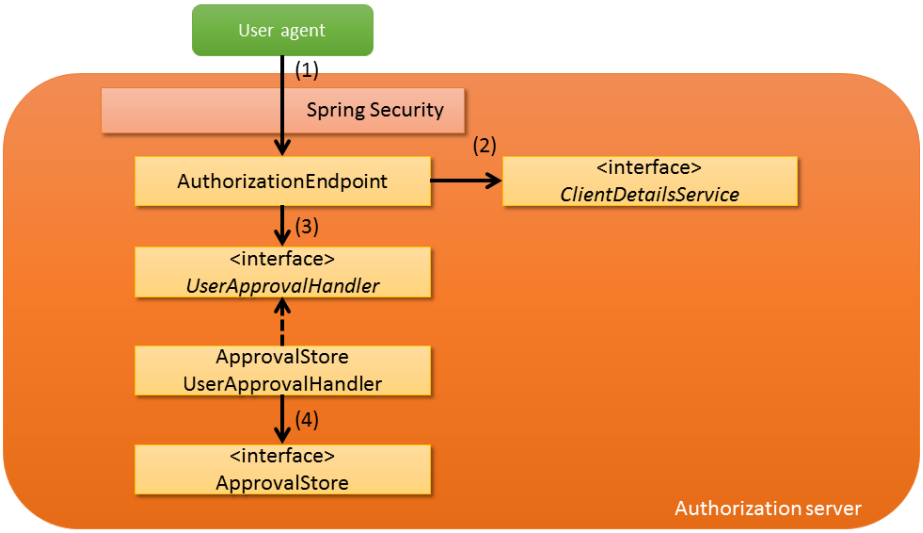


Table.9.57 **Working of authorization server (In case of authorization end point access)**

Sr. No.	Description
(1)	`AuthorizationEndpoint` process is executed by accessing the authorization end point (/oauth/authorize) of authorization server by the user agent.
(2)	Call `ClientDetailsService` method, and verify request parameter after fetching the client information registered in advance.
(3)	Call `UserApprovalHandler` method and check whether authorization of scope is registered for client When authorization is not registered, the screen (/oauth/confirm_access) asking the resource owner for authorization through user agent is displayed. In such case, the scope to be inquired is linked by getting the product of request parameter and client information registered in advance, and by using `@SessionAttributes` of Spring MVC.
(4)	Manage authorization status by `ApprovalStore` in `ApprovalStoreUserApprovalHandler` which is package of `UserApprovalHandler`. When authorization is performed by resource owner, call `ApprovalStore` method and register specified information.

Note: As described earlier, the scope to be inquired is the product of scope registered previously in authorization server and scope specified by request parameter at the time of authorization request by client. For example, when the scope of READ and CREATE is specified by request parameters for the client assigning the scopes such as READ, CREATE, DELETE on authorization server, then the scope such as READ, CREATE which is product of (READ,CREATE,DELETE) and (READ,CREATE) can be assigned. When the scope that is not assigned to client on authorization server is specified by request parameter, an error occurs and access is denied.

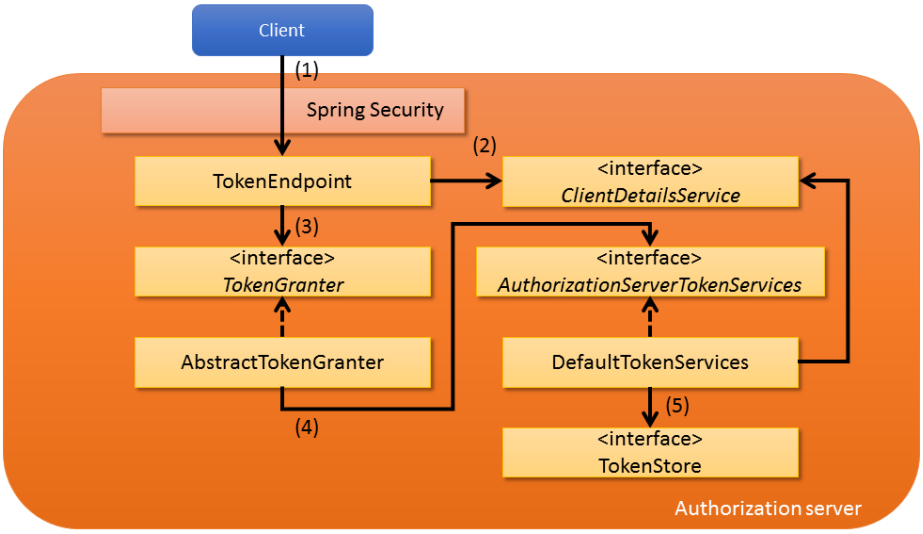

Client
(1)
Spring Security
(2)
TokenEndpoint
<interface>
ClientDetailsService
(3)
<interface>
TokenGranter
<interface>
AuthorizationServerTokenServices
AbstractTokenGranter
DefaultTokenServices
(4)
(5)
<interface>
TokenStore
Authorization server

Table.9.58 **Working of authorization server (In case of token end point access)**

Sr. No.	Description
(1)	`TokenEndpoint` process is executed by accessing the token end point (/oauth/token) of authorization server by the client
(2)	Call `ClientDetailsService` method and check whether the scope of request parameter is registered in client after fetching the previously registered client information.
(3)	When scope is registered, call `TokenGranter` method and issue access token.
(4)	Call `AuthorizationServerTokenServices` method and issue an access token in `AbstractTokenGranter` which is an implementation of `TokenGranter`. `AbstractTokenGranter` which is the base class of `TokenGranter` implemented as per grant type, and actual process is assigned to each class.
(5)	Call `TokenStore` method and manage status of access token in `DefaultTokenServices` which is an implementation of `AuthorizationServerTokenServices`.

Resource server

Resource server processes the access request for protected resource from the client and returns the response. Resource server verifies whether the access token added to the request from client is within the validity period and gets the authentication information associated with the access token. After fetching the authentication information, requested resource verifies whether the concerned access token is within scope. Process after access token verification can be implemented similar to the normal applications.

Spring Security OAuth implements the verification function of access token, by using authentication and authorization framework of Spring Security. `OAuth2AuthenticationProcessingFilter` provided by Spring Security OAuth is specifically used in `ServletFilter`. `OAuth2AuthenticationEntryPoint` is

used as an `AuthenticationEntryPoint` interface and `OAuth2AuthenticationManager` is used as `AuthenticationManager` respectively. Refer to "*Authentication*" for the details of Spring Security.

Configuration of resource server is as follows.

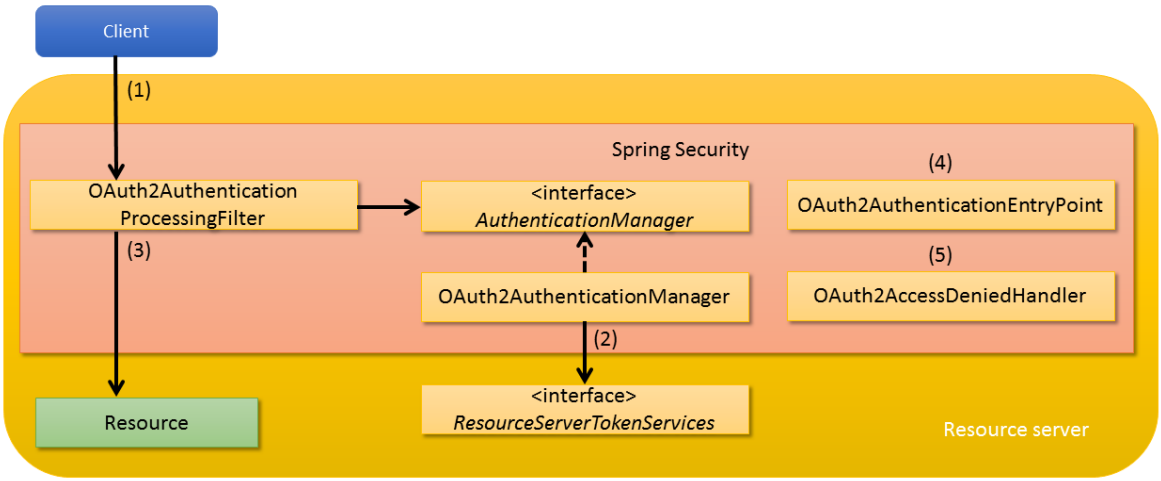


Table.9.59 **Working of resource server**

Sr. No.	Description
(1)	When client accesses the resource server at the beginning, `OAuth2AuthenticationProcessingFilter` is called. Access token is extracted in `OAuth2AuthenticationProcessingFilter`.
(2)	After extracting the access token, fetch the authentication information associated with access token through `ResourceServerTokenServices` in `OAuth2AuthenticationManager` which is an implementation of `AuthenticationManager`. Further, verify the access token when authentication information is fetched. As a method of fetching authentication information associated with access token, a method of fetching the same by using `TokenStore` commonly with the authorization server is listed besides the method of inquiring by HTTP to authorization server. How to fetch the authentication information depends on the implementation of `ResourceServerTokenServices`.
(3)	When verification of access token is successful, return the resource for the requests from client.
(4)	Handle the exception occurred at the time of authentication error by using `OAuth2AuthenticationEntryPoint`and receive error response.
(5)	Handle the exception occurred at the time of authorization error by using `OAuth2AccessDeniedHandler` and receive error response.

Client

Client authorizes the resource owner and fetches the access token, accesses the resource protected on resource server on behalf of resource owner. In such a case, the access token issued from authorization server is added for the request to the resource.

Spring Security OAuth provides `OAuth2RestTemplate` which is OAuth 2.0 implementation of `RestOperations`, as the implementation method of client's basic functions.

In `OAuth2RestTemplate`, authorization function required in authorization code grant is implemented by using `OAuth2ClientContextFilter` as servlet filter, in addition to functions such as issuing access token, re-issuing access token using refresh token and accessing the resource server using access token.

Further, in `OAuth2RestTemplate`, access token acquired from authorization server along with grant type specified `OAuth2ProtectedResourceDetails` is saved in `DefaultOAuth2ClientContext` which is an implementation of `OAuth2ClientContext`. `DefaultOAuth2ClientContext` is defined as Bean of session scope by default, and access token can be shared among multiple requests.

When function to access resource server is to be developed, it is implemented similarly as development of usual REST client, except using `OAuth2RestTemplate` provided by Spring Security OAuth instead of `RestTemplate` provided by Spring Web, as an implementation of package of `RestOperations`.

Configuration while using `OAuth2RestTemplate` as client function is given below.

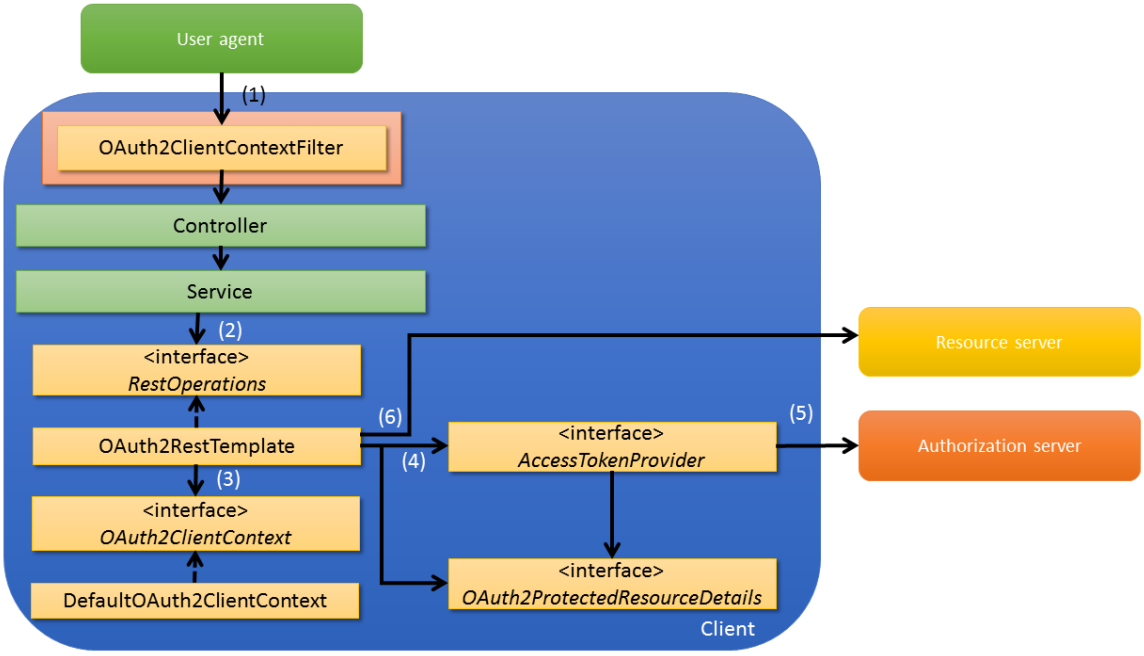


Table.9.60: **Working of client**

Sr. No,	Description
(1)	User agent accesses the `Controller` so that `Service` of client can be called. Servlet function (`OAuth2ClientContextFilter`) for capturing `UserRedirectRequiredException` that may occur in (5) is applied for access along with access to resource server. User agent can access the authorization end point on authorization server, by applying this servlet function when ‘‘ UserRedirectRequiredException‘‘ occurs.
(2)	Call `OAuth2RestTemplate` from `Service`.
(3)	Before accessing resource server, confirm that access token is retained by `DefaultOAuth2ClientContext` which is retained as a member. When access token is saved and is within the validity period, specify access token fetched from `DefaultOAuth2ClientContext` and access the resource server.
(4)	When access token is not saved at first access, or when validity period is lapsed, call `AccessTokenProvider` and fetch the access token.
(5)	Fetch `AccessTokenProvider`the access token for grant type defined in `OAuth2ProtectedResourceDetails` as detailed information of resource. `UserRedirectRequiredException` is thrown when fetching of authorization code is not completed in `AuthorizationCodeAccessTokenProvider` which is an implementation of authorization code grant.
(6)	Specify access token fetched in (3) or (5) and access resource server. When an error occurs such as “access token expired” (`AccessTokenRequiredException`) while accessing, perform subsequent processes from (4) onwards again after initializing retained access token.

9.10.2 How to use

Bean definition example and implementation method required for using Spring Security OAuth is described.

Set-up of Spring Security OAuth

Spring Security OAuth is added as a dependent library in order to use the class provided by Spring Security OAuth.

```
<!-- (1) -->
<dependency>
    <groupId>org.springframework.security.oauth</groupId>
    <artifactId>spring-security-oauth2</artifactId>
</dependency>
```

Sr. No.	Description
(1)	Add Spring Security OAuth as a dependent library in `pom.xml` of project using Spring Security OAuth. When resource server, authorization server, client are added as different project, respective description should be added.

Note: As given in above setting example, since it is assumed that dependency library version is managed by the parent project "terasoluna- gfw- parent", it is not necessary to specify the version in pom.xml The dependent library mentioned above is defined in Spring IO Platform used by terasoluna-gfw-parent.

Application settings

Settings of the application using Spring Security OAuth are described.

As shown in "*Authorization grant*",flow differs between authorization server and client differs in OAuth 2.0. Hence, settings should be done in the application using Spring Security OAuth in accordance with the grant type supported by application. Refer to implementation of each role for the setting details for each grant type.

Implementation of authorization server

Implementation method of authorization server is described.

Authorization server provides end points for "fetching authorization from resource owners" and "issuing access tokens", by using the function of Spring Security OAuth. Further, while accessing the above-mentioned end point, it is necessary to authenticate the resource owner or client, and these guidelines are implemented by using the technique for authentication/ authorization of Spring Security.

Creation of setting file (Authorization server)

`oauth2-auth.xml` is created as configuration file for defining in authorization server.

`oauth2-auth.xml` performs Bean definition of end points for providing the functions of authentication server, and makes security setting for the end points and sets the grant type supporting the authorization server.

- `oauth2-auth.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:sec="http://www.springframework.org/schema/security"
       xmlns:oauth2="http://www.springframework.org/schema/security/oauth2"
       xsi:schemaLocation="http://www.springframework.org/schema/security
           http://www.springframework.org/schema/security/spring-security.xsd
           http://www.springframework.org/schema/security/oauth2
           http://www.springframework.org/schema/security/spring-security-oauth2.
↪xsd
           http://www.springframework.org/schema/beans
           http://www.springframework.org/schema/beans/spring-beans.xsd">


</beans>
```

Next, settings are described in `web.xml` in order to read the created `oauth2-auth.xml`.

- `web.xml`

```
<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>
        classpath*:META-INF/spring/applicationContext.xml
        classpath*:META-INF/spring/oauth2-auth.xml  <!-- (1) -->
        classpath*:META-INF/spring/spring-security.xml
```

```
    </param-value>
</context-param>
```

Sr. No.	Description
(1)	Specify a Bean definition file of OAuth 2.0. It should be described before `spring-security.xml`, by considering the case when URL targeted for access control set in `oauth2-auth.xml` is included in the URL targeted for access control set in `spring-security.xml`.

Defining authorization server

Definitions of authorization server are added as below.

- `oauth2-auth.xml`

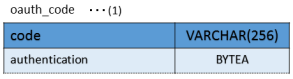
```
<oauth2:authorization-server
     token-endpoint-url="/oth2/token"
     authorization-endpoint-url="/oth2/authorize" >  <!-- (1) -->
    <oauth2:authorization-code />  <!-- (2) -->
    <oauth2:implicit />  <!-- (3) -->
    <oauth2:refresh-token />  <!-- (4) -->
    <oauth2:client-credentials />  <!-- (5) -->
    <oauth2:password />  <!-- (6) -->
</oauth2:authorization-server>
```

Sr. No.	Description
(1)	Use `<oauth2:authorization-server>` tag and define settings of authorization server. Authorization end points for authorization and token end points for issuing the access tokens are registered as components, by using the `<oauth2:authorization-server>` tags. Specify URL of token end point in `token-endpoint-url` attribute. When it is not specified, default value "/oauth/token" is specified. Specify URL of authorization end point in `authorization-endpoint-url` attribute. When it is not specified, default value "/oauth/authorize" is specified.
(2)	Use `<oauth2:authorization-code>` tag and support authorization code grant.
(3)	Use `<oauth2:implicit>` tag and support implicit grant.
(4)	Use `<oauth2:refresh-token>` tag and support refresh token.
(5)	Use `<oauth2:client-credentials>` tag and support client credential grant.
(6)	Use `<oauth2:password>` tag and support resource owner password credential grant.

Note: When multiple supporting grant types are to be specified, they should be specified in the above mentioned sequence.

Note: Authorization code is used for only the short period from issuing the authorization code till issuing the access token, hence it is managed in in-memory by default. In case of configuration of multiple authorization servers, it should be managed in DB, in order to share authorization codes among multiple servers. When authorization code is to be managed in DB, the following table, consisting of the column having authorization code as primary key and the column having authentication information, is created. Following example explains the DB definitions when PostgreSQL is used.

Bean ID of `org.springframework.security.oauth2.provider.code.JdbcAuthorizationCodeServices`



managing authorization code in DB is specified in `authorization-code-services-ref` of `<oauth2:authorization-code>` tag, in the configuration file of authorization server. Data source to be connected to table for authorization code storage is specified in the constructor of `JdbcAuthorizationCodeServices`. **Always** refer to *Transaction control* for the precautions while managing the authorization code permanently in DB.

- `oauth2-auth.xml`

```
<oauth2:authorization-server
     token-endpoint-url="/oth2/token"
     authorization-endpoint-url="/oth2/authorize" >
    <oauth2:authorization-code authorization-code-services-ref=
↪"authorizationCodeServices"/>
    <!-- omitted -->
</oauth2:authorization-server>

<bean id="authorizationCodeServices"
      class="org.springframework.security.oauth2.provider.code.
↪JdbcAuthorizationCodeServices">
    <constructor-arg ref="codeDataSource"/>
</bean>

<!-- omitted -->
```

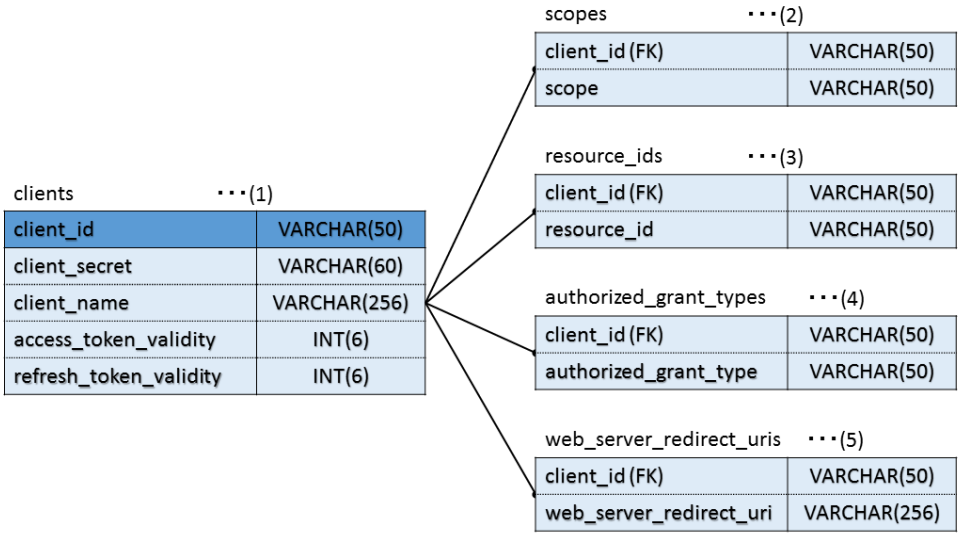
Client authentication

Clients that have accessed end points should be authenticated, in order to confirm whether they are registered clients. Client is authenticated by verifying the client ID and password passed by client as parameters based on the client information retained by authorization server. Basic authentication is used for authentication.

Spring Security OAuth provides the implementation class of `oorg.springframework.security.oauth2.provider.Cl` which is an interface for fetching the client information. Further, `org.springframework.security.oauth2.provider.client.BaseClientDetails` class which is implementation class of `ClientDetails` interface is provided as a class retaining the client information. `BaseClientDetails` provides the basic parameters such as client ID and supporting grant type etc. by using OAuth 2.0, and the parameters can be added by extending `BaseClientDetails`. In this case, extension

of `BaseClientDetails` and creation of `ClientDetailsService` implementation class is performed, and client information wherein client name is added as individual parameter is managed by using DB and the implementation method for authentication is explained.

At first, DB is created as below.



Sr. No.	Description
(1)	A table to retain client information. client_id is considered as primary key. Roles of each column are as below. ・client_id: A column to retain the client ID which is an ID to identify a client. ・client_secret: A column to retain the password which is used to authenticate the client. ・client_name: An individual column to retain the client name. Not required since it is an individual column. ・access_token_validity: A column to retain the validity period [seconds] of access token. ・refresh_token_validity: A column to retain the validity period [seconds] of refresh token.
(2)	A table to retain the scope information. It is mapped with the client information by considering client_id as the external key. Scope used for client authorization is retained in scope column. Records are registered only for the scope of client.

次のページに続く

Table. 9.61 – 前のページからの続き

Sr. No.	Description
(3)	A table to retain the resource information. It is mapped with the client information by considering client_id as the external key. Resource ID used in resource server for identifying whether it is a resource that can be accessed by client is retained in resource_id column. Access to resource is permitted, only when resource ID defined for the resource retained by resource server is included in resource ID registered here. Records are registered only for the resource IDs accessible by client. When not even single resource ID is registered, all resources can be accessed, hence precaution must be taken when it is not registered.
(4)	A table to retain the grant information. It is mapped with the client information by considering client_id as the external key. Grant to be used by client is retained in authorized_grant_type column. Records are registered only for the grant count to be used by client
(5)	A table to retain the redirect URL information. It is mapped with the client information by considering client_id as the external key. web_server_redirect_uri column retains the URL that redirects user agent after authorization by resource owner. Redirect URLs are used only for authorization code grants, implicit grants. Table itself is not required, when grant types other than authorization code grant and implicit grant are used. An error occurs, when there is no URL declared by client at the time of authorization request and when there is no redirect URL matching with host and route path. Records are registered only for the URLs that can be declared by client.

A model retaining the client information is created.

- `Client.java`

```
public class Client implements Serializable{
    private String clientId; // (1)
    private String clientSecret; // (2)
    private String clientName; // (3)
    private Integer accessTokenValidity; // (4)
```

```
    private Integer refreshTokenValidity; // (5)
    // Getters and Setters are omitted
}
```

Sr.No.	Description
(1)	A field to retain the client ID identifying the client.
(2)	A field to retain the password used for client authentication.
(3)	An extended field to retain the client name which is not provided in Spring Security OAuth. Not required, since it is an extended field.
(4)	A field to retain the validity period [seconds] of access token.
(5)	A field to retain the validity period [seconds] of refresh token.

Detailed information of client can be extended easily by creating the class inheriting the `BaseClientDetails` class.

- `OAuthClientDetails.java`

```
public class OAuthClientDetails extends BaseClientDetails{
    private Client client;
    // Getter and Setter are omitted
}
```

`org.springframework.security.oauth2.provider.ClientDetailsService` is an interface for fetching the detailed client information required in authorization process from the data store. Creation of `ClientDetailsService` implementation class is described below.

- `OAuthClientDetailsService.java`

```
@Service("clientDetailsService") // (1)
@Transactional
public class OAuthClientDetailsService implements ClientDetailsService {

    @Inject
```

```
    ClientRepository clientRepository;

    @Override
    public ClientDetails loadClientByClientId(String clientId)
            throws ClientRegistrationException {

        Client client = clientRepository.findClientByClientId(clientId); // (2)

        if (client == null) { // (3)
              throw new NoSuchClientException("No client with requested id: " +␣
↪clientId);
        }

        // (4)
        Set<String> clientScopes = clientRepository.
↪findClientScopesByClientId(clientId);
        Set<String> clientResources = clientRepository.
↪findClientResourcesByClientId(clientId);
        Set<String> clientGrants = clientRepository.
↪findClientGrantsByClientId(clientId);
        Set<String> clientRedirectUris = clientRepository.
↪findClientRedirectUrisByClientId(clientId);


         // (5)
        OAuthClientDetails clientDetails = new OAuthClientDetails();
        clientDetails.setClientId(client.getClientId());
        clientDetails.setClientSecret(client.getClientSecret());
        clientDetails.setAccessTokenValiditySeconds(client.
↪getAccessTokenValidity());
        clientDetails.setRefreshTokenValiditySeconds(client.
↪getRefreshTokenValidity());
        clientDetails.setResourceIds(clientResources);
        clientDetails.setScope(clientScopes);
        clientDetails.setAuthorizedGrantTypes(clientGrants);
        clientDetails.setRegisteredRedirectUri(clientRedirectUris);
        clientDetails.setClient(client);

        return clientDetails;
    }

}
```

Sr. No.	Description
(1)	Add `@Service` annotation to the class level as service in order to use it as target for component-scan. Specify Bean name as `clientDetailsService`.
(2)	Client information fetched from database is stored in the Client model.
(3)	When the client information is not found, an exception of Spring Security OAuth - `NoSuchClientException` is generated.
(4)	Fetch information associated with the client. If the process efficiency is reduced by calling Repository by dividing it for multiple times, fetch collectively in (2).
(5)	Set various types of fetched information in `OAuthClientDetails` field.

Add settings necessary for client authentication to `oauth2-auth.xml`.

- `oauth2-auth.xml`

```
<sec:http pattern="/oth2/*token*/**"
    authentication-manager-ref="clientAuthenticationManager" realm="Realm">  <!--␣
↪(1) -->
    <sec:http-basic />           <!-- (2) -->
    <sec:csrf disabled="true"/>  <!-- (3) -->
    <sec:intercept-url pattern="/**" access="isAuthenticated()"/>  <!-- (4) -->
</sec:http>

<oauth2:authorization-server
     token-endpoint-url="/oth2/token"
     authorization-endpoint-url="/oth2/authorize"
     client-details-service-ref="clientDetailsService">  <!-- (5) -->
    <oauth2:authorization-code />
    <oauth2:implicit />
    <oauth2:refresh-token />
    <oauth2:client-credentials />
    <oauth2:password />
</oauth2:authorization-server>
```

```
<sec:authentication-manager id="clientAuthenticationManager">  <!-- (6) -->
    <sec:authentication-provider user-service-ref="clientDetailsUserService" >  <!-
↪- (7) -->
        <sec:password-encoder ref="passwordEncoder"/>  <!-- (8) -->
    </sec:authentication-provider>
</sec:authentication-manager>

<bean id="clientDetailsUserService"
    class="org.springframework.security.oauth2.provider.client.
↪ClientDetailsUserDetailsService">  <!-- (9) -->
    <constructor-arg ref="clientDetailsService" />  <!-- (10) -->
</bean>
```

Sr. No.	Description
(1)	Specify a location below `/oth2/*token*/` as a target for access control as an endpoint URL in order to perform security settings for endpoint related to access token operation. Here, the endpoint URL acting as a access control target is set to a value starting from `/oth2/`, endpoint URL defined by Spring Security OAuth and its default value are as below. ・`/oauth/token` - an endpoint URL of endpoint used in issuing a token ・`/oauth/check_token`- an endpoint URL of endpoint used to verify a token ・`/oauth/token_key`- an endpoint URL of endpoint used for fetching a public key, when JWT signature is created by public key encryption method Specify a Bean of `AuthenticationManager` for client authentication defined in (5), in `authentication-manager-ref` attribute. Also, when Basic authentication is enabled by XML Namespace as shown in (2), Realm name of Basic authentication becomes `"Spring Security Application"`. Specify an appropriate value in `realm` attribute since internal information of the application gets revealed.
(2)	Apply Basic authentication to client authentication. For details, refer http://docs.spring.io/spring-security/site/docs/4.1.4.RELEASE/reference/html/basic.html .
(3)	Disable CSRF countermeasures function for accessing `/oth2/*token*/**`. Spring Security OAuth adopts validity check of request wherein state parameter is used, recommended as a CSRF countermeasure of OAuth 2.0.

次のページに続く

Table. 9.62 – 前のページからの続き

Sr. No.	Description
(4)	Settings which assign rights of access only to authenticated users, for locations under endpoint URL. How to specify access policy for Web resource, refer *Authorization*.
(5)	Specify a Bean of `OAuthClientDetailsService` in `client-details-service-ref` attribute. BeanID to be specified must match with Bean ID specified by implementation class of `ClientDetailsService`.
(6)	Define a Bean for `AuthenticationManager` for authentication of client. `AuthenticationManager` used in the authentication of resource owner and Bean ID of alias must be specified. For authentication of resource owner, refer *Resource owner authentication*.
(7)	Specify a Bean of `ClientDetailsUserDetailsService` defined in (9), in `user-service-ref` attribute of `sec:authentication-provider`.
(8)	Specify a Bean of `PasswordEncoder` used in password hashing, which is used in client authentication. For details of password hashing, refer *Password hashing*.
(9)	Define a Bean for `ClientDetailsUserDetailsService` which is an implementation class of `UserDetailsService` interface. `UserDetailsService` used in the resource owner authentication and Bean ID of alias must be specified.
(10)	Specify a Bean of `OAuthClientDetailsService` which fetches client information from database, in constructor argument. Bean ID to be specified must match with Bean ID specified by the implementation class of `ClientDetailsService`.

Resource owner authentication

When an authorization code grant is used for fetching an access token, a resource owner must be authenticated by a method like providing a login screen among others.

This guideline assumes the use of Spring Security for authentication of resource owner.

A URL with an authorization end point URL must be defined in authorization settings as an access policy to enable access to authorized endpoint URL only to authenticated users. Further, a controller URL which displays the authorization screen and a controller URL which handles exceptions in authorization endpoint must also be similarly defined as access policies.

For the controller which displays authorization screen, refer *Customising scope authorization screen* and for the controller which handles exception in authorization endpoint, refer *Error handling at the time of authorization request*.

For details of Spring Security, refer *Authentication* and *Authorization*.

Definition examples of access policies which include authorization endpoint URL, a URL of controller which displays authorization and a URL of controller which handles errors of authorization endpoint are shown below.

- `spring-security.xml`

```
<sec:http authentication-manager-ref="userLoginManager"> <!-- (1) -->
    <sec:form-login login-page="/login"
        authentication-failure-url="/login?error=true"
        login-processing-url="/login" />
    <sec:logout logout-url="/logout"
        logout-success-url="/"
        delete-cookies="JSESSIONID" />
    <sec:access-denied-handler ref="accessDeniedHandler"/>
    <sec:custom-filter ref="userIdMDCPutFilter" after="ANONYMOUS_FILTER"/>
    <sec:session-management />
    <sec:intercept-url pattern="/login/**" access="permitAll" />
    <sec:intercept-url pattern="/oth2/**" access="isAuthenticated()" /> <!-- (2) --
↪>
    <!-- omitted -->
</sec:http>

 <sec:authentication-manager id="userLoginManager"> <!-- (3) -->
    <sec:authentication-provider
        user-service-ref="userDetailsService">
        <sec:password-encoder ref="passwordEncoder" />
```

```
    </sec:authentication-provider>
</sec:authentication-manager>

<bean id="userDetailsService"
    class="org.springframework.security.core.userdetails.jdbc.JdbcDaoImpl">
    <property name="dataSource" ref="dataSource" />
</bean>
```

Sr. No.	Description
(1)	Specify location under root (“/”) as an access control target, which includes authorization endpoint URL - `/oth2/authorize`, a URL of controller which displays authorization screen - `/oauth/confirm_access` and a URL which handles errors of authorization endpoint - `/oauth/error`.
(2)	Specify location under root (“/”) to enable the access only by authenticated users, which includes authorization endpoint URL - `/oth2/authorize`, a URL of controller which displays authorization screen - `/oauth/confirm_access` and a URL of controller which handles errors of authorization endpoint - `/oauth/error`.
(3)	Define a Bean for `AuthenticationManager` for authenticating resource owner. `AuthenticationManager` used in client authentication and Bean ID of alias must be specified.

Authorization for each scope

A setup method wherein each scope is authorized individually instead of authorizing the requested scope together while requesting the authorization for resource owner, is explained.

Authorization information must be controlled in a DB for performing perpetual management to prevent loss of authorization information at the time of restarting authorization server or to share authorization information across multiple authorization servers. Following DB is created for storing authorization information for each scope. The example below explains DB definition when PostgreSQL is used.

oauth_approvals ・・・(1)

userId	VARCHAR(50)
clientId	VARCHAR(50)
scope	VARCHAR(50)
status	VARCHAR(10)
expiresAt	TIMESTAMP
lastModifiedAt	TIMESTAMP

Sr. No.	Description
(1)	A table to retain authorization information. userId, clientId and scope are used as primary keys. Role of each column is as below. ・userId: A column to retain user name of resource owner who performs the authorization. ・clientId: A column to retain client ID of the client authorized by the resource owner. ・scope: A column to retain scope authorized by resource owner. ・status: A column to check whether the authorization is done by the resource owner. `APPROVED` is set when authorized and `DENIED` is set when authorization is denied. ・expiresAt: A column to retain validity period of authorization information. ・lastModifiedAt: A column to retain last modified date for authorization information.

Fetch authorization for each scope of resource owner and apply settings to store the same in DB and control.

Implementation example is as below.

- `oauth2-auth.xml`

```
<oauth2:authorization-server
     client-details-service-ref="clientDetailsService"
     token-endpoint-url="/oth2/token"
     authorization-endpoint-url="/oth2/authorize"
     user-approval-handler-ref="userApprovalHandler"> <!-- (1) -->

     <!-- omitted -->

</oauth2:authorization-server>

<bean id="userApprovalHandler"
      class="org.springframework.security.oauth2.provider.approval.
↪ApprovalStoreUserApprovalHandler">  <!-- (2) -->
    <property name="clientDetailsService" ref="clientDetailsService"/>
    <property name="approvalStore" ref="approvalStore"/>
```

```
    <property name="requestFactory" ref="requestFactory"/>
    <property name="approvalExpiryInSeconds" value="3200" />
</bean>

<bean id="approvalStore"
      class="org.springframework.security.oauth2.provider.approval.
↪JdbcApprovalStore">  <!-- (3) -->
    <constructor-arg ref="approvalDataSource"/>
</bean>

<bean id="requestFactory"
      class="org.springframework.security.oauth2.provider.request.
↪DefaultOAuth2RequestFactory">
    <constructor-arg ref="clientDetailsService"/>
</bean>
```

Sr. No.	Description
(1)	Specify a Bean of `ApprovalStoreUserApprovalHandler` defined in (2), in `user-approval-handler-ref` as `UserApprovalHandler` which performs authorization process of scope.
(2)	Define a Bean for `ApprovalStoreUserApprovalHandler` which performs authorization process of scope. Specify a Bean of `JdbcApprovalStore` defined in (3), in `approvalStore` property which manages authorization results of resource owner. Specify a Bean of `OAuthClientDetailsService` in `clientDetailsService` property which fetches client information to be used in authorization process of scope. Specify a Bean of `DefaultOAuth2RequestFactory` in `requestFactory` property. Bean set in `requestFactory` property is not used by `ApprovalStoreUserApprovalHandler`, however, since an error occurs at the time of Bean generation of `ApprovalStoreUserApprovalHandler` when the setting is not applied, settings in `requestFactory` property is necessary. When validity period [Seconds] of authorization information is to be specified, validity period [Seconds] is set in `approvalExpiryInSeconds` property. If any setting is not applied, the authorization information will remain valid a month from authorization.
(3)	Define a Bean for `JdbcApprovalStore` which manages authorization information in DB. Specify a database in the constructor to connect to the table for storing authorization information. For precautions while performing perpetual management of authorization information in DB, **always** refer *Transaction control*.

Note: When perpetual management is not required to be performed for authorization information and is to be managed by in-memory instead of a DB, a Bean is to be defined for `org.springframework.security.oauth2.provider.approval.InMemoryApprovalStore` as `approvalStore`.

Customising scope authorization screen

When the scope authorization screen is to be customised, it can be done by creating controller and JSP. An example is explained below wherein the scope authorization screen is customised.

When an endpoint which requests authorization to resource owner is to be called, it is forwarded to (context path)/oauth/confirm_access. A controller which handles (context path)/oauth/confirm_access is created.

- `OAuth2ApprovalController.java`

```
@Controller
public class OAuth2ApprovalController {

    @RequestMapping("/oauth/confirm_access") // (1)
    public String confirmAccess() {
        // omitted
        return "approval/oauthConfirm";
    }

}
```

Sr. No.	Description
(1)	Use `@RequestMapping` annotation and perform mapping as a method for accessing `"/oauth/confirm_access"`.

Next, a JSP of scope authorization screen is created. Since scope for authorization is registered in the Model by `scopes` key, scope authorization screen is displayed by using the same.

- `oauthConfirm.jsp`

```
<%@ taglib prefix="sec" uri="http://www.springframework.org/security/tags" %>

<body>
    <div id="wrapper">
        <h1>OAuth Approval</h1>
        <p>Do you authorize '${f:h(authorizationRequest.clientId)}' to access your protected resources?</p>
        <form id='confirmationForm' name='confirmationForm' action='${pageContext.request.contextPath}/oth2/authorize' method='post'>
            <c:forEach var="scope" items="${scopes}" varStatus="status">  <!-- (1) -->
                <li>
                    ${f:h(scope.key)}  <!-- (2) -->
                    <input type='radio' name="${f:h(scope.key)}" value='true'/>Approve
                    <input type='radio' name="${f:h(scope.key)}" value='false'/>Deny
                </li>
            </c:forEach>
            <input name='user_oauth_approval' value='true' type='hidden'/>  <!-- (3) -->
            <sec:csrfInput />  <!-- (4) -->
```

```
            <label>
                <input name='authorize' value='Authorize' type='submit'/>
            </label>
        </form>
    </div>
</body>
```

Sr. No.	Description
(1)	Output a radio button for specifying authorization for each scope. Since target scope is included in `scopes` list, specify `scopes` in `items`.
(2)	Since key names of elements which retain `scopes` list are used as respective scope names, key names are displayed on the screen. Set output of Approve and Deny radio buttons for selecting “Authorise” and “Deny”.
(3)	Spring Security OAuth assigns `user_oauth_approval` to the request parameter by embedding `user_oauth_approval` as a hidden item. `user_oauth_approval` assigned to the request parameter can be used for executing a method which performs scope authorization of authorization endpoint.
(4)	Specify `<sec:csrfInput>` element in `<form>` element of HTML in order to deliver CSRF.

Error handling at the time of authorization request

When an authorization error (error related to security like “client does not exist” or redirect URL check error) occurs in authorization end point, `OAuth2Exception` offered by Spring Security OAuth is thrown and the request (context path) /oauth/error is forwarded. Hence, when exception in authorization end point is to be handled, a controller to handle (context path)/oauth/error must be created.

- `OAuth2ErrorController.java`

```
@Controller
public class OAuth2ErrorController {

    @RequestMapping("/oauth/error") // (1)
    public String handleError() {
        // omitted
```

```
        return "common/error/oauthError";
    }

}
```

Sr. No.	Description
(1)	Use `@RequestMapping` annotation and perform mapping as a method for accessing `"/oauth/error"`.

Next, a JSP of error screen thus displayed is created. Since details of error occurred in authorization end point are registered in the Model by `error` key, error details are displayed on the screen using the same.

- `oauthError.jsp`

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>OAuth Error!</title>
<link rel="stylesheet"
    href="${pageContext.request.contextPath}/resources/app/css/styles.css">
</head>
<body>
    <div id="wrapper">
        <h1>OAuth Error!</h1>
        <p>${f:h(error.OAuth2ErrorCode)}</p> <!-- (1) -->
        <p>${f:h(error.message)}</p> <!-- (2) -->
    <br>
    </div>
</body>
</html>
```

Sr. No.	Description
(1)	Output error response included in `error`.
(2)	Output error message included in `error`.

Note: When an error other than authorization error (error related to security error like “client does not exist”, redirect URL check error etc) occur in authorization endpoint, errors are notified on the client side after redirecting.

How to share access token with resource server

Authorization server links with the access token via `TokenServices` so that the resource server can determine the authorization for accessing a resource based on access token. Various methods exist for methods for linking.

Sr. No.	Method of linking	Description
(1)	Linking via DB	A method wherein a common DB is used and access token is linked. It can be used when resource server and authorization server share a DB. Authorization server specifies `DefaultTokenServices` as an implementation of TokenService, and `JdbcTokenStore` as an implementation of TokenStore.
(2)	Linking via HTTP access	A method wherein access token is linked by accessing HTTP. This method is used when resource server and authorization server cannot use a common DB. Since the resource server requests the fetching and verification of access token to authorization server, authorization server is overloaded. Authorization server specifies `DefaultTokenServices` as an implementation of TokenService. Specify `JdbcTokenStore` when the access token is to be managed by DB and `InMemoryTokenStore` when it is to be managed by memory, as an implementation of TokenStore. Implementation wherein the access token is managed by memory is exclusively for testing purpose since access tokens are lost due to operations like restarting a server etc.

次のページに続く

Table. 9.63 – 前のページからの続き

Sr. No.	Method of linking	Description
(3)	Linking via JWT	A method wherein JWT is used and access token is linked. This method is used when resource server and authorization server cannot use a common DB. Since request is not sent to authorization server for fetching an access token, when compared with linking via HTTP access, authorization server is not overloaded. Authorization server specifies `DefaultTokenServices` as an implementation of TokenService and `JwtTokenStore` as an implementation of TokenStore. Access token is signed, encoded and decoded by using `org.springframework.security.oauth2.provider.token.store.Jwt` There are two methods for signing and verification of access token - a method which uses public key and a method which uses common key.
(4)	Linking via memory	A method wherein access token is linked by sharing the memory. It can be used when an application is designed wherein resource server and authorization server become a single process. Authorization server specifies `DefaultTokenServices` as an implementation of TokenService and `InMemoryTokenStore` as an implementation of TokenStore. Since the access token is linked via memory, linking of access token using common DB and HTTP access is not necessary. Implementation wherein the access token is shared via memory is exclusively for testing purpose since access tokens are lost due to operations like restarting the server etc.

Todo

TBD

How to implement linking by using JWT will be explained in detail in next version.

Here, a method wherein access token is linked via common DB is explained as a representative method. A method to link via HTTP access is explained in How To Extend of this section.

- *Linkage through HTTP access*

When access token is linked via a common DB, `JdbcTokenStore` offered by Spring Security OAuth is used.

Implementation example is as below.

- `oauth2-auth.xml`

```
<oauth2:authorization-server
     client-details-service-ref="clientDetailsService"
     token-endpoint-url="/oth2/token"
     authorization-endpoint-url="/oth2/authorize"
     user-approval-handler-ref="userApprovalHandler"
     token-services-ref="tokenServices">  <!-- (1) -->
    <oauth2:authorization-code />
    <oauth2:implicit />
    <oauth2:refresh-token />
    <oauth2:client-credentials />
    <oauth2:password />
</oauth2:authorization-server>

<bean id="tokenServices"
    class="org.springframework.security.oauth2.provider.token.DefaultTokenServices
↪">  <!-- (2) -->
    <property name="tokenStore" ref="tokenStore" />
    <property name="clientDetailsService" ref="clientDetailsService" />
    <property name="supportRefreshToken" value="true" />  <!-- (3) -->
</bean>

<bean id="tokenStore"
  class="org.springframework.security.oauth2.provider.token.store.JdbcTokenStore">
↪<!-- (4) -->
  <constructor-arg ref="tokenDataSource" />
</bean>
```

Sr. No.	Description
(1)	Specify `tokenServices` defined in (2) in `token-services-ref` attribute as TokenService used by authorization server.
(2)	Specify `DefaultTokenServices` in `tokenServices` class. Specify `TokenStore` defined in (4), in `tokenStore` property as the token store which manages access tokens. This setting is applied to resource server as well when the access token is linked with resource server via common DB.
(3)	Specify `true` in `supportRefreshToken` attribute when refresh token is enabled.
(4)	Define a Bean for `JdbcTokenStore` as the token store. Specify a data source in the constructor for connecting to a table wherein token information is stored.

`JdbcTokenStore` creates following DB wherein a schema is defined for Spring Security OAuth, for linking the access tokens. The example below explains the DB definition while using PostgreSQL as a common DB.

oauth_access_token ・・・(1)

authentication_id	VARCHAR(256)
token	BYTEA
token_id	VARCHAR(256)
user_name	VARCHAR(50)
client_id	VARCHAR(50)
authentication	BYTEA
refresh_token	VARCHAR(256)

oauth_refresh_token ・・・(2)

token_id	VARCHAR(256)
token	BYTEA
authentication	BYTEA

Sr. No.	Description
(1)	A table which manages the access token. It is used to share information of access token issued by authorization server with the resource server. Roles of each column are as below. ・authentication_id: A column to store authentication key which uniquely identifies authentication information. It is used as a primary key. ・token: A column which retains token information as a binary after serializing. Validity period of access token, scope, token ID of access token, token ID of refresh token, token type which shows types of token to be used are stored as information of the token to be retained. ・token_id: A column to retain token ID which uniquely identifies access token. ・user_name: A column to retain user name of authenticated resource owner. ・client_id: A column to retain client ID of authenticated client. ・authentication: A column which retains authentication information of resource owner and client as a binary after serializing. ・refresh_token: A column which retains token ID of refresh token associated with access token.
(2)	A table which manages refresh token associated with access token. Roles of each column are as given below. ・token_id: A column to retain token ID which uniquely identifies refresh token. It is used as a primary key. ・token: A column which retains token information as binary after serializing. It retains validity period of refresh token. ・authentication: A column that retains authentication information of resource owner and client as binary after serializing. Information same as authentication information retained by table which manages access tokens, is retained.

Note: When the token is managed by common DB, the expired token is deleted within the timing of use of access token by client. Therefore, even if token has expired, access token is deleted only when it is accessed by the client. Expired tokens must be purged separately by batch processing in order to delete the same from DB.

Canceling a token (authorization server)

How to implement cancellation of issued access token is explained.

Access token can be cancelled by calling `revokeToken` method of a class which implements `ConsumerTokenService` interface. `DefaultTokenService` class implements `ConsumerTokenService` interface.

Authorization information can be deleted as well at the time of cancellation of access token. When authorization request is sent without deleting authorization information after cancellation of access token while using authorization code grant and implicit grant, authorization information at the time of previous authorization request is reused. Authorization information at the time of previous authorization request can be reused when validity period of authorization information is valid and entire authorization request scope is authorized.

Implementation example is given below.

An interface of service class which cancels a token and implementation class are created.

- `RevokeTokenService.java`

```
public interface RevokeTokenService {

    String revokeToken(String tokenValue, String clientId);

}
```

- `RevokeTokenServiceImpl.java`

```
@Service
@Transactional
public class RevokeTokenServiceImpl implements RevokeTokenService {

    @Inject
    ConsumerTokenServices consumerService; // (1)

    @Inject
    TokenStore tokenStore; // (2)

    @Inject
    ApprovalStore approvalStore; // (3)

    public String revokeToken(String tokenValue, String clientId){ // (4)
        // (5)
        OAuth2Authentication authentication = tokenStore.
↪readAuthentication(tokenValue);
        if (authentication != null) {
            if (clientId.equals(authentication.getOAuth2Request().getClientId()))
↪{ // (6)
                // (7)
```

```
                Authentication user = authentication.getUserAuthentication();
                if (user != null) {
                    Collection<Approval> approvals = new ArrayList<Approval>();
                    for (String scope : authentication.getOAuth2Request().
↪getScope()) {
                        approvals.add(
                                new Approval(user.getName(), clientId, scope, new␣
↪Date(), ApprovalStatus.APPROVED));
                    }
                    approvalStore.revokeApprovals(approvals);
                }
                consumerService.revokeToken(tokenValue); // (8)
                return "success";

            } else {
                return "invalid client";
            }
        } else {
            return "invalid token";
        }
    }
}
```

Sr. No.	Description
(1)	Inject implementation class of `ConsumerTokenService` interface which cancels access tokens.
(2)	Inject implementation class of `TokenStore` used to fetch authentication information while issuing an access token.
(3)	Inject implementation class of `ApprovalStore` used to fetch authorization information while issuing an access token. It is not required when authorization information is not to be deleted while cancelling an access token.
(4)	Receive value of access token to be cancelled and client ID used for checking the client as parameters.
(5)	Call `readAuthentication` method of implementation class of `TokenStore` and fetch authentication information while issuing an access token. Token is deleted only when authentication information can be successfully fetched.

次のページに続く

Table. 9.64 – 前のページからの続き

Sr. No.	Description
(6)	Fetch client ID used at the time of issuing access token, from authentication information and check whether it matches with client ID of request parameter. Delete access token only when it matches with client ID at the time of issuing an access token.
(7)	Fetch authentication information of resource owner at the time of issuing an access token. When authentication information of resource owner could be fetched, call `revokeApprovals` method of implementation class of `TokenStore` and delete authorization information. Since authentication information for resource owner does not exist while using client credential grant, the parameters to be passed to `revokeApprovals` method cannot be generated. Therefore, authorization information cannot be deleted when it is not possible to fetch authentication information of resource owner. This process is not required when authorization information is not deleted while cancelling the access token.
(8)	Call `revokeToken` method of implementation class `ConsumerTokenService` and delete access token and refresh token associated with access token.

A controller which receives a cancellation request of token is created.

- `TokenRevocationRestController.java`

```
@RestController
@RequestMapping("oth2")
public class TokenRevocationRestController {

    @Inject
    RevokeTokenService revokeTokenService;

    @RequestMapping(value = "tokens/revoke", method = RequestMethod.POST) // (1)
    @ResponseStatus(HttpStatus.OK)
    public String revoke(@RequestParam("token") String token,
        @AuthenticationPrincipal UserDetails user){

        // (2)
        String clientId = user.getUsername();
        String result = revokeTokenService.revokeToken(token, clientId); // (3)
        return result;
    }
```

Sr. no.	Description
(1)	Use `@RequestMapping` annotation and map as a method for accessing `"/oth2/tokens/revoke"`. The path specified here must apply Basic authentication and disable CSRF countermeasures, similar to settings performed in *Client authentication*.
(2)	Fetch client ID used while checking the cancellation of token, from authentication information generated in Basic authentication.
(3)	Call `RevokeTokenService` and delete a token. Pass value of access token received as the request parameter and client ID received from authentication information as parameters.

Tip: RFC 7009 states that `token_type_hint` can be arbitrarily assigned as a request parameter. `token_type_hint` is a hint to determine whether to delete access token or refresh token. `ConsumerTokenService` offered by Spring Security OAuth is not used in the implementation example above in order to delete both access token and refresh token by passing the access token.

Note: The client which requests deletion of token to the authorization server should also delete a token retained by the client after deletion of authorization server. For deletion of token of client server, refer *Cancellation of token (Client server)*.

Transaction control

It explains precautions of transaction control in authorization server.

When the information handled by Spring Security Auth (authorization code, authorization information, token) is to be managed in DB, for authorization server, transaction control must be considered.

Default implementation of `AuthorizationServerTokenServices` - `DefaultTokenServices` contains a method to issue access token and refresh token. These methods can be called under transaction control by assigning `@Transactional` to `createAccessToken` and `refreshAccessToken` respectively, however, the operation otherwise becomes non-transactional control.

Hence, when `Connection` fetched from `DataSource` is set to `autoCommit=false`, transaction control is imperative since information to be managed is not registered in DB. Further, although it is not mandatory when `autoCommit=true`, adequate care must be taken since it is necessary to consider transaction control to ensure data consistency.

An example of transaction control when authorization code and authorization information are managed in DB is shown below.

- `oauth2-auth.xml`

```
<!-- omitted -->

<tx:advice id="oauthTransactionAdvice">
    <tx:attributes>
        <tx:method name="*"/>
    </tx:attributes>
</tx:advice>

<aop:config>
    <aop:pointcut id="authorizationOperation"
                  expression="execution(* org.springframework.security.oauth2.
↪provider.code.AuthorizationCodeServices.*(..))"/> <!-- (1) -->
    <aop:pointcut id="approvalOperation"
                  expression="execution(* org.springframework.security.oauth2.
↪provider.approval.UserApprovalHandler.*(..))"/> <!-- (2) -->
    <aop:advisor pointcut-ref="authorizationOperation" advice-ref=
↪"oauthTransactionAdvice"/>
    <aop:advisor pointcut-ref="approvalOperation" advice-ref=
↪"oauthTransactionAdvice"/>
</aop:config>
```

Sr. No.	Description
(1)	Use AOP and set transaction boundary in each method which operates authorization code.
(2)	Use AOP and set transaction boundary in each method which operates authorization information.

Implementation of resource server

Here, how to implement a resource server is explained by using implementation example wherein authorization is set for REST API of TODO resource.

Creating a setup file (resource server)

A new Bean definition file for OAuth 2.0 is created while implementing a resource server.

Here, it is `oauth2-resource.xml`.

Following settings are added to `oauth2-resource.xml`.

- `oauth2-resource.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:oauth2="http://www.springframework.org/schema/security/oauth2"
       xmlns:sec="http://www.springframework.org/schema/security"
       xsi:schemaLocation="http://www.springframework.org/schema/security
           http://www.springframework.org/schema/security/spring-security.xsd
           http://www.springframework.org/schema/security/oauth2
           http://www.springframework.org/schema/security/spring-security-oauth2.
↪xsd
           http://www.springframework.org/schema/beans
           http://www.springframework.org/schema/beans/spring-beans.xsd">

    <sec:http pattern="/api/v1/todos/**" create-session="stateless"
                  entry-point-ref="oauth2AuthenticationEntryPoint"> <!-- (1) -->
        <sec:access-denied-handler ref="oauth2AccessDeniedHandler"/> <!-- (2) -->
        <sec:csrf disabled="true"/> <!-- (3) -->
        <sec:custom-filter ref="oauth2AuthenticationFilter"
                           before="PRE_AUTH_FILTER" /> <!-- (4) -->
    </sec:http>

    <bean id="oauth2AccessDeniedHandler"
              class="org.springframework.security.oauth2.provider.error.
↪OAuth2AccessDeniedHandler" /> <!-- (5) -->

    <bean id="oauth2AuthenticationEntryPoint"
              class="org.springframework.security.oauth2.provider.error.
↪OAuth2AuthenticationEntryPoint" /> <!-- (6) -->

    <oauth2:resource-server id="oauth2AuthenticationFilter" resource-id=
↪"todoResource"
```

```
            token-services-ref="tokenServices" entry-point-ref=
↪"oauth2AuthenticationEntryPoint" /> <!-- (7) -->

</beans>
```

Sr. No.	Description
(1)	Specify the pattern on path that is the target of authorization setting of OAuth 2.0 in `pattern` attribute. Specify the Bean of `OAuth2AuthenticationEntryPoint` in `entry-point-ref`. Settings are required for definition, however the specified Bean is not used. Actually, the Bean of `OAuth2AuthenticationEntryPoint` specified in `OAuth2AuthenticationProcessingFilter` described later is used.
(2)	Set the Bean of `OAuth2AccessDeniedHandler` in `access-denied-handler`. Here, `oauth2AccessDeniedHandler` described later is specified.
(3)	Disable the CSRF since error occurs if the POST, PUT, DELETE requests are received by CSRF token check.
(4)	Specify the authentication filter for resource server in `custom-filter`. Here `oauth2AuthenticationFilter` described later is specified. `OAuth2AuthenticationProcessingFilter` is set as per this specification. Since `OAuth2AuthenticationProcessingFilter` is a filter for performing Pre-Authentication by using access token included in request, Set in such a way that `OAuth2AuthenticationProcessingFilter` process is executed before `PRE_AUTH_FILTER` by specifying `PRE_AUTH_FILTER` in `before`. For Pre-Authentication, refer http://docs.spring.io/spring-security/site/docs/3.0.x/reference/preauth.html.

次のページに続く

Table. 9.65 – 前のページからの続き

Sr. No.	Description
(5)	Define `AccessDeniedHandler` for resource server provided by Spring Security OAuth. `OAuth2AccessDeniedHandler` sends the error response by handling the exceptions that occur at the time of authorization error.
(6)	Define the Bean of `OAuth2AuthenticationEntryPoint` for sending the error response for OAuth.
(7)	Define the ServletFilter for the resource server provided by Spring Security OAuth. The string specified in `id` attribute is the Bean ID. Here,‘‘oauth2AuthenticationFilter‘‘ is specified. Specify the resource ID provided by server in `resource-id` attribute. Here, `todoResource` is specified. It is verified whether the resource ID specified in `resource-id` attribute is included for the resource ID of client information linked to access token. As a result of verification, it is allowed to access the resource only when resource ID is included. Note that, definition of `resource-id` is optional and when it is not defined, resource ID is not verified. Specify the ID of `TokenServices` in `token-services-ref` attribute. `TokenServices` is described later. Specify the Bean of `OAuth2AuthenticationEntryPoint` in `entry-point-reff` attribute. Here, `oauth2AuthenticationEntryPoint` is specified.

Add the settings in `web.xml` so as to read a created `oauth2-resource.xml`.

- `web.xml`

```
<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>
        classpath*:META-INF/spring/applicationContext.xml
        classpath*:META-INF/spring/oauth2-resource.xml <!-- (1) -->
        classpath*:META-INF/spring/spring-security.xml
    </param-value>
```

```
</context-param>
```

Sr. No.	Description
(1)	First read oauth2-resource.xml considering that the path including the path pattern set in oauth2-resource.xml is set as the access control target in `spring-security.xml`.

Setting of accessible scope for the resource

In order to define the accessible scope for each resource, add the scope definition and Bean definition for supporting SpEL expression in Bean definition file for OAuth 2.0.

Implementation example is as given below. Note that, Basic authentication is not performed for the Client in the settings of access control since it is assumed that the security between authorization server and resource is secured.

- `oauth2-resource.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:oauth2="http://www.springframework.org/schema/security/oauth2"
       xmlns:sec="http://www.springframework.org/schema/security"
       xsi:schemaLocation="http://www.springframework.org/schema/security
           http://www.springframework.org/schema/security/spring-security.xsd
           http://www.springframework.org/schema/security/oauth2
           http://www.springframework.org/schema/security/spring-security-oauth2.
↪xsd
           http://www.springframework.org/schema/beans
           http://www.springframework.org/schema/beans/spring-beans.xsd">

    <sec:http pattern="/api/v1/todos/**" create-session="stateless"
                  entry-point-ref="oauth2AuthenticationEntryPoint">
        <sec:access-denied-handler ref="oauth2AccessDeniedHandler"/>
        <sec:csrf disabled="true"/>
        <sec:expression-handler ref="oauth2ExpressionHandler"/> <!-- (1) -->
        <sec:intercept-url pattern="/**" method="GET"
                           access="#oauth2.hasScope('READ')" /> <!-- (2) -->
        <sec:intercept-url pattern="/**" method="POST"
                           access="#oauth2.hasScope('CREATE')" /> <!-- (2) -->
        <sec:custom-filter ref="oauth2ProviderFilter"
                           before="PRE_AUTH_FILTER" />
    </sec:http>
```

```
    <!-- omitted -->

    <oauth2:web-expression-handler id="oauth2ExpressionHandler" /> <!-- (3) -->

</beans>
```

Sr. No.	Description
(1)	Specify a Bean of `OAuth2WebSecurityExpressionHandler` in `expression-handler`.
(2)	Define access policy as per the scope for the resource by using `intercept-url`. Specify the path pattern of resource to be retained in `pattern` attribute. In this implementation example, resource is retained under /api/v1/todos/. Specify the HTTP method of resource in `method` attribute. Specify the scope that authorizes the access to resource in `access` attribute. Differentiate the setting value by upper-case and lower case characters. Here, it is specified by using SpEL expression.
(3)	Define the Bean of `OAuth2WebSecurityExpressionHandler`. SpeL for performing OAuth 2.0 authorization setting provided by Spring Security OAuth is supported by defining this Bean. Note that, the value specified in `id` attribute is the id for this bean.

The main Expression provided by Spring Security OAuth is introduced.

For details, refer JavaDoc of `OAuth2SecurityExpressionMethods`.

Table.9.66 **Expression provided by Spring Security OAuth**

Expression	Description
`clientHasRole(String role)`	Return `true` when the client performs the role specified in the argument.
`clientHasAnyRole(String... roles)`	Return `true` when the client performs one of the roles specified in the argument.
`hasScope(String scope)`	Return `true` when the scope authorized by the resource owner and the scope of the argument matches in the client.
`hasAnyScope(String... scopes)`	Return `true` when the scope authorized by the resource owner and one of the scopes of the argument matches in the client.
`hasScopeMatching(String scopeRegex)`	Return `true` when the scope authorized by the resource owner and the regular expression specified in the argument matches in the client.
`hasAnyScopeMatching(String... scopesRegex)`	Return `true` when the scope authorized by the resource owner and one of the regular expressions specified in the argument matches in the client.
`denyOAuthClient`	Deny request in OAuth 2.0. It is used so that only the resource owner can access the resources.
`isOAuth`	Allow request in OAuth 2.0. It is used so that the client can access the resources.

Note: SpEL expression can be used with SpEL provided by Spring Security.

Refer to :ref:*built-incommon-expressions*for SpEL provided by Spring Security.

Configuration of access token (Resource Server)

Authorization Server is linked with Resource Server via access token `TokenServices`.

Refer to *How to share access token with resource server* for the methods of linking.

Apply settings by the method of sharing database.

Refer to *How to share access token with resource server* for the description of settings as the settings are same as `TokenServices` settings in authorization server.

- `oauth2-resource.xml`

```
<bean id="tokenServices"
    class="org.springframework.security.oauth2.provider.token.DefaultTokenServices
↪">
    <property name="tokenStore" ref="tokenStore" />
</bean>

<bean id="tokenStore"
  class="org.springframework.security.oauth2.provider.token.store.JdbcTokenStore">
  <constructor-arg ref="tokenDataSource" />
</bean>
```

Note: A method which use `RemoteTokenServices` provided by Spring Security OAuth and a method which use JSON Web Token are offered as the methods for linking the authorization server and resource server. Refer to *Linkage through HTTP access* for how to use `RemoteTokenServices` provided by Spring Security OAuth.

Fetching user information

In the Resource Server, the authenticated resource owner information can be received by specifying `UserDetails` in method argument of Controller class and assigning `@AuthenticationPrincipal` an-

notation similar to fetching method of the authentication information explained in *Coordination between authentication process and Spring MVC*. The implementation example is shown below.

```
@RestController
@RequestMapping("api")
public class TodoRestController {

    // omitted

    @RequestMapping(value = "todos", method = RequestMethod.GET)
    @ResponseStatus(HttpStatus.OK)
    public Collection<Todo> list(@AuthenticationPrincipal UserDetails user) { //␣
↪(1)

        // omitted

    }
}
```

Sr.No.	Description
(1)	Authentication information of resource owner is stored in the argument ‘‘user‘‘

In case of implementation that specifies `UserDetails`, it should be noted that the intended information cannot be fetched in the client credential grant wherein authentication process is not carried out. When client credential grant is used, client ID can be fetched by specifying `String` as the method argument of Controller and assigning `@AuthenticationPrincipal` annotation.

Further, in case of implementation that specifies `UserDetails`, authentication information of client such as Client ID cannot be fetched. When client authentication information is to be fetched, “OAuth2Authentication‘‘ can be specified as the method argument of the Controller class. An example wherein ‘OAuth2Authentication‘‘ is specified in the method argument of Controller class and authentication information of the client and the resource owner is fetched, is given below.

```
@RestController
@RequestMapping("api")
public class TodoRestController {

    // omitted

    @RequestMapping(value = "todos", method = RequestMethod.GET)
    @ResponseStatus(HttpStatus.OK)
    public Collection<Todo> list(OAuth2Authentication authentication) { // (1)

        String username = authentication.getUserAuthentication().getName(); // (2)
        String clientId = authentication.getOAuth2Request().getClientId();  // (3)
```

```
        // omitted

    }
}
```

Sr.No.	Description
(1)	Authentication information of the resource owner and the client is stored in the argument `authentication`.
(2)	Fetch resource owner name from `authentication`.
(3)	Fetch client ID from `authentication`.

Note: The implementation example is given above wherein `OAuth2Authentication` is used in the application layer. When the implementation is to be done without depending on `OAuth2Authentication`, the same function is feasible by implementing `HandlerMethodArgumentResolver`. Refer to *Implementing HandlerMethodArgumentResolver* for specific implementation method.

Implementation of client

How to implement client is broadly classified into the following 2 methods based on the grant type to be used.

How to implement	Grant type
OAuth2RestTemplate	Authorization code grant Resource owner password credential grant Client credential grant
Javascript	Implicit grant

In this guideline, *Accessing a resource using OAuth2RestTemplate* and *Accessing a resource using JavaScript*are described respectively in classification described above.

Accessing a resource using OAuth2RestTemplate

How to access resource using `OAuth2RestTemplate` is described as the implementation of the client for authorization code grant, resource owner password credential grant and client credential grant.

In Spring Security OAuth, `OAuth2RestTemplate` is provided as implementation for OAuth 2.0 of `RestOperations`.

In `OAuth2RestTemplate`, the functions for fetching access token, sharing access token between multiple requests by `OAuth2ClientContext` and error handling when accessing the Resource Server are implemented as per the grant type by `AccessTokenProvider` as independent functions of OAuth 2.0.

In the client, the resource can be accessed using OAuth 2.0 functions by defining the parameters as per the application requirements such as grant type, scope using `OAuth2RestTemplate`.

Creation of configuration file (Client)

First create `oauth2-client.xml` as the configuration file for defining OAuth 2.0.

- `oauth2-client.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:sec="http://www.
↪springframework.org/schema/security"
    xmlns:oauth2="http://www.springframework.org/schema/security/oauth2"
    xsi:schemaLocation="
        http://www.springframework.org/schema/security http://www.springframework.
↪org/schema/security/spring-security.xsd
        http://www.springframework.org/schema/beans http://www.springframework.org/
↪schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/security/oauth2 http://www.
↪springframework.org/schema/security/spring-security-oauth2.xsd
    ">
```

```
</beans>
```

Add the settings to `web.xml` so as to read the created `oauth2-client.xml`.

- `web.xml`

```
<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>
        classpath*:META-INF/spring/applicationContext.xml
        classpath*:META-INF/spring/oauth2-client.xml  <!-- (1) -->
        classpath*:META-INF/spring/spring-security.xml
    </param-value>
</context-param>

<!-- omitted -->
```

Sr.No.	Description
(1)	Set so as to read `oauth2-client.xml`.

Applying OAuth2ClientContextFilter

Register `OAuth2ClientContextFilter` as a servlet filter.

By registering `OAuth2ClientContextFilter`, when an attempt is made to access the Resource Server when authorization cannot be fetched by the resource owner, a function to capture `UserRedirectRequiredException` and redirect to the page for fetching authorization of the resource owner provided by the Authorization Server can be incorporated.

Add the following settings to `oauth2-client.xml`.

- `oauth2-client.xml`

```
<oauth2:client id="oauth2ClientContextFilter" /> <!-- (1) -->
```

Sr.No.	Explanation
(1)	Bean is defined for `OAuth2ClientContextFilter`by using `<oauth2:client>`tag. The string specified in `id`attribute is used as Bean ID.

Add the settings of `OAuth2ClientContextFilter`to `web.xml`.

- `web.xml`

```
<filter> <!-- (1) -->
    <filter-name>oauth2ClientContextFilter</filter-name>
    <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-
↪class>
</filter>
<filter-mapping>
    <filter-name>oauth2ClientContextFilter</filter-name>
    <url-pattern>/*</url-pattern> <!-- (2) -->
</filter-mapping>
```

Sr.No.	Description
(1)	Register the Bean wherein Bean ID matches with the filter name (value specified in the `<filter-name>` element) as a servlet filter by using `DelegatingFilterProxy`. Set the value same as the Bean ID specified in `id` attribute of `<oauth2:client>` in the filter name. It is recommended to mention `OAuth2ClientContextFilter` at the end of servlet filter definition so as not to perform unintended exception handling of the exception generated in Spring Security OAuth.
(2)	‘‘OAuth2ClientContextFilter‘‘ is applied to the path wherein ‘‘UserRedirectRequiredException‘‘ may occur. In the above example, “OAuth2ClientContextFilter‘‘ is applied to all the requests.

Note: Since `OAuth2ClientContextFilter` generates redirect URL for returning the user agent after the authorization process by the Authorization Server in the pre-process of the filter, a wide definition such as `/*` will

result in futile processing in the path where `UserRedirectRequiredException` does not occur.

Note: ‘‘OAuth2ClientContextFilter‘‘ stores the URL to be redirected after the Authorization Server gets the authorization from the resource owner with the attribute name `currentUri` in the request scope. Therefore, the attribute name `currentUri` cannot be used in the client.

As mentioned above, ‘‘Oauth2ClientContextFilter‘‘ is a servlet filter that captures ‘‘UserRedirectRequiredException‘‘ and redirects it to the Authorization Server.

However, if the ‘‘SystemExceptionResolver‘‘ set in the blank project in advance, handles ‘‘UserRedirectRequiredException‘‘ first, ‘‘Oauth2ClientContextFilter‘‘ will not work as intended.

Therefore, it is necessary to change the setting of *spring-mvc.xml‘* so that the ‘‘SystemExceptionResolver‘‘ does not handle ‘‘UserRedirectRequiredException‘‘ . Refer to *Exception Handling* for the description of `SystemExceptionResolver`in detail.

Add `UserRedirectRequiredException` to `spring-mvc.xml` as the exclusion target of `SystemExceptionResolver`.

- `spring-mvc.xml`

```
<bean id="systemExceptionResolver"
    class="org.terasoluna.gfw.web.exception.SystemExceptionResolver">

    <!-- omitted -->

    <property name="excludedExceptions">
        <array>
            <!-- (1) -->
            <value>org.springframework.security.oauth2.client.resource.
↪UserRedirectRequiredException
            </value>
        </array>
    </property>
</bean>
```

Sr.No.	Description
(1)	Exclude `UserRedirectRequiredException` from handling target of `SystemExceptionResolver`.

Settings of OAuth2RestTemplate

How to set `OAuth2RestTemplate` for each grant type is explained.

Setting a resource while using authorization code grant

The setting example of `OAuth2RestTemplate` is shown while accessing with authorization code grant.

- `oauth2-client.xml`

```
<oauth2:resource id="todoAuthCodeGrantResource" client-id="firstSec"
                 client-secret="firstSecSecret"
                 type="authorization_code"
                 scope="READ,WRITE"
                 access-token-uri="${auth.serverUrl}/oth2/token"
                 user-authorization-uri="${auth.serverUrl}/oth2/authorize"/> <!--␣
↪(1) -->

<oauth2:rest-template id="todoAuthCodeGrantResourceRestTemplate" resource=
↪"todoAuthCodeGrantResource" /> <!-- (2) -->
```

Sr.No.	Description
(1)	Define detailed information of the resource to be accessed , to be referred by `OAuth2RestTemplate`. Refer to the following table for the setting value of each item.
(2)	Define `OAuth2RestTemplate`. Specify Bean ID of `OAuth2RestTemplate` in `id`. Specify `id` of Bean defined in (1) for `resource`.

Table.9.67 **Resource detailed information**

Items	Description
`id`	Bean ID of resource.
`client-id`	ID for identifying the client in the Authorization Server.
`client-secret`	Password used for authentication of client in the Authorization Server.
`type`	Grant type. Specify `authorization_code` in case of authorization code grant.
`scope`	Enumerate the scope that requires authorization by separating with commas. Setting value is case-sensitive. Request all scopes set for the client in Authorization Server at the time of omitting.
`access-token-uri`	URL of endpoint of the Authorization Server for requesting the issue of access token.
`user-authorization-uri`	URL of endpoint of the Authorization Server for getting the authorization of resource owner.

Note: In `<oauth2:resource>`tag, `client-authentication-scheme` parameter is provided as the method to specify client authentication method at the time of getting access token. The values that can be specified in `client-authentication-scheme` parameter are as follows.

- `header`: Basic authentication using Authorization header. Default value.
- `query` : Authentication using URL query parameter at the time of requesting.
- `form` : Authentication using body parameter at the time of requesting.

In this guideline, since Basic authentication is used for authenticating the client, parameters are not specified in the above-mentioned setting example, however, parameters should be specified as per the application requirements.

```
<oauth2:resource id="todoAuthCodeGrantResource" client-id="firstSec"
                 client-secret="firstSecSecret"
                 type="authorization_code"
                 scope="READ,WRITE"
                 access-token-uri="${auth.serverUrl}/oth2/token"
                 user-authorization-uri="${auth.serverUrl}/oth2/authorize"
                 client-authentication-scheme="form" />
```

Settings of resource at the time of using resource owner password credential grant

In the resource owner password credential grant, the client requests the issue of access token by using the user name and password of the resource owner.

There is a need to set the username and password of resource owner as separate parameters in `OAuth2RestTemplate`, however, when multiple resource owners use the same client, there is a need to consider switching the setting contents for each resource owner.

The method to implement switching of setting contents for each resource owner by setting resource of `OAuth2RestTemplate` in the Bean of Session scope and storing the information of resource owner in that Bean is explained here.

The setting example of `OAuth2RestTemplate` when accessing with resource owner password credential grant, is shown.

Define `ResourceOwnerPasswordResourceDetails` in the Session scope and set to `OAuth2RestTemplate` to enable switching of setting contents of each resource owner.

- `oauth2-client.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:sec="http://www.
↪springframework.org/schema/security"
    xmlns:oauth2="http://www.springframework.org/schema/security/oauth2"
    xmlns:aop="http://www.springframework.org/schema/aop"
    xsi:schemaLocation="
        http://www.springframework.org/schema/security http://www.springframework.
↪org/schema/security/spring-security.xsd
```

```
        http://www.springframework.org/schema/beans http://www.springframework.org/
↪schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/security/oauth2 http://www.
↪springframework.org/schema/security/spring-security-oauth2.xsd
        http://www.springframework.org/schema/aop http://www.springframework.org/
↪schema/aop/spring-aop.xsd
    ">

    <!-- omitted -->

    <bean id="todoPasswordGrantResource" class="org.springframework.security.
↪oauth2.client.token.grant.password.ResourceOwnerPasswordResourceDetails"
        scope="session">
        <aop:scoped-proxy>
        <property name="clientId" value="firstSec" />
        <property name="clientSecret" value="firstSecSecret" />
        <property name="accessTokenUri" value="${auth.serverUrl}/oth2/token" />
        <property name="scope" value="READ,WRITE" />
    </bean> <!-- (1) -->

    <oauth2:rest-template id="todoPasswordGrantResourceRestTemplate" resource=
↪"todoPasswordGrantResource" /> <!-- (2) -->
```

Sr. No.	Description
(1)	Define detailed information of the resource to be accessed, to be referred by ‘‘OAuth2RestTemplate‘‘. Refer to the following table for the setting value of each item.
(2)	Define ‘‘OAuth2RestTemplate‘‘. Specify Bean ID of `OAuth2RestTemplate` in `id`. Specify ‘‘id‘‘ of Bean defined in (1) in `resource`.

Item	Description
`class`	Specify the Bean that acts as a resource of ‘‘OAuth2RestTemplate‘‘ . Specify ‘‘ResourceOwnerPasswordResourceDetails‘‘ here.
`scope`	Specify session and consider the scope range as HTTPSession.
`<aop:scoped-proxy>`	Set Bean of Session scope for injecting it in `OAuth2RestTemplate` that is Bean of Singleton. This is a setting that is necessary because the lifecycle of bean of Singleton is longer than the Bean of Session scope. In order to use this tag, namespace and schema of “‘ aop‘‘ is added.
`clientId` property	Set ID for identifying the client in the Authorization Server for `clientId` of Bean.
`clientSecret` property	Set password used for client authentication in the Authorization Server for `clientSecrett` of Bean.
`accessTokenUri` property	Specify URL of endpoint of the Authorization Server for requesting the issue of access token.
`scope` property	Set scope list requesting authorization for `scope` of Bean.

Note: It is assumed that the username and password of resource owner is fetched after it is entered by the resource owner on the client screen when access is required, and is stored in Bean. In this guideline, the explanation of how

to fetch username and password is omitted.

Warning: In the Authorization Server described in this guideline, the password set in ‘‘ ResourceOwnerPasswordResourceDetails‘‘ must be in plain text in order to perform comparative verification after hashing the password used for authentication. Since there is high risk in handling the password of resource owner in plain text by the client, the resource owner password credential grant should be used in very limited cases such as when there is highly trustworthy relationship between client and resource owner and the client is kept in secure environment. Refer to *Client authentication* for the settings of hashing in Authorization Server.

Settings of resource at the time of using client credential grant

The setting example of `OAuth2RestTemplate` is shown while accessing with client credential grant.

The explanation of settings common to authorization code grant is omitted.

- `oauth2-client.xml`

```
<oauth2:resource id="todoClientGrantResource" client-id="firstSecClient"
                 client-secret="firstSecSecret"
                 type="client_credentials"
                 access-token-uri="${auth.serverUrl}/oth2/token" /> <!-- (1) -->

<oauth2:rest-template resource=id="todoClientGrantResourceRestTemplate" resource=
↪"todoClientGrantResource" />
```

Sr.No.	Description
(1)	Specify the grant type in `type` attribute. Specify `client_credentials` in case of client credential grant.

Accessing a resource Server

How to access the Resource Server using `OAuth2RestTemplate` is explained.

Since access to Authorization Server is hidden by `OAuth2RestTemplate` and `OAuth2ClientContextFilter`, the process to be performed for REST API provided by the Resource

Server is described similar to normal access to REST API without being aware of the Authorization Server at the time of development.

The implementation example of Service class is shown below.

```
import org.springframework.web.client.RestOperations;

@Service
public class TodoServiceImpl implements TodoService {

    @Inject
    @Named("todoAuthCodeGrantResourceRestTemplate")
    RestOperations restOperations; // (1)

    @Value("${resource.serverUrl}/api/v1/todos")
    String uri;

    @Override
    public List<Todo> getTodoList() {
        Todo[] todoArray = restOperations.
            getForObject(url, Todo[].class); // (2)
        return Arrays.asList(todoArray);
    }
}
```

Sr. No.	Description
(1)	Inject `RestOperations`. In the above-mentioned example, the Bean whose Bean ID is `todoAuthCodeGrantResourceRestTemplate` is Injected. `@Named` should be specified when multiple `OAuth2RestTemplate` are defined.
(2)	Access the specified URL with GET method in REST and receive the result in a list.

Note: When access token is not issued if `OAuth2RestTemplate` is used to access the Resource Server, it is redirected to the Authorization Server at once. Thereafter, when the issue of access token is complete, it is redirected to the client and access process to the resource server is called again. At this time, since redirect to client is performed by GET, the Controller that may be redirected from Authorization Server, must allow GET.

If access to the Resource Server before the redirect is POST, the parameter will be lost in GET after redirect. In that case, a measure like retaining POST parameter in session, should be taken. In this guideline, the explanation of specific handling method is omitted.

Accessing a resource using JavaScript

In the implicit grant, user agent such as Web browser requests authorization for the Authorization Server.

Therefore, implicit grant is usually used in JavaScript that runs on Web browser, however since JavaScript library is not provided in Spring Security OAuth, client should be implemented independently when using implicit grant.

In this guideline, how to fetch data in JSON format from the Resource Server using JavaScript and display it on the screen is explained as the implementation of client for implicit grant.

Note: Use jQuery as JavaScript library in the implementation example to be explained later. It is assumed to be stored under `src/main/webapp` with js file.

API example wherein OAuth 2.0 function is implemented independently, is shown.

- `oauth2.js`

```
var oauth2Func = (function(exp, $) {
    "use strict";

    var
        config = {},
        DEFAULT_LIFETIME = 3600;

    var uuid = function() {
        return "xxxxxxxx-xxxx-4xxx-yxxx-xxxxxxxxxxxx".replace(/[xy]/g, function(c)
↪{
            var r = Math.random()*16|0, v = c == "x" ? r : (r&0x3|0x8);
            return v.toString(16);
        });
    };

    var encodeURL= function(url, params) {
        var res = url;
        var k, i = 0;
        for(k in params) {
            res += (i++ === 0 ? "?" : "&") + encodeURIComponent(k) + "=" +␣
↪encodeURIComponent(params[k]);
        }
        return res;
    };

    var epoch = function() {
        return Math.round(new Date().getTime()/1000.0);
```

```
};

var parseQueryString = function (qs) {
    var e,
        a = /\+/g,
        r = /([^&;=]+)=?([^&;]*)/g,
        d = function (s) { return decodeURIComponent(s.replace(a, " ")); },
        q = qs,
        urlParams = {};

    while (e = r.exec(q)) {
       urlParams[d(e[1])] = d(e[2]);
    }

    return urlParams;
};

var saveState = function(state, obj) {
    localStorage.setItem("state-" + state, JSON.stringify(obj));
};

var getState = function(state) {
    var obj = JSON.parse(localStorage.getItem("state-" + state));
    localStorage.removeItem("state-" + state);
    return obj;
};

var hasScope = function(token, scope) {
    if (!token.scopes) return false;
    var i;
    for(i = 0; i < token.scopes.length; i++) {
        if (token.scopes[i] === scope) return true;
    }
    return false;
};

var filterTokens = function(tokens, scopes) { // (1)
    if (!scopes) scopes = [];

    var i, j,
    result = [],
    now = epoch(),
    usethis;
    for(i = 0; i < tokens.length; i++) {
        usethis = true;

        if (tokens[i].expires && tokens[i].expires < (now+1)) usethis = false;

        for(j = 0; j < scopes.length; j++) {
            if (!hasScope(tokens[i], scopes[j])) usethis = false;
```

```
            }

            if (usethis) result.push(tokens[i]);
        }
        return result;
    };

    var saveTokens = function(provider, tokens) {
        localStorage.setItem("tokens-" + provider, JSON.stringify(tokens));
    };

    var getTokens = function(provider) {
        var tokens = JSON.parse(localStorage.getItem("tokens-" + provider));
        if (!tokens) tokens = [];

        return tokens;
    };

    var wipeTokens = function(provider) {
        localStorage.removeItem("tokens-" + provider);
    };

    var saveToken = function(provider, token) {
        var tokens = getTokens(provider);
        tokens = filterTokens(tokens);
        tokens.push(token);
        saveTokens(provider, tokens);
    };

    var getToken = function(provider, scopes) {
        var tokens = getTokens(provider);
        tokens = filterTokens(tokens, scopes);
        if (tokens.length < 1) return null;
        return tokens[0];
    };

    var sendAuthRequest = function(providerId, scopes) { // (2)
        if (!config[providerId]) throw "Could not find configuration for provider
↪" + providerId;
        var co = config[providerId];

        var state = uuid();
        var request = {
            "response_type": "token"
        };
        request.state = state;

        if (co["redirectUrl"]) {
            request["redirect_uri"] = co["redirectUrl"];
        }
```

```
        if (co["clientId"]) {
            request["client_id"] = co["clientId"];
        }
        if (scopes) {
            request["scope"] = scopes.join(" ");
        }

        var authurl = encodeURL(co.authorization, request);

        if (window.location.hash) {
            request["restoreHash"] = window.location.hash;
        }
        request["providerId"] = providerId;
        if (scopes) {
            request["scopes"] = scopes;
        }

        saveState(state, request);
        redirect(authurl);

    };

    var checkForToken = function(providerId) { // (3)
        var h = window.location.hash;

        if (h.length < 2) return true;

        if (h.indexOf("error") > 0) { // (4)
            h = h.substring(1);
            var errorinfo = parseQueryString(h);
            handleError(providerId, errorinfo);
            return false;
        }

        if (h.indexOf("access_token") === -1) {
            return true;
        }
        h = h.substring(1);
        var atoken = parseQueryString(h);

        if (!atoken.state) {
            return true;
        }

        var state = getState(atoken.state);
        if (!state) throw "Could not retrieve state";
        if (!state.providerId) throw "Could not get providerId from state";
        if (!config[state.providerId]) throw "Could not retrieve config for this␣
↪provider.";
```

```
        var now = epoch();
        if (atoken["expires_in"]) {
            atoken["expires"] = now + parseInt(atoken["expires_in"]);
        } else {
            atoken["expires"] = now + DEFAULT_LIFETIME;
        }

        if (atoken["scope"]) {
            atoken["scopes"] = atoken["scope"].split(" ");
        } else if (state["scopes"]) {
            atoken["scopes"] = state["scopes"];
        }

        saveToken(state.providerId, atoken);

        if (state.restoreHash) {
            window.location.hash = state.restoreHash;
        } else {
            window.location.hash = "";
        }
        return true;
    };

    var handleError = function(providerId, cause) { // (5)
        if (!config[providerId]) throw "Could not retrieve config for this␣
↪provider.";

        var co = config[providerId];
        var errorDetail = cause["error"];

        // redirect error page
        if(co["errRedirectUrl"]) {
            redirect(co["errRedirectUrl"] + "/" + errorDetail);
        } else {
            alert("Access Error. cause: " + errorDetail);
        }
    };


    var redirect = function(url) {
        window.location = url;
    };

    var initialize = function(c) {
        config = c;
        try {
            var key, providerId;
            for(key in c) {
                    providerId = key;
            }
```

```
            return checkForToken(providerId);
        } catch(e) {
            console.log("Error when retrieving token from hash: " + e);
            window.location.hash = "";
            return false;
        }
    };

    var clearTokens = function() {
        var key;
        for(key in config) {
            wipeTokens(key);
        }
    };

    var oajax = function(settings) { // (6)
        var providerId = settings.providerId;
        var scopes = settings.scopes;
        var token = getToken(providerId, scopes);

        if (!token) {
            sendAuthRequest(providerId, scopes);
            return;
        }

        if (!settings.headers) settings.headers = {};
        settings.headers["Authorization"] = "Bearer " + token["access_token"];

        $.ajax(settings);
    };

    return {
        initialize: function(config) {
            return initialize(config);
        },
        clearTokens: function() {
            return clearTokens();
        },
        oajax: function(settings) {
            return oajax(settings);
        }
    };

})(window, jQuery);
```

Sr.No.	Description
(1)	Function to check the expiry and scope availability of access token. The client returns the available access token.
(2)	Function to request authorization of the Authorization Server. Fetch the required parameters from the configuration information and create a request.
(3)	Function to get access token from authorization response. When access token can be fetched, store the information in the local storage.
(4)	As the result of authorization, when an error is returned, call `handleError` as error handling.
(5)	Function to handle authorization error. In this guideline, redirect to redirect destination URL is carried out at the time of error specified in configuration information as the implementation example of error handling.
(6)	Function to request access resource to the Resource Server. Fetch access token from local storage and send request to the Resource Server using ajax function of jQuery.

The implementation example of client is shown below.

- `todoList.jsp`

```
<script type="text/javascript" src="${pageContext.request.contextPath}/resources/
↪vendor/jquery/jquery.js"></script> <!-- (1) -->
<script type="text/javascript" src="${pageContext.request.contextPath}/resources/
↪app/js/oth2-implicit.js"></script> <!-- (1) -->
<script type="text/javascript">
```

```
"use strict";

$(document).ready(function() {
    var result = oauth2Func.initialize({ // (2)
        "todo" : { // (3)
            clientId : "client", // (4)
            redirectUrl : "${client.serverUrl}/oth2/api", // (5)
            errRedirectUrl : "${client.serverUrl}/oth2/error", // (6)
            authorization : "${auth.serverUrl}/oth2/authorize" // (7)
        }
    });

    if (result) {
        oauth2Func.oajax({ // (8)
            url : "${resource.serverUrl}/api/v1/todos", // (9)
            providerId : "todo",  // (10)
            scopes : [ "READ" ],  // (11)
            dataType : "json",    // (12)
            type : "GET",  // (13)
            success : function(data) { // (14)
                $("#message").text(JSON.stringify(data));
            },
            error : function() {
                oauth2Func.clearTokens();
            }
        });
    } else {
        oauth2Func.clearTokens(); // (15)
    }


};

</script>
<div id="wrapper">
    <p id="message"></p>
</div>
```

Sr. No.	Description
(1)	As described later, specify the path storing js file, jQuery for which OAuth 2.0 function is implemented individually.
(2)	Define configuration information to be used for authorization request and initialize the same.

次のページに続く

Table. 9.68 – 前のページからの続き

Sr. No.	Description
(3)	Specify a unique value as an identifier for distinguishing the configuration information of each client. In the process of accessing resources described later, manage and fetch configuration information by using this item as a key.
(4)	Specify ID identifying the client.
(5)	Specify URL to redirect the client after authenticating the resource owner of authorization server. In this implementation example, it is assumed that parameters of URL are received from the controller.
(6)	Specify URL for redirecting when an error is received from authorization server as the authorization response. In this implementation example, it is assumed that parameters of URL are received from the controller. These guidelines show the method of redirecting to client screen and displaying the error screen, as an example of implementation when error is received. Description about client Controller is omitted.
(7)	Specify authorization end point of authorization server. In this implementation example, it is assumed that parameters of URL are received from the controller.
(8)	Access the resource.
(9)	Specify access destination of resource server.
(10)	Specify identifier of configuration information to be referred.

次のページに続く

Table. 9.68 – 前のページからの続き

Sr. No.	Description
(11)	Specify scope to be requested to the resource. Setting value is classified by upper case character and lower chase character.
(12)	Specify response type.
(13)	Access resource server GET method.
(14)	Specify process when processing is successful. Response after successful processing is stored in `message`.
(15)	Clear the access token.

Todo

TBD

When authorization servers and resource server that are not in the same domain are accessed from the clients created in JavaScript, support of `Cross-Origin Resource Sharing` is required on authorization server and resource server.

Details will be described in the next versions.

Cancellation of token (Client server)

Implementation method of cancelling the issued token is described.

For cancelling the access token, request is made to authorization server and access token is deleted from `TokenStore`. Request header of Basic authentication is set in order to perform Basic authentication of client

when request is made to authorization server.

Refer to *Canceling a token (authorization server)* , for the cancellation of tokens in authorization server.

Access token retained in `OAuth2RestTemplate` should be deleted after requesting the cancellation of access token to authorization server by the client server.

Implementation example is given below.

At first, settings of `RestTemplate` are added in configuration file in order to request cancelation of token to authorization server.

- `oauth2-client.xml`

```
<!-- (1) -->
<bean id="revokeRestTemplate" class="org.springframework.web.client.RestTemplate">
    <property name="interceptors">
        <list>
            <ref bean="basicAuthInterceptor" />
        </list>
    </property>
</bean>

<bean id="basicAuthInterceptor" class="com.example.oauth2.client.restclient.
↪BasicAuthInterceptor" />
```

Sr.No.	Description
(1)	Define a Bean for `RestTemplate` in order to request the cancellation of token to authorization server. Implementation class `ClientHttpRequestInterceptor` is specified in `interceptors` property, in order to set request header for basic authentication. Refer to *Process to configure a request header for Basic authentication*, for the implementation details of `ClientHttpRequestInterceptor` implementation class that sets the request header of Basic authentication.

Interface of service class and implementation class are created for cancelling the token.

- `RevokeTokenClientService.java`

```
public interface RevokeTokenClientService {

    String revokeToken();

}
```

- RevokeTokenClientServiceImpl.java

```
@Service
public class RevokeTokenClientServiceImpl implements RevokeTokenClientService {

    @Value("${auth.serverUrl}/api/v1/oth2/tokens/revoke")
    String revokeTokenUrl; // (1)

    @Inject
    @Named("todoAuthCodeGrantResourceRestTemplate")
    OAuth2RestOperations oauth2RestOperations; // (2)

    @Inject
    @Named("revokeRestTemplate")
    RestOperations revokeRestOperations; // (3)

    @Override
    public String revokeToken() {

        String token = getTokenValue(oauth2RestOperations);
        HttpHeaders headers = new HttpHeaders();
        headers.setContentType(MediaType.APPLICATION_FORM_URLENCODED);
        MultiValueMap<String, String> variables = new LinkedMultiValueMap<String,␣
↪String>();
        variables.add("token", token);

        String result = revokeRestOperations.postForObject(revokeTokenUrl,
            new HttpEntity<MultiValueMap<String, String>>(variables, headers),
            String.class); // (4)
        // (5)
        if ("success".equals(result)) {
            initContextToken(oauth2RestOperations);
        }
        return result;
    }

    // (6)
    private String getTokenValue(OAuth2RestOperations oauth2RestOperations) {
        String tokenValue = "";
        OAuth2AccessToken token = oauth2RestOperations.getAccessToken();
        if (token != null) {
            tokenValue = token.getValue();
        }
        return tokenValue;
    }

    // (7)
    private void initContextToken(OAuth2RestOperations oauth2RestOperations) {
        oauth2RestOperations.getOAuth2ClientContext().setAccessToken(null);
    }
}
```

Sr.No.	Description
(1)	URL to be used while requesting the cancellation of access token to the authorization server.
(2)	Inject `OAuth2RestTemplate` which retains the access token to be cancelled.
(3)	Inject `RestTemplate` for cancelling the access token.
(4)	In order to cancel the access token in authorization server, it is accessed with method POST in REST. Set value of access token to be cancelled in the request parameter for passing it to the authorization server. Access token is fetched by passing `OAuth2RestOperations` to `getTokenValue` method defined in (5).
(5)	Process result of authorization server is determined and only when it is normal, the access token retained by ‘‘OAuth2RestOperations‘‘ is deleted. Access token is deleted by passing `OAuth2RestOperations` retaining the access token in `initContextToken` method defined in (6).
(6)	Method of getting the access token retained by `OAuth2RestOperations` . Access token is fetched and returned by calling the `getAccessToken` method of `OAuth2RestOperations` passed as a parameter.
(7)	Method of deleting the access token retained by `OAuth2RestOperations` . Delete the access token by passing the null in `setAccessToken` method of `OAuth2RestOperations` passed as a parameter.

Client cancels the token by calling the service created above when the access token is not required.

Since the default implementation of Spring Security OAuth retains the access token in scope of session, the token seems to be cancelled when client user logs-out or when session is revoked due to session timeout.

9.10.3 How to extend

Linking Authorization Server and Resource Server through endpoints

The Resource Server and Authorization Server can be linked by performing HTTP access from the Resource Server to the check token endpoint of the Authorization Server.

Check token endpoint is the endpoint that receives the value of access token from the Resource Server and verifies the token instead of the Resource Server. Check token endpoint fetches and verifies the access token using ‘‘TokenServices‘‘, and passes the information associated with the access token to the Resource Server when there is no problem in the verification.

Linkage through HTTP access

Use ‘‘org.springframework.security.oauth2.provider.token.RemoteTokenServices‘‘ that is the implementation class of ‘‘ TokenServices‘‘ for the Resource Server to access the check token endpoint of the Authorization Server.

‘‘RemoteTokenServices‘‘ performs HTTP access of check token endpoint using `org.springframework.web.client.RestTemplate` and fetches the information associated with the access token.

Since the access token is verified by the check token endpoint, it is not verified by `RemoteTokenServices`.

The implementation example is shown below.

First, perform the settings for registering `org.springframework.security.oauth2.provider.endpoint.CheckTo` as component for verifying token in the Authorization Server.

- `oauth2-auth.xml`

```
<sec:http pattern="/oth2/check-token" security="none" />  <!-- (1) -->

<oauth2:authorization-server
     client-details-service-ref="clientDetailsService"
     token-endpoint-url="/oth2/token"
     authorization-endpoint-url="/oth2/authorize"
     user-approval-handler-ref="userApprovalHandler"
     token-services-ref="tokenServices"
     check-token-enabled="true"
     check-token-endpoint-url="/oth2/check-token">  <!-- (2) -->
    <oauth2:authorization-code />
```

```
    <oauth2:implicit />
    <oauth2:refresh-token />
    <oauth2:client-credentials />
    <oauth2:password />
</oauth2:authorization-server>
```

Sr.No.	Description
(1)	In this guideline, it is assumed that access to the check token endpoint is by network configuration that allows only the Resource Server and is excluded from the security function (Security Filter) of Spring Security. It should be mentioned at the beginning of the configuration file so that it will be evaluated before any other security settings.
(2)	`CheckTokenEndpoint` is registered as the component by specifying `true` in `check-token-enabled` attribute of `<oauth2:authorization-server>tag`. Specify URL of `CheckTokenEndpoint` in `check-token-endpoint-url` attribute of `<oauth2:authorization-server>tag`. When it is not specified, the default value "/oauth/check_token" is specified.

Warning: Security measures of check token endpoint

- Check token endpoint is accessible only between Authorization Server and Resource Server so it cannot be accessed from open network. When disclosing the check token endpoint, it is necessary to take appropriate security measures.

Note: Note that when Spring Security OAuth 2.0.12 or earlier version is used and `authorization-endpoint-url` or `token-endpoint-url`is not specified in `check-token-endpoint-url`, it will not be reflected. It is addressed in the following issue and is expected to be fixed in Version 2.0.13.

https://github.com/spring-projects/spring-security-oauth/issues/897

Note: Use `DefaultTokenServices` same when linking `TokenServices` through shared DB. `TokenStore` referred by `TokenServices` uses the implementation class of the interface as per the appli-

cation requirements.

Perform the settings to use `org.springframework.security.oauth2.provider.token.RemoteTokenServices` as `TokenServices` in the configuration file of the Resource Server.

- `oauth2-resource.xml`

```
<bean id="tokenServices"
    class="org.springframework.security.oauth2.provider.token.RemoteTokenServices">
    <property name="checkTokenEndpointUrl" value="${auth.serverUrl}/oth2/check-
↪token" />  <!-- (1) -->
</bean>
```

Sr. No.	Description
(1)	Define Bean for `RemoteTokenServices` to enable fetching of information associated with the access token by accessing the check token endpoint of the Authorization Server. Set the URL in `checkTokenEndpointUrl` property to access check token endpoint.

Note: When verification error of access token occurs in the check token endpoint, HTTP status code 400(Bad Request) is returned in `RemoteTokenServices`. When HTTP status code 400(Bad Request) is returned in the default implementation of `RestTemplate`, error handling is done and client error exception is generated. `RestTemplate`to be used by `RemoteTokenServices` by default, is extended so that error handling is not performed when HTTP status code of response is 400 in order to link verification error of the access token to the client server. When `RestTemplate` is injected in `RemoteTokenServices`, note that this extension will not be applied.

When `RemoteTokenServices` is used in the Resource Server, the username of the resource owner can be fetched by specifying `@AuthenticationPrincipal` annotation in the handler method as the argument annotation in `String`.

The implementation example is as follows.

```
@RestController
@RequestMapping("api")
public class TodoRestController {

    // omitted

    @RequestMapping(value = "todos", method = RequestMethod.GET)
    @ResponseStatus(HttpStatus.OK)
    public Collection<Todo> list(@AuthenticationPrincipal String userName) { // (1)

        // omitted

    }
}
```

Sr.No.	Description
(1)	Username of resource owner is stored in argument `userName`.

Extension of DefaultAccessTokenConverter

Even in case of the structure where DB is not shared between Authorization Server and Resource Server, there may be a case to link the item corresponding to the user information from the Authorization Server to the Resource Server, however when `RemoteTokenServices` is to be used as `TokenServices` by the Resource Server, the information other than the user information cannot be fetched by the annotation `@AuthenticationPrincipal`of handler method argument of the Resource Server.

An example of extending `org.springframework.security.oauth2.provider.token.DefaultAccessTokenCo` is a class to be used for linking access token by using `RemoteTokenServices`and linking the information other than the username to the Resource Server is shown here.

DefaultAccessTokenConverter means

Request authentication information of resource owner and client of the access token value to the Authorization Server from the Resource Server by using `RestTemplate`and fetch the result as Map for linking access token wherein `RemoteTokenServices` is used. `DefaultAccessTokenConverter` is used as a convertor that converts the authentication information in Authorization Server to Map and Map in Resource Server to the authentication information.

By using this, the parameters linked between the Authorization Server and the Resource Server can be customized

by extending ‘‘DefaultAccessTokenConverter‘‘ so as to add the return value from the Authorization Server to the Map.

In the following explanation, an example of linking independent items related to user information and independent items other than this to each other by customizing ‘‘DefaultAccessTokenConverter‘‘ and its property ‘‘DefaultUserAuthenticationConverter‘‘ in the Authorization Server, is shown.

Implementation of Authorization Server

How to implement Authorization Server is explained.

First, extend `DefaultUserAuthenticationConverter` for adding independent item of the user information.

- `AuthCustomUserTokenConverter.java`

```
public class CustomUserTokenConverter extends DefaultUserAuthenticationConverter {
    @Override
    public Map<String, ?> convertUserAuthentication(
            Authentication authentication) {
        Map<String, Object> response = new LinkedHashMap<String, Object>();
        response.put(USERNAME, authentication.getName());

        if (authentication.getAuthorities() != null &&
                !authentication.getAuthorities().isEmpty()) {
            response.put(AUTHORITIES, AuthorityUtils.authorityListToSet(
                    authentication.getAuthorities()));
        }
        response.put("user_additional_key", "user_additional_value"); // (1)
        return response;
    }
}
```

Sr.No.	Description
(1)	Define the information to be passed to the Resource Server as an independent item `user_additional_key` and set to `response`. The information set in `response`, is returned to the Resource Server in JSON format as response BODY at the time of verifying the token of check token endpoint.

Then, extend `DefaultAccessTokenConverter` for adding independent item of other than the user information.

- `CustomAccessTokenConverter.java`

```
public class CustomAccessTokenConverter extends DefaultAccessTokenConverter {
```

```
    @Override
    public Map<String, ?> convertAccessToken(OAuth2AccessToken token,␣
↪OAuth2Authentication authentication) {

        @SuppressWarnings("unchecked")
        Map<String, Object> response = (Map<String, Object>) super.
↪convertAccessToken(token, authentication);
        response.put("client_additional_key","client_additional_value"); // (1)
        // omitted

        return response;
    }
}
```

Sr. No.	Description
(1)	Define the information to be passed to the Resource Server as an independent item `client_additional_key` and set to `response`. The information set in `response`, is returned to the Resource Server in JSON format as response BODY at the time of verifying the token of check token endpoint.

Perform the settings of `CustomUserTokenConverter` and `CustomAccessTokenConverter` created in the configuration file of the Authorization Server.

- `oauth2-auth.xml`

```
<oauth2:authorization-server
     client-details-service-ref="clientDetailsService"
     token-endpoint-url="/oth2/token"
     authorization-endpoint-url="/oth2/authorize"
     user-approval-handler-ref="userApprovalHandler"
     token-services-ref="tokenServices"
     check-token-endpoint-url="/oth2/check-token">  <!-- (1) -->
    <oauth2:authorization-code />
    <oauth2:implicit />
    <oauth2:refresh-token />
    <oauth2:client-credentials />
    <oauth2:password />
</oauth2:authorization-server>

<bean id="checkTokenEndpoint"
    class="org.springframework.security.oauth2.provider.endpoint.CheckTokenEndpoint
↪">  <!-- (2) -->
    <constructor-arg ref="tokenServices" />
    <property name="accessTokenConverter" ref="accessTokenConverter" />
</bean>
```

```
<bean id="accessTokenConverter"
    class="com.example.oauth2.auth.converter.CustomAccessTokenConverter"/>  <!--␣
↪(3) -->
    <property name="userTokenConverter">
        <bean
            class="com.example.oauth2.auth.converter.CustomUserTokenConverter" />
    </property>
</bean>
```

Sr. No.	Description
(1)	Specify the URL of ‘‘CheckTokenEndpoint‘‘ defined in (2) in the `check-token-endpoint-url` attribute of ‘‘<oauth2: authorization-server>‘‘tag. When not specified, the default value “/ oauth / check_token” is specified. Since Bean definition for ‘‘CheckTokenEndpoint‘‘ is done independently in (2), do not specify ‘‘check-token-enabled‘‘attribute of ‘‘<oauth2:authorization-server>‘‘tag.
(2)	Define Bean for `CheckTokenEndpoint`. By specifying Bean of `CustomAccessTokenConverter` defined in (2) in `accessTokenConverter`property, independent items added in `CustomAccessTokenConverter` and `CustomUserTokenConverter` can be linked to the Resource Server.
(3)	Define Bean for `CustomAccessTokenConverter` wherein `DefaultAccessTokenConverter` is extended. Specify Bean for `CustomUserTokenConverter` wherein `DefaultUserAuthenticationConverter` is extended in `userTokenConverter` property.

Implementation of Resource Server

Add the function to fetch the information linked from Authorization Server in the Resource Server by annotation `@AuthenticationPrincipal` of the handler method argument. First, create `OauthUser` class holding the information to be fetched by annotation `@AuthenticationPrincipal`.

- `OauthUser.java`

```
public class OauthUser implements Serializable{

```

```
    private static final long serialVersionUID = 1L;

    private String username;

    private String userAdditionalValue;

    private String clientAdditionalValue;

    // omitted

    public User(String username, String additionalValue){
        this.username = username;
        this.additionalValue = additionalValue;
    }

    // Getters and Setters are omitted

}
```

`org.springframework.security.oauth2.provider.token.DefaultAccessTokenConverter` which performs setup for enabling fetching of user information by `@AuthenticationPrincipal` annotationverter‘‘ is expanded and a function is added to enable fetching of information other than user name.

- `CustomUserTokenConverter.java`

```
public class CustomUserTokenConverter extends DefaultUserAuthenticationConverter{

    private Collection<? extends GrantedAuthority> defaultAuthorities; // (1)

    public void setDefaultAuthorities(String[] defaultAuthorities) {
        this.defaultAuthorities = AuthorityUtils.
↪commaSeparatedStringToAuthorityList(StringUtils
                .arrayToCommaDelimitedString(defaultAuthorities));
    }

     // (2)
    @Override
    public Authentication extractAuthentication(Map<String, ?> map) {
        if (map.containsKey(USERNAME)) {
            Collection<? extends GrantedAuthority> authorities =␣
↪getAuthorities(map);
            OauthUser user = new OauthUser(
                    (String) map.get(USERNAME),
                    (String) map.get("user_additional_key"),
                    (String) map.get("client_additional_key"),
                    (String) map.get("client_id")); // (3)

            // omitted

            return new UsernamePasswordAuthenticationToken(user, "N/A",␣
↪authorities); // (4)
```

```
        }
        return null;
    }

    private Collection<? extends GrantedAuthority> getAuthorities(Map<String, ?>␣
↪map) {
        if (!map.containsKey(AUTHORITIES)) {
            return defaultAuthorities;
        }
        Object authorities = map.get(AUTHORITIES);
        if (authorities instanceof String) {
            return AuthorityUtils.commaSeparatedStringToAuthorityList((String)␣
↪authorities);
        }
        if (authorities instanceof Collection) {
            return AuthorityUtils.commaSeparatedStringToAuthorityList(StringUtils
                    .collectionToCommaDelimitedString((Collection<?>)␣
↪authorities));
        }
        throw new IllegalArgumentException("Authorities must be either a String or␣
↪a Collection");
    }
}
```

Sr.No.	Description
(1)	Implement `defaultAuthorities` used by `defaultAuthorities` and `getAuthorities` method since `getAuthorities` method implemented by `DefaultUserAuthenticationConverter` is defined by private.
(2)	A method that extracts authentication information from the information linked from authorization server.
(3)	Set information linked from authorization server in `OauthUser` class.
(4)	Information linked from authorization server can be fetched `@AuthenticationPrincipal` annotation by setting `OauthUser` in the first argument of `UsernamePasswordAuthenticationToken`.

Configure `CustomUserTokenConverter` in setup file of resource server.

- oauth2-resource.xml

```
<bean id="tokenServices"
    class="org.springframework.security.oauth2.provider.token.RemoteTokenServices">
    <property name="checkTokenEndpointUrl" value="${auth.serverUrl}/oth2/check-
↪token" />
    <property name="accessTokenConverter" ref="accessTokenConverter" />
</bean>

<bean id="accessTokenConverter"
    class="org.springframework.security.oauth2.provider.token.
↪DefaultAccessTokenConverter">
    <property name="userTokenConverter">
        <bean class="com.example.oauth2.resource.converter.CustomUserTokenConverter
↪"/>  <!-- (1) -->
    </property>
</bean>
```

Sr. No.	Description
(1)	Information other than user name can be fetched by `@AuthenticationPrincipal` annotation, by specifying `CustomUserTokenConverter` class in `userTokenConverter` property of `accessTokenConverter`.

10

Tutorials

10.1 Tutorial (Todo Application)

10.1.1 Introduction

Points to study in this tutorial

- Basic application development using TERASOLUNA Server Framework for Java (5.x)
- Configuration of Maven and STS(Eclipse) project
- Way of development using application layering of TERASOLUNA Server Framework for Java (5.x).

Target readers

- Basic knowledge of Spring DI and AOP required
- Web application development using Servlet/JSP
- SQL knowledge

Verification environment

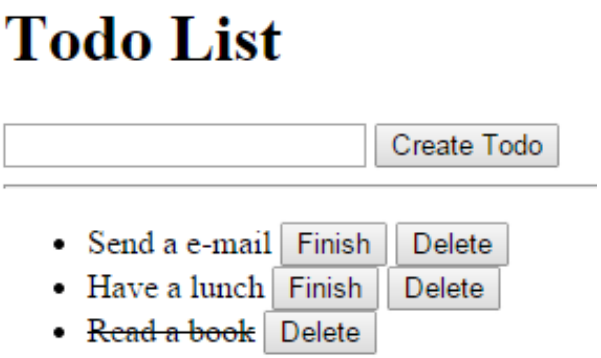
In this tutorial, operations are verified on following environment. In case of implementation on other environment, perform appropriate setting based on this guideline.

Type	Name
OS	Windows 7
JVM	Java 1.8
IDE	Spring Tool Suite 3.6.4.RELEASE (Onwards referred as [STS])
Build Tool	Apache Maven 3.3.9 (Onwards referred as [Maven])
Application Server	Pivotal tc Server Developer Edition v3.1 (Enclosed in STS)
Web Browser	Google Chrome 46.0.2490.80 m

10.1.2 Description of application to be created

Overview of the application

Application to manage TODO list is to be developed. TODO list display, TODO registration, TODO completion and TODO deletion can be performed.



Business requirements of application

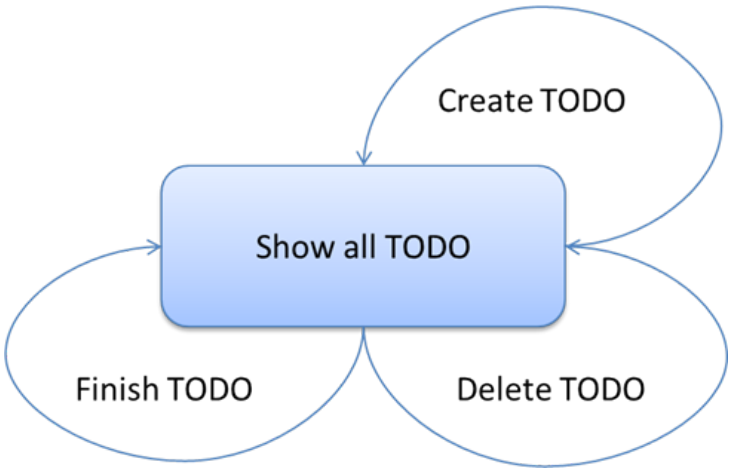
Business requirement of the application is as follows.

RuleID	Description
B01	Only up to 5 incomplete TODO records can be registered
B02	For TODOs which are already completed, “TODO Complete” processing cannot be done.

Note: This application is for learning purpose only. It is not suitable as a real todo management application.

Screen transition of application

Processing specification and screen transition of an application is as follows.



Sr.No.	Process name	HTTP method	URL	Remark
1	Show all TODO	-	/todo/list	
2	Create TODO	POST	/todo/create	Redirect to Show all TODO after creation is completed
3	Finish TODO	POST	/todo/finish	Redirect to Show all TODO after completion process
4	Delete TODO	POST	/todo/delete	Redirect to Show all TODO after deletion is completed

Show all TODO

- Display all records of TODO
- Provide `Finish` and `Delete` buttons for incomplete TODO
- Strike-through the completed records of TODO
- Display only record name of TODO

Create TODO

- Save TODO sent from the form
- Record name of TODO should be between 1 - 30 characters
- When *Business requirements of application* B01 is not fulfilled, business exception with error code E001 is thrown
- Display “Created successfully!” at the transited screen when the creation process is successful.

Finish TODO

- For the TODOs corresponding to `todoId` which is received from the form object, change the status to `completed`.
- When the corresponding TODO does not exist, resource not found exception with error code E404 is thrown
- When *Business requirements of application* B02 is not fulfilled, business exception with error code E002 is thrown
- Display "Finished successfully!" at the transited screen when the finishing process is successful.

Delete TODO

- Delete TODO corresponding to `todoId` sent from the form
- When the corresponding TODO does not exist, resources undetected exception with error code E404 is thrown
- Display "Deleted successfully!" at the transited screen when the deletion process is successful.

Error message list

Define below 3 error messages.

Error code	Message	Parameter to be replaced
E001	[E001] The count of un-finished Todo must not be over {0}.	{0}… max unfinished count
E002	[E002] The requested Todo is already finished. (id={0})	{0}… todoId
E404	[E404] The requested Todo is not found. (id={0})	{0}… todoId

10.1.3 Environment creation

In this tutorial, as RepositoryImpl implementation of the infrastructure layer,

- In-memory implementation of RepositoryImpl using `java.util.Map` without using the database
- RepositoryImpl access the database using MyBatis3

- RepositoryImpl access the database using Spring Data JPA

3 types are prepared. Select anyone depending upon the use.

Under this tutorial, First, try the in-memory implementation followed by select myBatis3 or Spring Data JPA.

Project creation

First, create a blank project for implementation of infrastructure layer using `mvn archetype:generate`. This is a procedure to create a blank project using the Windows command prompt.

Note: If internet connection is accessed through proxy server, In order to perform the following tasks, necessary STS proxy settings and Maven proxy setting needs to be done.

Tip: If `mvn archetype:generate` executes on Bash, it can be executed by replacing the `^` with `\`.

```
mvn archetype:generate -B\
 -DarchetypeGroupId=org.terasoluna.gfw.blank\
 -DarchetypeArtifactId=terasoluna-gfw-web-blank-archetype\
 -DarchetypeVersion=5.3.0.RELEASE\
 -DgroupId=todo\
 -DartifactId=todo\
 -Dversion=1.0.0-SNAPSHOT
```

Creating O/R Mapper independent blank project

If you want to create a project for RepositoryImpl using `java.util.Map`(without accessing the database), run the following command to create O/R Mapper independent blank project in command prompt. **If you read through this tutorial in consecutive order, first of all, create a project in this way**.

```
mvn archetype:generate -B^
 -DarchetypeGroupId=org.terasoluna.gfw.blank^
 -DarchetypeArtifactId=terasoluna-gfw-web-blank-archetype^
 -DarchetypeVersion=5.3.0.RELEASE^
 -DgroupId=todo^
 -DartifactId=todo^
 -Dversion=1.0.0-SNAPSHOT
```

Creating blank project for MyBatis3

If you want to create a project for RepositoryImpl to access the database using MyBatis3, run the following command to create a blank project for the MyBatis3. This way to create a project is to be done in *Creating infrastructure layer with MyBatis3*.

```
mvn archetype:generate -B^
 -DarchetypeGroupId=org.terasoluna.gfw.blank^
 -DarchetypeArtifactId=terasoluna-gfw-web-blank-mybatis3-archetype^
 -DarchetypeVersion=5.3.0.RELEASE^
 -DgroupId=todo^
 -DartifactId=todo^
 -Dversion=1.0.0-SNAPSHOT
```

Creating blank project for JPA

If you want to create a project for RepositoryImpl to access the database using Spring Data JPA, run the following command to create a blank project for the JPA. This way to create a project is to be done in *Creating infrastructure layer with Spring Data JPA*.

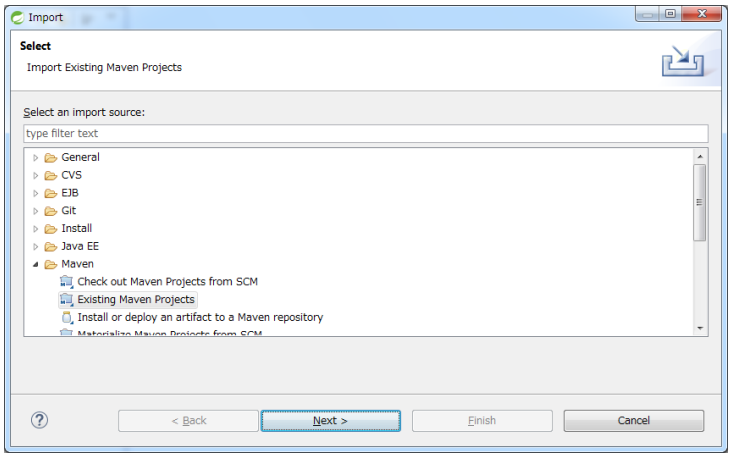
```
mvn archetype:generate -B^
 -DarchetypeGroupId=org.terasoluna.gfw.blank^
 -DarchetypeArtifactId=terasoluna-gfw-web-blank-jpa-archetype^
 -DarchetypeVersion=5.3.0.RELEASE^
 -DgroupId=todo^
 -DartifactId=todo^
 -Dversion=1.0.0-SNAPSHOT
```

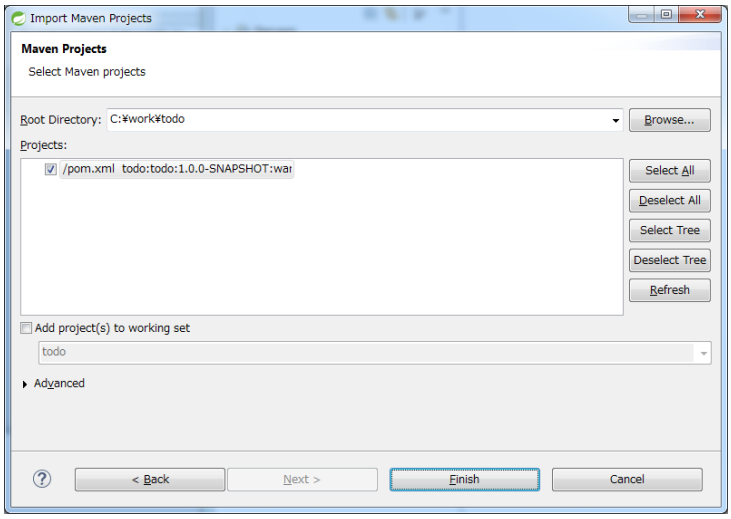
Project import

Import created blank project into STS.

Select the archetype created project from STS menu [File] -> [Import] -> [Maven] -> [Existing Maven Projects] -> [Next].

Click on [Finish] by selecting `C:\work\todo` in Root Directory and selecting pom.xml of todo in Projects.



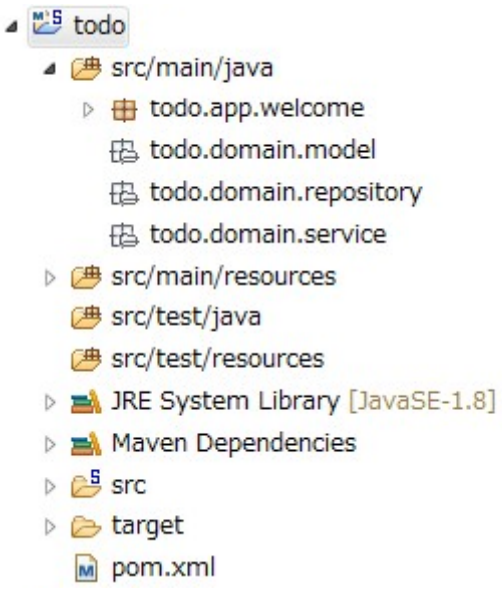


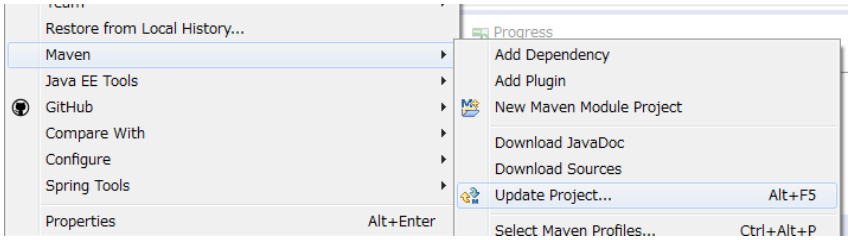
When the import is completed, project is displayed in the Package Explorer as follows.

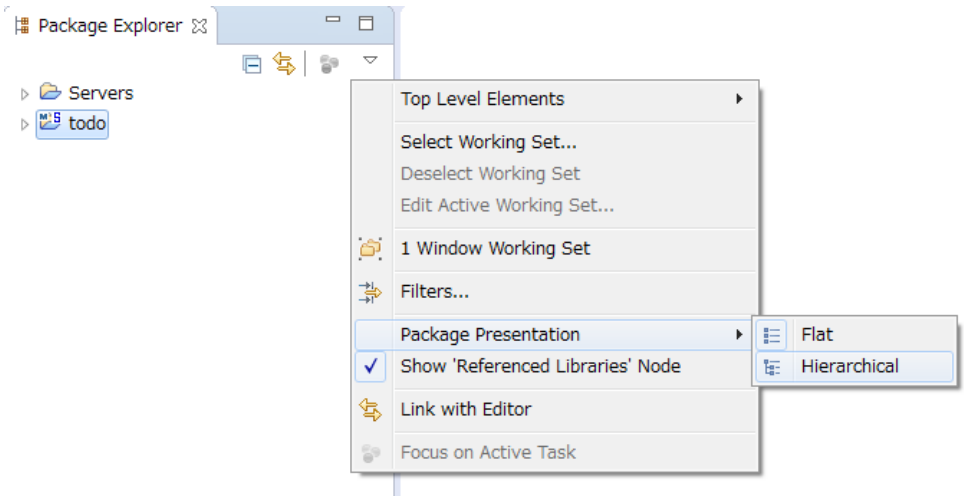
Note: If the build error occurs after the import, it can be removed by Right click on the project name in Package Explorer and select [Maven] -> [Update Project...] -> [OK] .

Tip: For better visibility, Package Presentation must be changed to [Hierarchical] from default [Flat].

Click [View Menu] (The right edge of the down arrow) of the Package Explorer and select [Package Presentation] -> [Hierarchical].

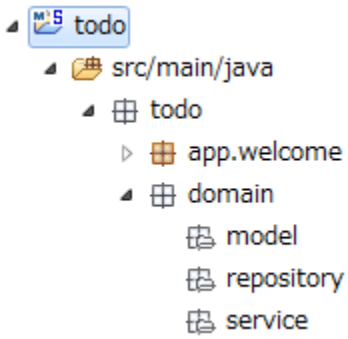






It can be displayed as follows if Package Presentation changed into [Hierarchical].

Warning: H2 Database has been defined as a dependency in O/R Mapper type blank project but, this setting is done to create a simple application easily therefore it is not intended to use in the actual application



development.

The following definitions shall be removed while performing the actual application development.

```
<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <scope>runtime</scope>
</dependency>
```

note:

```
In the above setting example, since it is assumed that the dependent library␣
↪version is managed by the parent project terasoluna-gfw-parent , specifying the␣
↪version in pom.xml is not necessary.
The above dependent library used by terasoluna-gfw-parent is defined by \ `Spring␣
↪IO Platform <http://platform.spring.io/platform/>`_\ .
```

Project configuration

Below is the structure of the project to be created in this tutorial.

Note: It had been recommended to use a multi-project structure in *[Project Structure] section of previous chapter* but, in this tutorial, a single project configuration is used because it focuses on ease of learning. However, when in a real project, multi project configuration is strongly recommended.

However, when in a real project, multi project configuration is strongly recommended.

For creating multi-project, Refer [*Create Web application development project*].

[Configuration of blank project created for MyBatis3]

```
src
  └main
      ├java
      |  └todo
      |    ├ app
      |    |   └todo
      |    └domain
      |        ├model
      |        ├repository
      |        |   └todo
      |        └service
      |            └todo
      ├resources
      |  ├META-INF
      |  |  ├mybatis ... (8)
      |  |  └spring
      |  └todo
      |    └domain
      |        └repository ... (9)
      |            └todo
      └wepapp
          └WEB-INF
              └views
```

Sr. No.	Description
(8)	Directory that contains the MyBatis configuration files.
(9)	Directory that contains the Mapper of MyBatis describing the SQL. In this tutorial, create a directory for storing the Mapper file of Repository for Todo object.

[Configuration of blank project created for JPA, O/R Mapper independent blank project]

```
src
  └main
      ├java
      |  └todo
```

```
|     ├ app ... (1)
|     |   └todo
|     └domain ... (2)
|         ├model ... (3)
|         ├repository ... (4)
|         |   └todo
|         └service ... (5)
|             └todo
├resources
|   └META-INF
|       └spring ... (6)
└wepapp
    └WEB-INF
        └views ... (7)
```

Sr. No.	Description
(1)	Packages for storing application layer classes. In this tutorial, creating package for storing the Todo business classes.
(2)	Packages for storing domain layer class.
(3)	Packages for storing Domain Object.
(4)	Packages for storing Repository In this tutorial, creating package for storing the Todo object (Domain Object) Repository.
(5)	Packages for storing Service. In this tutorial, creating package for storing the Todo business services.
(6)	Directory that contains the Spring configuration files.
(7)	Directory for storing jsp.

Confirmation of configuration file

Many settings those are required in advanced tutorial already done in the created blank project.

If only the implementation of the tutorial is concern, understanding of these settings are not required but it is recommended that you understand what settings are necessary to run an application.

For description of the required configuration (settings file) to run an application, Refer [*Description of the configuration file*].

Note: If you are creating a Todo application for familiarizing with the system, you may skip the confirmation of configuration file but suggest to read this after creating the Todo application.

Operation verification of the project

Before starting the development of Todo application, verify the project operation.

Since the implementation of the Controller and JSP for displaying the top page are provided in the blank project, it is possible to check the operation by displaying the top page.

The following implementation has been done in the Controller(`src/main/java/todo/app/welcome/HelloController` provided in the blank project.

```
package todo.app.welcome;

import java.text.DateFormat;
import java.util.Date;
import java.util.Locale;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

/**
 * Handles requests for the application home page.
 */
// (1)
@Controller
public class HelloController {
```

```
    // (2)
    private static final Logger logger = LoggerFactory
            .getLogger(HelloController.class);

    /**
     * Simply selects the home view to render by returning its name.
     */
    // (3)
    @RequestMapping(value = "/", method = {RequestMethod.GET, RequestMethod.POST})
    public String home(Locale locale, Model model) {
        // (4)
        logger.info("Welcome home! The client locale is {}.", locale);

        Date date = new Date();
        DateFormat dateFormat = DateFormat.getDateTimeInstance(DateFormat.LONG,
                DateFormat.LONG, locale);

        String formattedDate = dateFormat.format(date);

        // (5)
        model.addAttribute("serverTime", formattedDate);

        // (6)
        return "welcome/home";
    }

}
```

Sr. No.	Description
(1)	In order to make the Controller as component-scan target, attach `@Controller` annotation to class level.
(2)	Generating logger for outputting the log at (4). Logger impmements the logback but, the API `org.slf4j.Logger` of SLF4J is used.
(3)	Set mapping methods for accessing the `"/"` (root) using `@RequestMapping` annotation.
(4)	Outputting info level log for notifying that the method is called.
(5)	For displaying date on screen set date as `"serverTime"` attribute name to the Model.
(6)	Return `"welcome/home"` as view name. Using `ViewResolver` settings, `WEB-INF/views/welcome/home.jsp` is called.

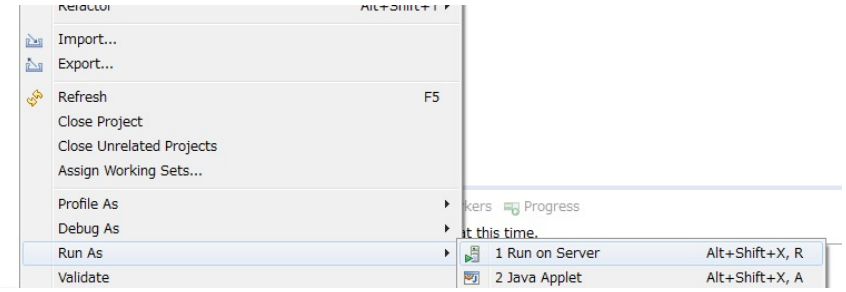
The following implementation has been done in the JSP(`src/main/webapp/WEB-INF/views/welcome/home.jsp`), provided in the blank project.

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>Home</title>
<link rel="stylesheet"
    href="${pageContext.request.contextPath}/resources/app/css/styles.css">
</head>
<body>
    <div id="wrapper">
        <h1>Hello world!</h1>
        <!-- (7) -->
```

```
        <p>The time on the server is ${serverTime}.</p>
    </div>
</body>
</html>
```

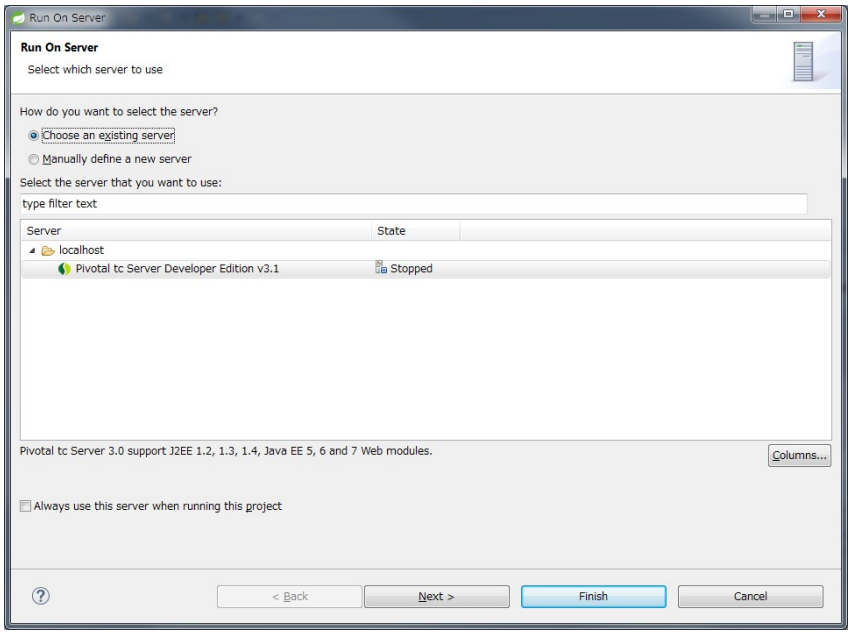
Sr. No.	Description
(7)	Display "serverTime" passed from Controller in Model. Here, XSS measures are done but always performs the XSS measures for displaying the user input values by f:h() function.

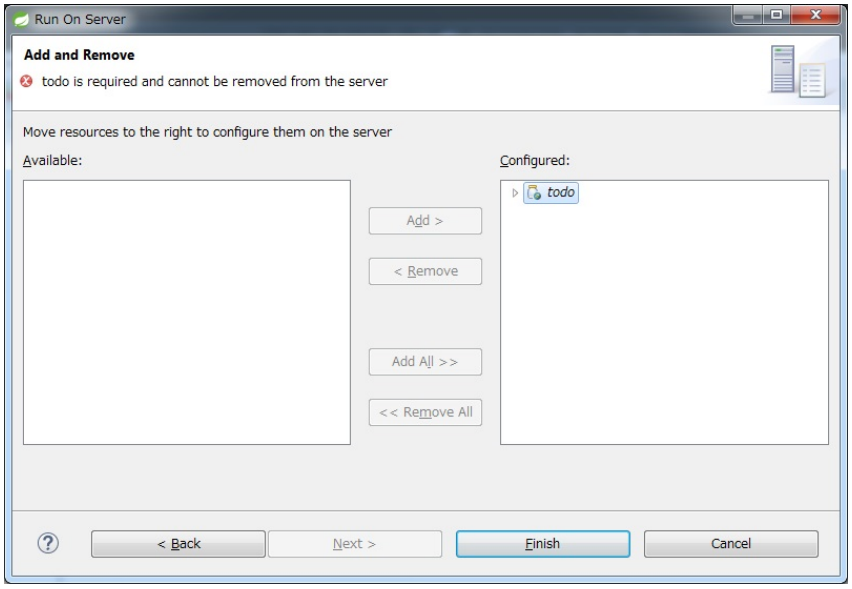
Right click on project and select [Run As] -> [Run on Server].



Select [Next] after selecting AP server (Pivotal tc Server Developer Edition v3.1).

Verify that todo is included in [Configured] and click [Finish] to start the server.





When started, log shown as below will be output. For `"/"` path, it is understood that hello method of `todo.app.welcome.HelloController` is mapped.

```
date:2016-02-17 11:25:30    thread:localhost-startStop-1    X-Track:        level:
↪INFO      logger:o.springframework.web.servlet.DispatcherServlet  message:
↪FrameworkServlet 'appServlet': initialization started
date:2016-02-17 11:25:31    thread:localhost-startStop-1    X-Track:        level:
↪DEBUG     logger:o.t.gfw.web.codelist.CodeListInterceptor         message:
↪registered codeList : []
date:2016-02-17 11:25:31    thread:localhost-startStop-1    X-Track:        level:
↪INFO      logger:o.s.w.s.m.m.a.RequestMappingHandlerMapping       message:Mapped
↪"{[/],methods=[GET || POST],params=[],headers=[],consumes=[],produces=[],
↪custom=[]}" onto public java.lang.String todo.app.welcome.HelloController.
↪home(java.util.Locale,org.springframework.ui.Model)
```

```
date:2016-02-17 11:25:31    thread:localhost-startStop-1    X-Track:        level:
↪INFO      logger:o.s.w.s.m.m.a.RequestMappingHandlerAdapter       message:
↪Looking for @ControllerAdvice: WebApplicationContext for namespace 'appServlet-
↪servlet': startup date [Wed Feb 17 11:25:30 JST 2016]; parent: Root␣
↪WebApplicationContext
date:2016-02-17 11:25:32    thread:localhost-startStop-1    X-Track:        level:
↪INFO      logger:o.s.w.s.m.m.a.RequestMappingHandlerAdapter       message:
↪Looking for @ControllerAdvice: WebApplicationContext for namespace 'appServlet-
↪servlet': startup date [Wed Feb 17 11:25:30 JST 2016]; parent: Root␣
↪WebApplicationContext
date:2016-02-17 11:25:32    thread:localhost-startStop-1    X-Track:        level:
↪INFO      logger:o.s.web.servlet.handler.SimpleUrlHandlerMapping  message:Mapped␣
↪URL path [/**] onto handler 'org.springframework.web.servlet.resource.
↪DefaultServletHttpRequestHandler#0'
date:2016-02-17 11:25:32    thread:localhost-startStop-1    X-Track:        level:
↪INFO      logger:o.s.web.servlet.handler.SimpleUrlHandlerMapping  message:Mapped␣
↪URL path [/resources/**] onto handler 'org.springframework.web.servlet.resource.
↪ResourceHttpRequestHandler#0'
date:2016-02-17 11:25:33    thread:localhost-startStop-1    X-Track:        level:
↪INFO      logger:o.springframework.web.servlet.DispatcherServlet  message:
↪FrameworkServlet 'appServlet': initialization completed in 2826 ms
```

If http://localhost:8080/todo is accessed in browser, following is displayed.



If you confirm the console,

- TRACE log of `TraceLoggingInterceptor` provided by the common library
- INFO log implemented in the Controller

are the log output.

```
date:2016-02-17 11:25:35    thread:tomcat-http--11  X-Track:
↪b49b630274974bffbcd9e8d13261f6a7        level:TRACE     logger:o.t.gfw.web.
↪logging.TraceLoggingInterceptor      message:[START CONTROLLER] HelloController.
↪home(Locale,Model)
```

```
date:2016-02-17 11:25:35    thread:tomcat-http--11  X-Track:
↪b49b630274974bffbcd9e8d13261f6a7        level:INFO      logger:todo.app.welcome.
↪HelloController                         message:Welcome home! The client locale␣
↪is ja_JP.
date:2016-02-17 11:25:35    thread:tomcat-http--11  X-Track:
↪b49b630274974bffbcd9e8d13261f6a7        level:TRACE     logger:o.t.gfw.web.
↪logging.TraceLoggingInterceptor      message:[END CONTROLLER  ] HelloController.
↪home(Locale,Model)-> view=welcome/home, model={serverTime=2016/02/17 11:25:35␣
↪JST}
date:2016-02-17 11:25:35    thread:tomcat-http--11  X-Track:
↪b49b630274974bffbcd9e8d13261f6a7        level:TRACE     logger:o.t.gfw.web.
↪logging.TraceLoggingInterceptor      message:[HANDLING TIME   ] HelloController.
↪home(Locale,Model)-> 97,346,576 ns
```

Note: The `TraceLoggingInterceptor` outputs start and end log of the Controller. While ending, the information of `View` and `Model` as well as processing time are output.

10.1.4 Creation of Todo application

Create Todo application. Order in which it must be created is as follows

- Domain layer (+ Infrastructure layer)
- Domain Object creation
- Repository creation
- RepositoryImpl creation
- Service creation
- Application layer
- Controller creation
- Form creation
- View creation

About the creation of RepositoryImpl, implementation is differ depending on the type of the selected infrastructure layer.

Here, In-memory implemented RepositoryImpl is created using `java.util.Map` without using the database is explained. If you want to use database, create Todo application by referring the [*Creating infrastructure layer with a Database access*] content.

Creation of Domain layer

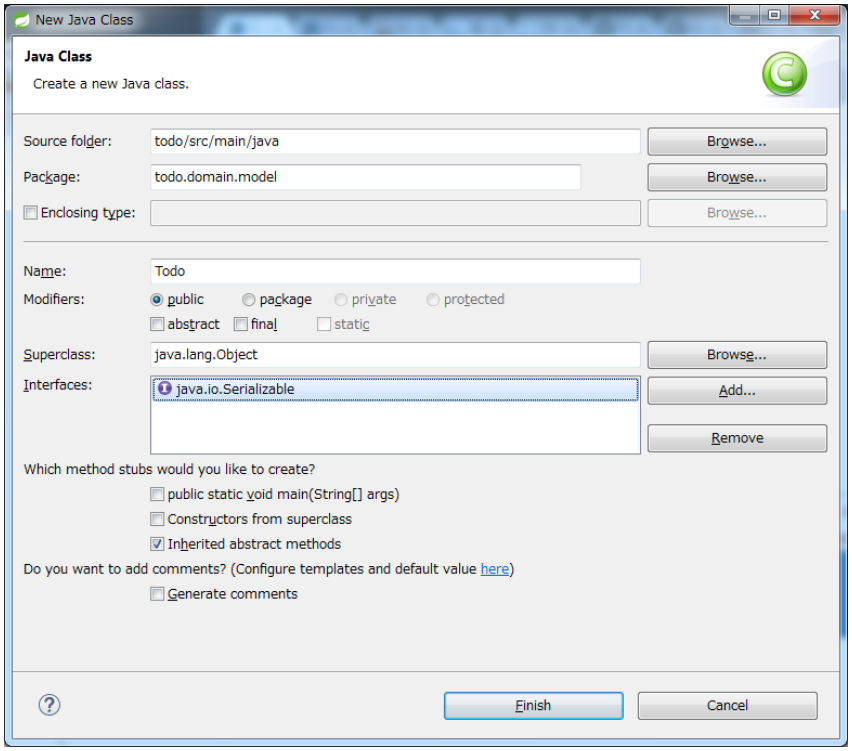
Creation of Domain Object

Create Domain object.

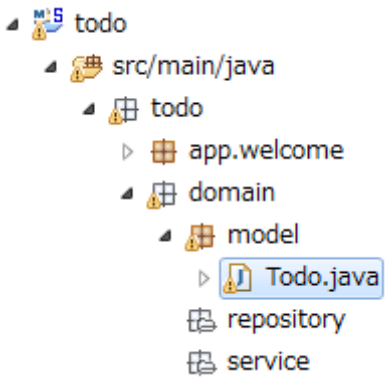
Right click on the Package Explorer, select -> [New] -> [Class] -> [New Java Class] dialog box appears,

Sr. No.	Item	Input value
1	Package	`todo.domain.model`
2	Name	`Todo`
3	Interfaces	`java.io.Serializable`

and click [Finish] after entering above information.



Created class stored in the following directory.



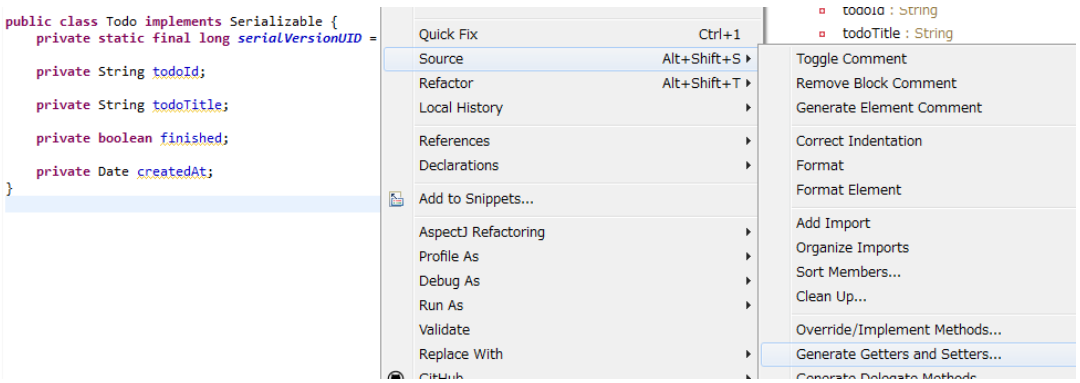
Add following properties in created class.

- ID = todoId
- Title = todoTitle
- Completion flag = finished
- Created on = createdAt

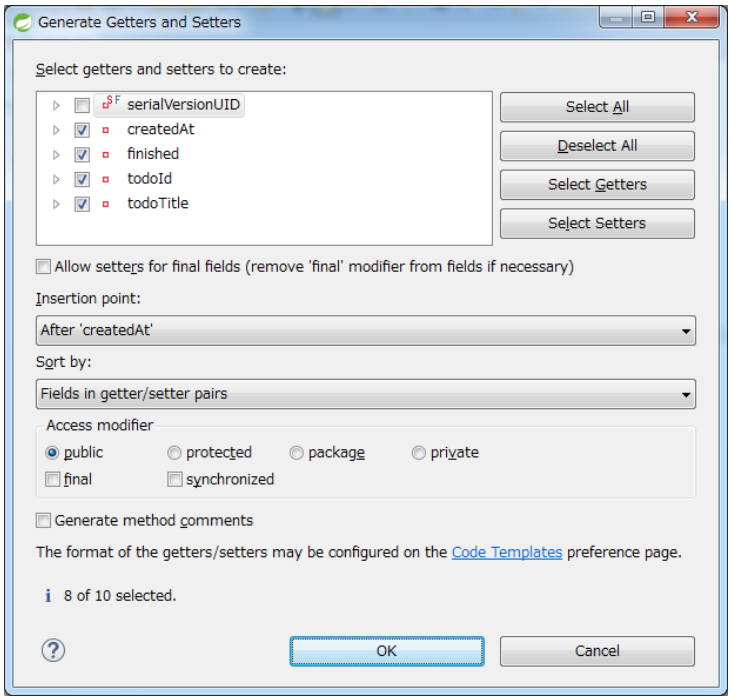
```
package todo.domain.model;

import java.io.Serializable;
import java.util.Date;

public class Todo implements Serializable {

    private static final long serialVersionUID = 1L;

    private String todoId;

    private String todoTitle;

    private boolean finished;

    private Date createdAt;

    public String getTodoId() {
        return todoId;
    }
```

```
    public void setTodoId(String todoId) {
        this.todoId = todoId;
    }

    public String getTodoTitle() {
        return todoTitle;
    }

    public void setTodoTitle(String todoTitle) {
        this.todoTitle = todoTitle;
    }

    public boolean isFinished() {
        return finished;
    }

    public void setFinished(boolean finished) {
        this.finished = finished;
    }

    public Date getCreatedAt() {
        return createdAt;
    }

    public void setCreatedAt(Date createdAt) {
        this.createdAt = createdAt;
    }
}
```

Tip: Getter/Setter methods can be generated automatically by using STS feature. After defining fields, right click on editor and select [Source] -> [Generate Getter and Setters…]



Click [OK] after selecting all other than serialVersionUID.



Repository creation

Create `TodoRepository` interface. If you want use database, create Repository by referring the [*Creating infrastructure layer with a Database access*] content.

Right click on the Package Explorer, select -> [New] -> [Interface] -> [New Java Interface] dialog box appears,

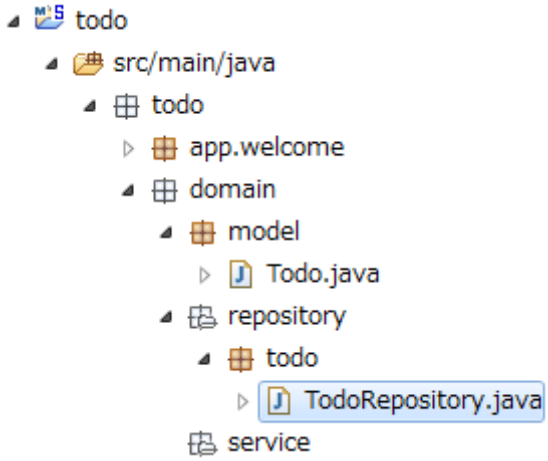
Sr. No.	Item	Input value
1	Package	`todo.domain.repository.todo`
2	Name	`TodoRepository`

and click [Finish] after entering above information.

Created Interface stored in the following directory.

Define following CRUD operation methods pertaining to this application in created interface.

- Fetch 1 record of TODO = findOne
- Fetch all records of TODO = findAll
- Create 1 record of TODO = create



- Update 1 record of TODO = update
- Delete 1 record of TODO = delete
- Fetch record count of completed TODO = countByFinished

```
package todo.domain.repository.todo;

import java.util.Collection;

import todo.domain.model.Todo;

public interface TodoRepository {
    Todo findOne(String todoId);

    Collection<Todo> findAll();

    void create(Todo todo);

    boolean update(Todo todo);

    void delete(Todo todo);

    long countByFinished(boolean finished);
}
```

Note: Here, to improve versatility of `TodoRepository`, instead of (`long countFinished()`) method for [Fetch completed record count], (`long countByFinished(boolean)`) method for [record count having xx completion status] is defined.

If pass `true` as an argument to `long countByFinished(boolean)`, [completed record count] and if pass `false` as an argument, [not completed record count] can be fetched by specification.

Creation of RepositoryImpl (Infrastructure layer)

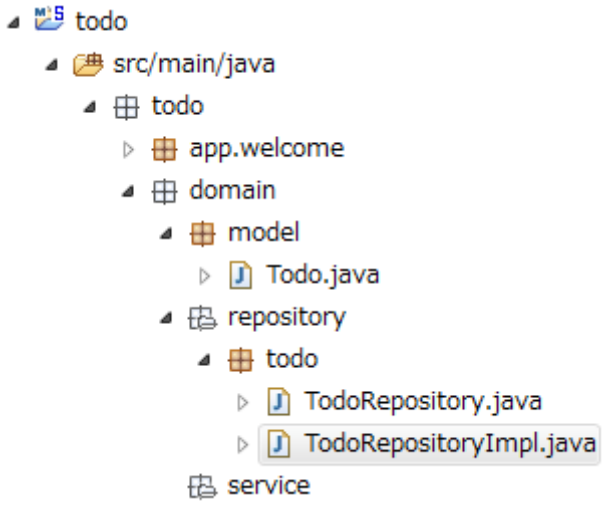
Here, In-memory implemented RepositoryImpl is created using `java.util.Map` for simplification. If you want use database, create RepositoryImpl by referring the [*Creating infrastructure layer with a Database access*] content.

Right click on the Package Explorer, select -> [New] -> [Class] -> [New Java Class] dialog box appears,

Sr. No.	Item	Input value
1	Package	`todo.domain.repository.todo`
2	Name	`TodoRepositoryImpl`
3	Interfaces	`todo.domain.repository.todo.TodoRepository`

and click [Finish] after entering above information.

Created class stored in the following directory.



Implement the CRUD operations in created class.

Note: Business logic must not be included in RepositoryImpl, it should focus only on inserting and removing (CRUD operation) from the persistence store.

```
package todo.domain.repository.todo;

import java.util.Collection;
import java.util.Map;
import java.util.concurrent.ConcurrentHashMap;
```

```
import org.springframework.stereotype.Repository;

import todo.domain.model.Todo;

@Repository // (1)
public class TodoRepositoryImpl implements TodoRepository {
    private static final Map<String, Todo> TODO_MAP = new ConcurrentHashMap<String,
↪ Todo>();

    @Override
    public Todo findOne(String todoId) {
        return TODO_MAP.get(todoId);
    }

    @Override
    public Collection<Todo> findAll() {
        return TODO_MAP.values();
    }

    @Override
    public void create(Todo todo) {
        TODO_MAP.put(todo.getTodoId(), todo);
    }

    @Override
    public boolean update(Todo todo) {
        TODO_MAP.put(todo.getTodoId(), todo);
        return true;
    }

    @Override
    public void delete(Todo todo) {
        TODO_MAP.remove(todo.getTodoId());
    }

    @Override
    public long countByFinished(boolean finished) {
        long count = 0;
        for (Todo todo : TODO_MAP.values()) {
            if (finished == todo.isFinished()) {
                count++;
            }
        }
        return count;
    }
}
```

Sr. No.	Description
(1)	To consider Repository as component scan target, add `@Repository` annotation at class level.

Note: In this tutorial, although infrastructure layer belonging classes (RepositoryImpl) are stored under domain layer package (`todo.domain`), if package is divided completed on the basis of layers, it is better to create classes of infrastructure layer under `todo.infra`.

However, in a normal project, infrastructure layer rarely changes (such projects are less). Hence, in order to improve the work efficiency, RepositoryImpl can be created in the layer same as the repository of domain layer.

Service creation

First create `TodoService` interface.

Right click on the Package Explorer, select -> [New] -> [Interface] -> [New Java Interface] dialog box appears,

Sr. No.	Item	Input value
1	Package	`todo.domain.service.todo`
2	Name	`TodoService`

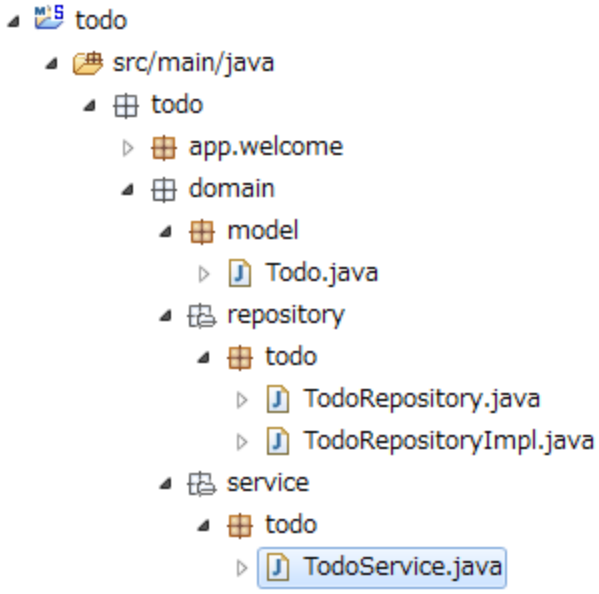
and click [Finish] after entering above information.

Created Interface stored in the following directory.

Define method to perform the following business processing in created class.

- Fetch all records of Todo = findAll
- New creation of Todo = create
- Completed of Todo = finish
- Delete of Todo = delete

```
package todo.domain.service.todo;

import java.util.Collection;

import todo.domain.model.Todo;
```




```
public interface TodoService {
    Collection<Todo> findAll();

    Todo create(Todo todo);

    Todo finish(String todoId);

    void delete(String todoId);
}
```

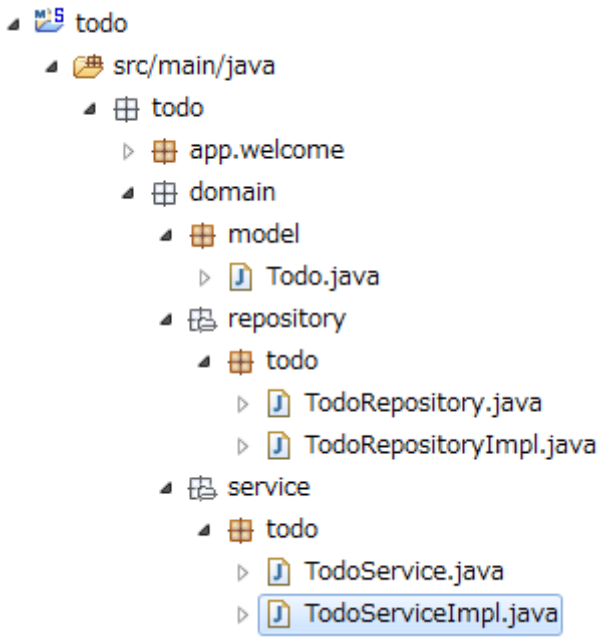
Next, create `TodoServiceImpl` class that implements the methods defined in `TodoService` interface.

Right click on the Package Explorer, select -> [New] -> [Class] -> [New Java Class] dialog box appears,

Sr. No.	Item	Input value
1	Package	`todo.domain.service.todo`
2	Name	`TodoServiceImpl`
3	Interfaces	`todo.domain.service.todo.TodoService`

and click [Finish] after entering above information.

Created class stored in the following directory.




```
package todo.domain.service.todo;

import java.util.Collection;
import java.util.Date;
import java.util.UUID;

import javax.inject.Inject;

import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import org.terasoluna.gfw.common.exception.BusinessException;
import org.terasoluna.gfw.common.exception.ResourceNotFoundException;
import org.terasoluna.gfw.common.message.ResultMessage;
import org.terasoluna.gfw.common.message.ResultMessages;

import todo.domain.model.Todo;
import todo.domain.repository.todo.TodoRepository;

@Service// (1)
@Transactional // (2)
public class TodoServiceImpl implements TodoService {

    private static final long MAX_UNFINISHED_COUNT = 5;

    @Inject// (3)
    TodoRepository todoRepository;

    // (4)
    public Todo findOne(String todoId) {
        Todo todo = todoRepository.findOne(todoId);
```

```
        if (todo == null) {
            // (5)
            ResultMessages messages = ResultMessages.error();
            messages.add(ResultMessage
                    .fromText("[E404] The requested Todo is not found. (id="
                            + todoId + ")"));
            // (6)
            throw new ResourceNotFoundException(messages);
        }
        return todo;
    }

    @Override
    @Transactional(readOnly = true) // (7)
    public Collection<Todo> findAll() {
        return todoRepository.findAll();
    }

    @Override
    public Todo create(Todo todo) {
        long unfinishedCount = todoRepository.countByFinished(false);
        if (unfinishedCount >= MAX_UNFINISHED_COUNT) {
            ResultMessages messages = ResultMessages.error();
            messages.add(ResultMessage
                    .fromText("[E001] The count of un-finished Todo must not be␣
↪over "
                            + MAX_UNFINISHED_COUNT + "."));
            // (8)
            throw new BusinessException(messages);
        }

        // (9)
        String todoId = UUID.randomUUID().toString();
        Date createdAt = new Date();

        todo.setTodoId(todoId);
        todo.setCreatedAt(createdAt);
        todo.setFinished(false);

        todoRepository.create(todo);
        /* REMOVE THIS LINE IF YOU USE JPA
            todoRepository.save(todo); // 10
           REMOVE THIS LINE IF YOU USE JPA */

        return todo;
    }

    @Override
    public Todo finish(String todoId) {
        Todo todo = findOne(todoId);
```

```
        if (todo.isFinished()) {
            ResultMessages messages = ResultMessages.error();
            messages.add(ResultMessage
                    .fromText("[E002] The requested Todo is already finished. (id="
                            + todoId + ")"));
            throw new BusinessException(messages);
        }
        todo.setFinished(true);
        todoRepository.update(todo);
        /* REMOVE THIS LINE IF YOU USE JPA
            todoRepository.save(todo); // (11)
           REMOVE THIS LINE IF YOU USE JPA */
        return todo;
    }

    @Override
    public void delete(String todoId) {
        Todo todo = findOne(todoId);
        todoRepository.delete(todo);
    }
}
```

Sr. No.	Description
(1)	To consider Service as component-scan target, add `@Service` annotation at class level.
(2)	All public methods will be treated as transaction management by attaching the `@Transactional` annotation at class level. By applying annotation, the transaction starts at the timing of method execution starts and transaction commits at the time of method execution successful completion. However, if unexpected exception occurs in between, the transaction roll-backs. If database is not used, `@Transactional` annotation is not required.
(3)	Inject `TodoRepository` implementation using `@Inject` annotation.
(4)	Logic of fetching single record is used in both delete and finish methods. Hence it should be implemented in a method (OK to make it public by declaring in the interface).

次のページに続く

Table. 10.1 – 前のページからの続き

Sr. No.	Description
(5)	Use `org.terasoluna.gfw.common.message.ResultMessage` provided in common library, as a class that stores result messages. Currently, for throwing error message, `ResultMessage` is added by specifying message type using `ResultMessages.error()` .
(6)	When targeted data not found, `org.terasoluna.gfw.common.exception.ResourceNotFoundException` provided in common library is thrown.
(7)	Regarding the reading process only, put `readOnly=true` . By this settings, the optimization of transaction control is done while reading process depending upon the O/R Mapper (Not effective in case of JPA used). If database is not used, `@Transactional` annotation is not required.
(8)	When business error occurs, `org.terasoluna.gfw.common.exception.BusinessException` provided in common library is thrown.
(9)	UUID is used to generate a unique value. DB sequence may be used.
(10)	If database is accessed using Spring Data JPA, instead of `create` method, call `save` method.
(11)	If database is accessed using Spring Data JPA, instead of `update` method, call `save` method.

Note: In this chapter, error messages are hard coded for simplification, but in reality it is not preferred from maintenance viewpoint. Usually, it is recommended to create message externally in property file. The method for creating the external property file is described in *Properties Management*.

Creation of JUnit for Service

Todo

TBD

For information about how to Unit test the service, planned to be described in next version later.

Creation of application layer

Since domain layer implementation is completed, use the domain layer to create application layer.

Creation of Controller

First create Controller that controls screen transition of todo business application.

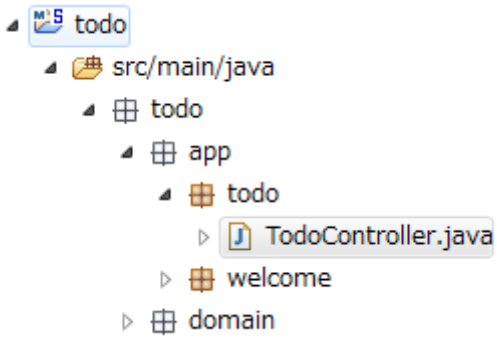
Right click on the Package Explorer, select -> [New] -> [Class] -> [New Java Class] dialog box appears,

Sr. No.	Item	Input value
1	Package	`todo.app.todo`
2	Name	`TodoController`

and click [Finish] after entering above information.

Note: It should be noted that the higher level package is different from the domain layer.

Created class stored in the following directory.




```
package todo.app.todo;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller // (1)
@RequestMapping("todo") // (2)
public class TodoController {

}
```

Sr. No.	Description
(1)	In order to make Controller as component-scan target, add `@Controller` annotation at class level.
(2)	In order to bring all screen transitions handled by `TodoController`, under `<contextPath>/todo`, set `@RequestMapping(“todo”)` at class level.

Show all TODO implementation

Perform below in screen that are created in this tutorial.

- Display new form
- Display all records of TODO

Form creation

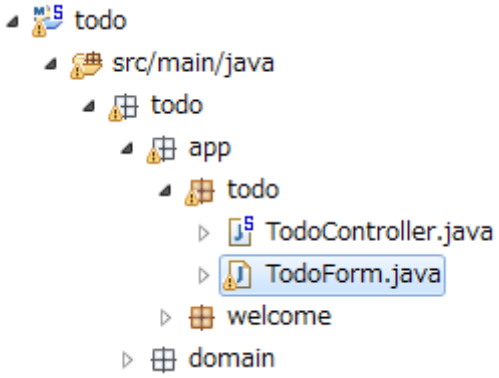
Create Form class (JavaBean).

Right click on the Package Explorer, select -> [New] -> [Class] -> [New Java Class] dialog box appears,

Sr. No.	Item	Input value
1	Package	`todo.app.todo`
2	Name	`TodoForm`
3	Interfaces	`java.io.Serializable`

and click [Finish] after entering above information.

Created class stored in the following directory.



Add following property in the created class.

- Title = todoTitle

```
package todo.app.todo;

import java.io.Serializable;

public class TodoForm implements Serializable {
    private static final long serialVersionUID = 1L;

    private String todoTitle;

    public String getTodoTitle() {
        return todoTitle;
    }

    public void setTodoTitle(String todoTitle) {
        this.todoTitle = todoTitle;
    }

}
```

Implementation of Controller

Add list screen display into `TodoController`.

```
package todo.app.todo;

import java.util.Collection;

import javax.inject.Inject;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;

import todo.domain.model.Todo;
import todo.domain.service.todo.TodoService;

@Controller
@RequestMapping("todo")
public class TodoController {
    @Inject // (1)
    TodoService todoService;

    @ModelAttribute // (2)
    public TodoForm setUpForm() {
        TodoForm form = new TodoForm();
        return form;
    }

    @RequestMapping(value = "list") // (3)
    public String list(Model model) {
        Collection<Todo> todos = todoService.findAll();
        model.addAttribute("todos", todos); // (4)
        return "todo/list"; // (5)
    }
}
```

Sr. No.	Description
(1)	Add `@Inject` annotation for injecting `TodoService` using DI container. Instance of type `TodoService` (instance of `TodoServiceImpl`) managed by DI container is injected.
(2)	Initialize Form By adding `@ModelAttribute` annotation, form object of the return value of this method is added to `Model` with name `"todoForm"`. It is same as executing `model.addAttribute("todoForm",form)` in each method of `TodoController`.
(3)	set `@RequestMapping` annotation such a way that method of list screen display (`list` method) gets executed when it is requested to `/todo/list` path. Since `@RequestMapping(“todo”)` is being set at class level, only `@RequestMapping("list")` is fine here.
(4)	Add Todo list to `Model` and pass to View.
(5)	If `"todo/list"` is returned as View name, `WEB-INF/views/todo/list.jsp` will be rendered using `ViewResolver` defined in spring-mvc.xml.

JSP creation

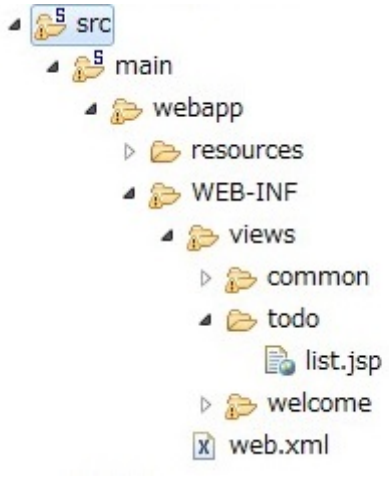
Display Model passed from Controller by creating JSP.

Right click on the Package Explorer, select -> [New] -> [File] -> [New File] dialog box appears,

Sr. No.	Item	Input value
1	Enter or select the parent folder	`todo/src/main/webapp/WEB-INF/views/todo`
2	File name	`list.jsp`

and click [Finish] after entering above information.

Created file stored in the following directory.



First, implement necessary JSP for following display.

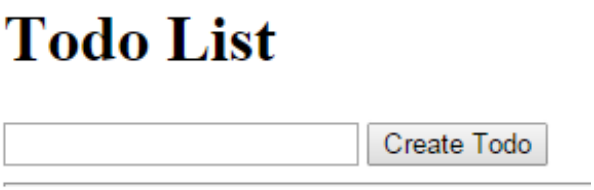
- Input form of TODO
- [Create Todo] button
- List display area of TODO

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Todo List</title>
<style type="text/css">
.strike {
    text-decoration: line-through;
}
</style>
</head>
<body>
    <h1>Todo List</h1>
    <div id="todoForm">
        <!-- (1) -->
        <form:form
           action="${pageContext.request.contextPath}/todo/create"
            method="post" modelAttribute="todoForm">
            <!-- (2) -->
            <form:input path="todoTitle" />
            <form:button>Create Todo</form:button>
        </form:form>
    </div>
    <hr />
```

```
    <div id="todoList">
        <ul>
            <!-- (3) -->
            <c:forEach items="${todos}" var="todo">
                <li><c:choose>
                        <c:when test="${todo.finished}"><!-- (4) -->
                            <span class="strike">
                            <!-- (5) -->
                            ${f:h(todo.todoTitle)}
                            </span>
                        </c:when>
                        <c:otherwise>
                            ${f:h(todo.todoTitle)}
                         </c:otherwise>
                    </c:choose></li>
            </c:forEach>
        </ul>
    </div>
</body>
</html>
```

Sr. No.	Description
(1)	Display the form of new creation process. The `<form:form>` tag is used for displaying form. Specify name of the form object added to `Model` by Controller in `modelAttribute` attribute. Specify URL(`<contextPath>/todo/create`) into `action` attribute for running the new creation process. Since the new creation process is a process of updation, specify the `POST` method into the `method` attribute. <contextPath> to be specified in `action` attribute can be fetched by `${pageContext.request.contextPath}`.
(2)	Bind form property using `<form:input>` tag. Property name of form which is specified in `modelAttribute` attribute should match with the value of `path` attribute.
(3)	Display entire list of Todo using `<c:forEach>` tag.
(4)	Determine whether to decorate text using strike through(`text-decoration: line-through;`) to display, if it is completed (`finished`).
(5)	**To take XSS countermeasures at the time of output of character string, HTML escape should be performed using f:h() function.** Regarding XSS measures, refer to *XSS Countermeasures*.

Right click [todo] project in STS and start Web application by [Run As] -> [Run on Server]. If http://localhost:8080/todo/todo/list is accessed in browser, the following screen gets displayed.



Create TODO

Next, implement a new creation logic after clicking [Create TODO] button on List display screen.

Modifications in Controller

Add new creation process into `TodoController`.

```
package todo.app.todo;

import java.util.Collection;

import javax.inject.Inject;
import javax.validation.Valid;

import org.dozer.Mapper;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.servlet.mvc.support.RedirectAttributes;
import org.terasoluna.gfw.common.exception.BusinessException;
import org.terasoluna.gfw.common.message.ResultMessage;
import org.terasoluna.gfw.common.message.ResultMessages;

import todo.domain.model.Todo;
import todo.domain.service.todo.TodoService;

@Controller
@RequestMapping("todo")
public class TodoController {
    @Inject
    TodoService todoService;
```

```
    // (1)
    @Inject
    Mapper beanMapper;

    @ModelAttribute
    public TodoForm setUpForm() {
        TodoForm form = new TodoForm();
        return form;
    }

    @RequestMapping(value = "list")
    public String list(Model model) {
        Collection<Todo> todos = todoService.findAll();
        model.addAttribute("todos", todos);
        return "todo/list";
    }

    @RequestMapping(value = "create", method = RequestMethod.POST) // (2)
    public String create(@Valid TodoForm todoForm, BindingResult bindingResult, //␣
↪(3)
            Model model, RedirectAttributes attributes) { // (4)

        // (5)
        if (bindingResult.hasErrors()) {
            return list(model);
        }

        // (6)
        Todo todo = beanMapper.map(todoForm, Todo.class);

        try {
            todoService.create(todo);
        } catch (BusinessException e) {
            // (7)
            model.addAttribute(e.getResultMessages());
            return list(model);
        }

        // (8)
        attributes.addFlashAttribute(ResultMessages.success().add(
                ResultMessage.fromText("Created successfully!")));
        return "redirect:/todo/list";
    }

}
```

Sr. No.	Description
(1)	At the time of converting form object into domain object. Inject the `Mapper` interface of Dozer.
(2)	Set `@RequestMapping` annotation such a way that method of new creation process (`create` method) gets executed when it is requested to `/todo/create` path using `POST` method.
(3)	For performing input validation of form, add `@Valid` annotation to form argument. Input validation result is stored in the immediate next argument `BindingResult`.
(4)	Display the list screen by redirecting after successful completion. Add `RedirectAttributes` into argument for storing the information to be redirected.
(5)	Return to list screen in case of input error. Re-execute `list` method as it is necessary to fetch all records of Todo again.
(6)	Create `Todo` object from `TodoForm` object using `Mapper` interface of Dozer. No need to set if the property name of conversion source and destination is the same. There is no merit in using `Mapper` interface of Dozer to convert only `todoTitle` property, but it is very convenient in case of multiple properties.
(7)	In case of `BusinessException` while executing business logic, add the result message to `Model` and return to list screen.
(8)	Since it is created successfully, add the result message to flash scope and redirect to list screen. Since redirect is used, there is no case of browser being read again and a new registration process being `POST`. (For details, refer to "*About PRG (Post-Redirect-Get) pattern*") Since this time Created successfully message is displayed, `ResultMessages.success()` is used.

Modifications in Form

To define input validation rules, add annotation to the form object.

```
package todo.app.todo;

import java.io.Serializable;

import javax.validation.constraints.NotNull;
import javax.validation.constraints.Size;

public class TodoForm implements Serializable {
    private static final long serialVersionUID = 1L;

    @NotNull // (1)
    @Size(min = 1, max = 30) // (2)
    private String todoTitle;

    public String getTodoTitle() {
        return todoTitle;
    }

    public void setTodoTitle(String todoTitle) {
        this.todoTitle = todoTitle;
    }
}
```

Sr. No.	Description
(1)	Enable mandatory check using `@NotNull` annotation.
(2)	Enable string length check using `@Size` annotation.

Modifications in JSP

Add the tag for displaying the result message and input check error.

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Todo List</title>
<style type="text/css">
.strike {
    text-decoration: line-through;
```

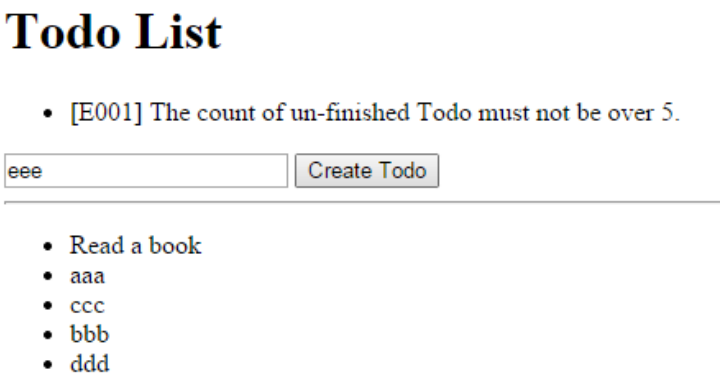
```
}
</style>
</head>
<body>
    <h1>Todo List</h1>
    <div id="todoForm">
        <!-- (1) -->
        <t:messagesPanel />

        <form:form
           action="${pageContext.request.contextPath}/todo/create"
            method="post" modelAttribute="todoForm">
            <form:input path="todoTitle" />
            <form:errors path="todoTitle" /><!-- (2) -->
            <form:button>Create Todo</form:button>
        </form:form>
    </div>
    <hr />
    <div id="todoList">
        <ul>
            <c:forEach items="${todos}" var="todo">
                <li><c:choose>
                        <c:when test="${todo.finished}">
                            <span style="text-decoration: line-through;">
                            ${f:h(todo.todoTitle)}
                            </span>
                        </c:when>
                        <c:otherwise>
                            ${f:h(todo.todoTitle)}
                         </c:otherwise>
                    </c:choose></li>
            </c:forEach>
        </ul>
    </div>
</body>
</html>
```

Sr. No.	Description
(1)	Display result message using <t:messagesPanel> tag.
(2)	Display errors in case of input error using <form:errors> tag. Match the value of path attribute with <form:input> tag.

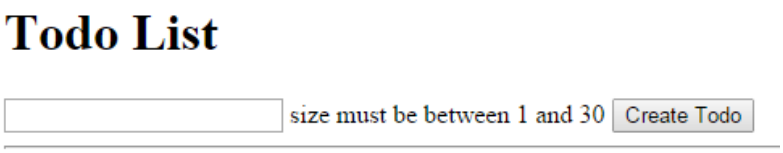
If form is submitted by entering appropriate value in the form, success message is displayed as given below.



When 6 or more records are registered and business error occurs, error message is displayed.



If form is submitted by entering null character, the following error message is displayed.



Customize message display

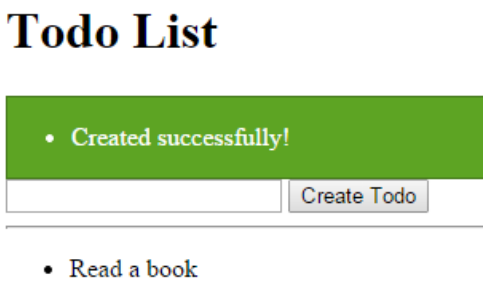
If `<t:messagesPanel>` is used, following is the HTML output.

```
<div class="alert alert-success"><ul><li>Created successfully!</li></ul></div>
```

With the following modifications in style sheet (in `<style>` tag of `list.jsp`), customize appearance of the result message.

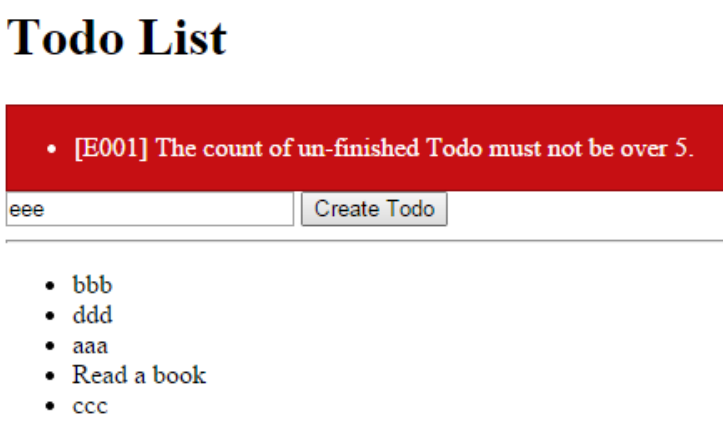
```
.alert {
    border: 1px solid;
}

.alert-error {
    background-color: #c60f13;
    border-color: #970b0e;
    color: white;
}

.alert-success {
    background-color: #5da423;
    border-color: #457a1a;
    color: white;
}
```

The message is as follows.



Moreover, input error message class can be specified to `cssClass` attribute of `<form:errors>` tag.

Modify JSP as follows,

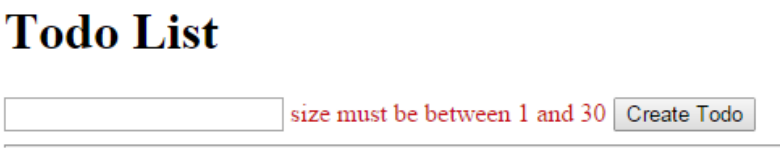



```
<form:errors path="todoTitle" cssClass="text-error" />
```

and add the following to style sheet.

```
.text-error {
    color: #c60f13;
}
```

Input error message is as follows.



Finish TODO

Add [Finish] button to List display screen and add completion process of TODO.

Modifications in Form

About the completion Form, same `TodoForm` is used.

The `todoId` property needs to be added to `TodoForm` but if simply added, `todoId` property check rules for new creation are applied as it is. For specifying separate rules for new creation and completion in a single Form, set `groups` attribute and perform input check rule group.

Add below property in Form class.

- ID → todoId

```
package todo.app.todo;

import java.io.Serializable;

import javax.validation.constraints.NotNull;
import javax.validation.constraints.Size;

public class TodoForm implements Serializable {
    // (1)
    public static interface TodoCreate {
    };

    public static interface TodoFinish {
    };

    private static final long serialVersionUID = 1L;

    // (2)
    @NotNull(groups = { TodoFinish.class })
    private String todoId;

    // (3)
    @NotNull(groups = { TodoCreate.class })
    @Size(min = 1, max = 30, groups = { TodoCreate.class })
    private String todoTitle;

    public String getTodoId() {
        return todoId;
    }

    public void setTodoId(String todoId) {
        this.todoId = todoId;
    }

    public String getTodoTitle() {
        return todoTitle;
    }

    public void setTodoTitle(String todoTitle) {
        this.todoTitle = todoTitle;
    }

}
```

Sr. No.	Description
(1)	Create an interface for grouping the input check rules. For grouping the input check rules, Refer *Input Validation*. However as a new creation prcess `TodoCreate` interface and as a completion process `TodoFinish` interface is created.
(2)	`todoId` is a property for completion process. Therefore in `groups` attribute of `@NotNull` annotation, the `TodoFinish` interface is specified for indicating input check rule of completion process.
(3)	`todoTitle` is a property for new creation process. Therefore in `groups` attribute of `@NotNull` annotation and `@Size` annotation, the `TodoCreate` interface is specified for indicating input check rule of new creation process.

Modifications in Controller

Add completion process logic to `TodoController`.

Take precaution of **using @Validated instead of @Valid** for executing the group validation.

```
package todo.app.todo;

import java.util.Collection;

import javax.inject.Inject;
import javax.validation.groups.Default;

import org.dozer.Mapper;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.BindingResult;
import org.springframework.validation.annotation.Validated;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.servlet.mvc.support.RedirectAttributes;
import org.terasoluna.gfw.common.exception.BusinessException;
import org.terasoluna.gfw.common.message.ResultMessage;
import org.terasoluna.gfw.common.message.ResultMessages;
```

```
import todo.app.todo.TodoForm.TodoCreate;
import todo.app.todo.TodoForm.TodoFinish;
import todo.domain.model.Todo;
import todo.domain.service.todo.TodoService;

@Controller
@RequestMapping("todo")
public class TodoController {
    @Inject
    TodoService todoService;

    @Inject
    Mapper beanMapper;

    @ModelAttribute
    public TodoForm setUpForm() {
        TodoForm form = new TodoForm();
        return form;
    }

    @RequestMapping(value = "list")
    public String list(Model model) {
        Collection<Todo> todos = todoService.findAll();
        model.addAttribute("todos", todos);
        return "todo/list";
    }

    @RequestMapping(value = "create", method = RequestMethod.POST)
    public String create(
            @Validated({ Default.class, TodoCreate.class }) TodoForm todoForm, //␣
↪(1)
            BindingResult bindingResult, Model model,
            RedirectAttributes attributes) {

        if (bindingResult.hasErrors()) {
            return list(model);
        }

        Todo todo = beanMapper.map(todoForm, Todo.class);

        try {
            todoService.create(todo);
        } catch (BusinessException e) {
            model.addAttribute(e.getResultMessages());
            return list(model);
        }

        attributes.addFlashAttribute(ResultMessages.success().add(
                ResultMessage.fromText("Created successfully!")));
```

```
        return "redirect:/todo/list";
    }

    @RequestMapping(value = "finish", method = RequestMethod.POST) // (2)
    public String finish(
            @Validated({ Default.class, TodoFinish.class }) TodoForm form, // (3)
            BindingResult bindingResult, Model model,
            RedirectAttributes attributes) {
        // (4)
        if (bindingResult.hasErrors()) {
            return list(model);
        }

        try {
            todoService.finish(form.getTodoId());
        } catch (BusinessException e) {
            // (5)
            model.addAttribute(e.getResultMessages());
            return list(model);
        }

        // (6)
        attributes.addFlashAttribute(ResultMessages.success().add(
                ResultMessage.fromText("Finished successfully!")));
        return "redirect:/todo/list";
    }
}
```

Sr. No.	Description
(1)	Change `@Valid` to `@Validated` for executing group validation. Group of input check rules (group interface) can be specified in `value` attribute. `Default.class` is a group interface provided to apply an input validation rules when group is not specified.
(2)	Set `@RequestMapping` annotation such a way that method of completion process (`finish` method) gets executed when it is requested to `/todo/finish` path using `POST` method.
(3)	Specify the group interface (`TodoFinish` interface) for Finish processing as group of input check.
(4)	In case of input error, return to list screen.
(5)	In case of `BusinessException` while executing business logic, add the result message to `Model` and return to list screen.
(6)	Since it is created successfully, add the result message to flash scope and redirect to list screen.

Note: Separate Form can also be created for Create and Finish. In case of separate Form class, there is no need to group the input check rules therefore definition of input check rules will be simple.

However, as the number of Form class increase,

- The number of classes increases
- Not possible to centrally manage the input check rules due to duplicate properties increases

Therefore, please note that when the specifications changes, the modification cost will also be more.

Moreover, if multiple Form objects are initialized by `@ModelAttribute` method, unnecessary instance gets generated because every time all Forms are being initialized.

Modifications in JSP

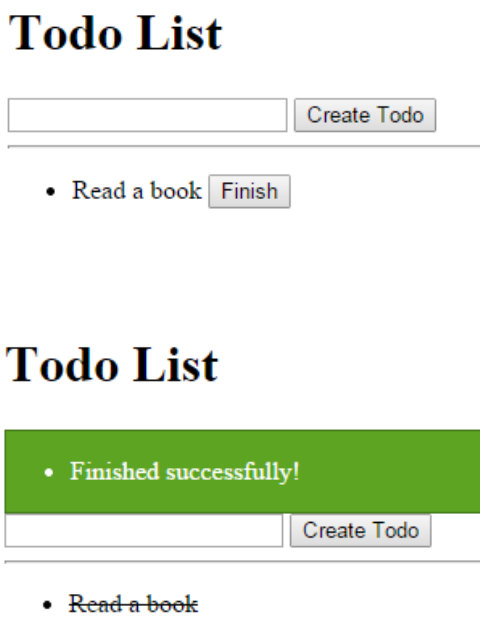
Add completion process form.

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Todo List</title>
</head>
<style type="text/css">
.strike {
    text-decoration: line-through;
}

.alert {
    border: 1px solid;
}

.alert-error {
    background-color: #c60f13;
    border-color: #970b0e;
    color: white;
}

.alert-success {
    background-color: #5da423;
    border-color: #457a1a;
    color: white;
}

.text-error {
    color: #c60f13;
}
</style>
<body>
    <h1>Todo List</h1>

    <div id="todoForm">
        <t:messagesPanel />

        <form:form
            action="${pageContext.request.contextPath}/todo/create"
            method="post" modelAttribute="todoForm">
            <form:input path="todoTitle" />
            <form:errors path="todoTitle" cssClass="text-error" />
            <form:button>Create Todo</form:button>
        </form:form>
    </div>
    <hr />
```

```
    <div id="todoList">
        <ul>
            <c:forEach items="${todos}" var="todo">
                <li><c:choose>
                        <c:when test="${todo.finished}">
                            <span class="strike">${f:h(todo.todoTitle)}</span>
                        </c:when>
                        <c:otherwise>
                            ${f:h(todo.todoTitle)}
                            <!-- (1) -->
                            <form:form
                                action="${pageContext.request.contextPath}/todo/
↪finish"
                                method="post"
                                modelAttribute="todoForm"
                                cssStyle="display: inline-block;">
                                <!-- (2) -->
                                <form:hidden path="todoId"
                                    value="${f:h(todo.todoId)}" />
                                <form:button>Finish</form:button>
                            </form:form>
                        </c:otherwise>
                    </c:choose></li>
            </c:forEach>
        </ul>
    </div>
</body>
</html>
```

Sr. No.	Description
(1)	Display the form for sending the request to complete the TODO if there are incomplete Todo. Specify URL(`<contextPath>/todo/finish`) into `action` attribute for running the completion process. Since the completion process is a process of updating, specify the `POST` method into the `method` attribute.
(2)	Pass `todoId` as request parameter using `<form:hidden>` tag. Also while setting the value in `value` attribute, **HTML escaping should always be performed using f:h() function.**

When pressing the [Finish] button after newly creating Todo, strike-through is shown as below and it can be understood that the operation is completed.



Delete TODO

Add [Delete] button on the list display screen and add the deletion process for TODO removal.

Modification in Form

Create `TodoForm` for deletion form.

```
package todo.app.todo;

import java.io.Serializable;

import javax.validation.constraints.NotNull;
import javax.validation.constraints.Size;

public class TodoForm implements Serializable {
    public static interface TodoCreate {
    };

    public static interface TodoFinish {
    };

    // (1)
    public static interface TodoDelete {
    }
```

```
    private static final long serialVersionUID = 1L;

    // (2)
    @NotNull(groups = { TodoFinish.class, TodoDelete.class })
    private String todoId;

    @NotNull(groups = { TodoCreate.class })
    @Size(min = 1, max = 30, groups = { TodoCreate.class })
    private String todoTitle;

    public String getTodoId() {
        return todoId;
    }

    public void setTodoId(String todoId) {
        this.todoId = todoId;
    }

    public String getTodoTitle() {
        return todoTitle;
    }

    public void setTodoTitle(String todoTitle) {
        this.todoTitle = todoTitle;
    }

}
```

Sr. No.	Description
(1)	Create the `TodoDelete` interface for deletion processing as group of input check rule.
(2)	`todoId` property is used in deletion process. Therefore, in the `groups` attribute of `@NotNull` annotation of `todoId`, specify the `TodoDelete` interface indicating that it is an input validation rules for the deletion process.

Modifications in Controller

Add the logic for delete processing to `TodoController`. It is almost same as the completion process.

```
package todo.app.todo;

import java.util.Collection;
```

```
import javax.inject.Inject;
import javax.validation.groups.Default;

import org.dozer.Mapper;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.BindingResult;
import org.springframework.validation.annotation.Validated;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.servlet.mvc.support.RedirectAttributes;
import org.terasoluna.gfw.common.exception.BusinessException;
import org.terasoluna.gfw.common.message.ResultMessage;
import org.terasoluna.gfw.common.message.ResultMessages;

import todo.app.todo.TodoForm.TodoDelete;
import todo.app.todo.TodoForm.TodoCreate;
import todo.app.todo.TodoForm.TodoFinish;
import todo.domain.model.Todo;
import todo.domain.service.todo.TodoService;

@Controller
@RequestMapping("todo")
public class TodoController {
    @Inject
    TodoService todoService;

    @Inject
    Mapper beanMapper;

    @ModelAttribute
    public TodoForm setUpForm() {
        TodoForm form = new TodoForm();
        return form;
    }

    @RequestMapping(value = "list")
    public String list(Model model) {
        Collection<Todo> todos = todoService.findAll();
        model.addAttribute("todos", todos);
        return "todo/list";
    }

    @RequestMapping(value = "create", method = RequestMethod.POST)
    public String create(
            @Validated({ Default.class, TodoCreate.class }) TodoForm todoForm,
            BindingResult bindingResult, Model model,
            RedirectAttributes attributes) {
```

```
        if (bindingResult.hasErrors()) {
            return list(model);
        }

        Todo todo = beanMapper.map(todoForm, Todo.class);

        try {
            todoService.create(todo);
        } catch (BusinessException e) {
            model.addAttribute(e.getResultMessages());
            return list(model);
        }

        attributes.addFlashAttribute(ResultMessages.success().add(
                ResultMessage.fromText("Created successfully!")));
        return "redirect:/todo/list";
    }

    @RequestMapping(value = "finish", method = RequestMethod.POST)
    public String finish(
            @Validated({ Default.class, TodoFinish.class }) TodoForm form,
            BindingResult bindingResult, Model model,
            RedirectAttributes attributes) {
        if (bindingResult.hasErrors()) {
            return list(model);
        }

        try {
            todoService.finish(form.getTodoId());
        } catch (BusinessException e) {
            model.addAttribute(e.getResultMessages());
            return list(model);
        }

        attributes.addFlashAttribute(ResultMessages.success().add(
                ResultMessage.fromText("Finished successfully!")));
        return "redirect:/todo/list";
    }

    @RequestMapping(value = "delete", method = RequestMethod.POST) // (1)
    public String delete(
            @Validated({ Default.class, TodoDelete.class }) TodoForm form,
            BindingResult bindingResult, Model model,
            RedirectAttributes attributes) {

        if (bindingResult.hasErrors()) {
            return list(model);
        }
```

```
        try {
            todoService.delete(form.getTodoId());
        } catch (BusinessException e) {
            model.addAttribute(e.getResultMessages());
            return list(model);
        }

        attributes.addFlashAttribute(ResultMessages.success().add(
                ResultMessage.fromText("Deleted successfully!")));
        return "redirect:/todo/list";
    }

}
```

Sr. No.	Description
(1)	Set `@RequestMapping` annotation such a way that method of deletion process (`delete` method) gets executed when it is requested to `/todo/delete` path using `POST` method.

Modifications in JSP

Add deletion process form.

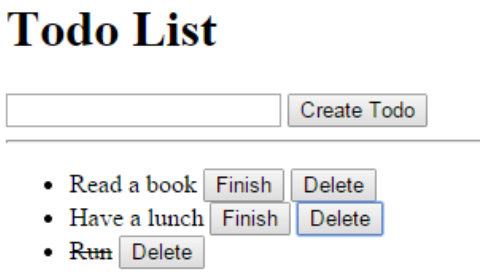
```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Todo List</title>
</head>
<style type="text/css">
.strike {
    text-decoration: line-through;
}

.alert {
    border: 1px solid;
}

.alert-error {
    background-color: #c60f13;
    border-color: #970b0e;
    color: white;
}

.alert-success {
    background-color: #5da423;
    border-color: #457a1a;
    color: white;
```

```
}

.text-error {
    color: #c60f13;
}
</style>
<body>
    <h1>Todo List</h1>

    <div id="todoForm">
        <t:messagesPanel />

        <form:form
            action="${pageContext.request.contextPath}/todo/create"
            method="post" modelAttribute="todoForm">
            <form:input path="todoTitle" />
            <form:errors path="todoTitle" cssClass="text-error" />
            <form:button>Create Todo</form:button>
        </form:form>
    </div>
    <hr />
    <div id="todoList">
        <ul>
            <c:forEach items="${todos}" var="todo">
                <li><c:choose>
                        <c:when test="${todo.finished}">
                            <span class="strike">${f:h(todo.todoTitle)}</span>
                        </c:when>
                        <c:otherwise>
                            ${f:h(todo.todoTitle)}
                            <form:form
                                action="${pageContext.request.contextPath}/todo/
↪finish"
                                method="post"
                                modelAttribute="todoForm"
                                cssStyle="display: inline-block;">
                                <form:hidden path="todoId"
                                    value="${f:h(todo.todoId)}" />
                                <form:button>Finish</form:button>
                            </form:form>
                        </c:otherwise>
                    </c:choose>
                    <!-- (1) -->
                    <form:form
                        action="${pageContext.request.contextPath}/todo/delete"
                        method="post" modelAttribute="todoForm"
                        cssStyle="display: inline-block;">
                        <!-- (2) -->
                        <form:hidden path="todoId"
                            value="${f:h(todo.todoId)}" />
```

```
                        <form:button>Delete</form:button>
                    </form:form>
                </li>
            </c:forEach>
        </ul>
    </div>
</body>
</html>
```

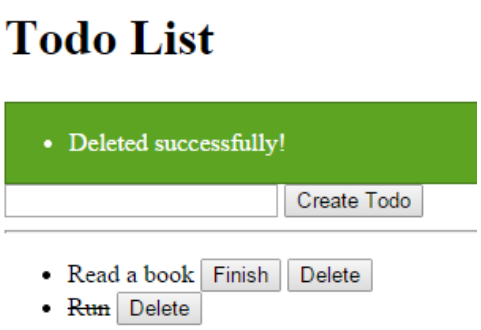
Sr. No.	Description
(1)	Display deletion process form. Specify URL(`<contextPath>/todo/delete`) into `action` attribute for running the deletion process. Since the deletion process is a process of updating, specify the `POST` method into the `method` attribute.
(2)	Pass `todoId` as request parameter using `<form:hidden>` tag. Also while setting the value in `value` attribute, **HTML escaping should always be performed using f:h() function.**

When pressing the [Delete] button in an uncompleted TODO state, TODO is deleted as follows.



Use of CSS file

Although style sheets are directly defined in a JSP file, generally it is defined in the CSS file while developing the actual application.



Here, how to define style sheet into the CSS file are described.

Add the style sheet definitions in the CSS file (`src/main/webapp/resources/app/css/styles.css`) that is provided in the blank project.

```
/* ... */

.strike {
    text-decoration: line-through;
}

.alert {
    border: 1px solid;
    margin-bottom: 5px;
}

.alert-error {
    background-color: #c60f13;
    border-color: #970b0e;
    color: white;
}

.alert-success {
    background-color: #5da423;
    border-color: #457a1a;
    color: white;
}

.text-error {
    color: #c60f13;
}

.alert ul {
    margin: 15px 0px 15px 0px;
}

#todoList li {
    margin-top: 5px;
}
```

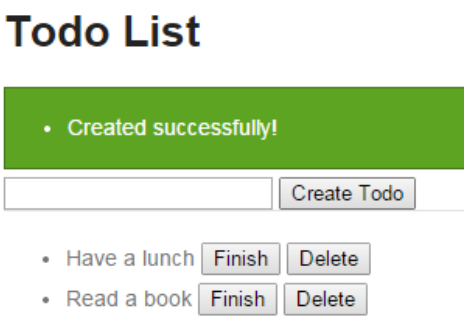
Loading CSS file within JSP.

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Todo List</title>
<!-- (1) -->
<link rel="stylesheet" href="${pageContext.request.contextPath}/resources/app/css/
↪styles.css" type="text/css">
</head>
<body>
    <h1>Todo List</h1>

    <div id="todoForm">
        <t:messagesPanel />

        <form:form
            action="${pageContext.request.contextPath}/todo/create"
            method="post" modelAttribute="todoForm">
            <form:input path="todoTitle" />
            <form:errors path="todoTitle" cssClass="text-error" />
            <form:button>Create Todo</form:button>
        </form:form>
    </div>
    <hr />
    <div id="todoList">
        <ul>
            <c:forEach items="${todos}" var="todo">
                <li><c:choose>
                        <c:when test="${todo.finished}">
                            <span class="strike">${f:h(todo.todoTitle)}</span>
                        </c:when>
                        <c:otherwise>
                            ${f:h(todo.todoTitle)}
                            <form:form
                                action="${pageContext.request.contextPath}/todo/
↪finish"
                                method="post"
                                modelAttribute="todoForm"
                                cssStyle="display: inline-block;">
                                <form:hidden path="todoId"
                                    value="${f:h(todo.todoId)}" />
```

```
                                <form:button>Finish</form:button>
                            </form:form>
                        </c:otherwise>
                    </c:choose>
                    <form:form
                        action="${pageContext.request.contextPath}/todo/delete"
                        method="post" modelAttribute="todoForm"
                        cssStyle="display: inline-block;">
                        <form:hidden path="todoId"
                            value="${f:h(todo.todoId)}" />
                        <form:button>Delete</form:button>
                    </form:form>
                </li>
            </c:forEach>
        </ul>
    </div>
</body>
</html>
```

Sr. No.	Description
(1)	Delete the style sheet definitions from the JSP file and load the CSS file in which the style sheets are defined.

Following would be the layout if CSS file is applied.



10.1.5 Creating infrastructure layer with a Database access

In this section, infrastructure layer for persisting Domain objects in the database is explained.

In this tutorial, it explains how to implement the infrastructure layer using following two O/R Mapper.

- MyBatis3
- Spring Data JPA

Creating a blank project dependent on O/R Mapper

Here, a blank project dependent on O/R Mapper is created.

At first, recreate the project corresponding to O/R Mapper being used.

- *Creating blank project for MyBatis3*
- *Creating blank project for JPA*

Next, **copy a file other than **TodoRepositoryImpl class to a newly created project**under `src`folder created upto *Creating infrastructure layer with a Database access*.

However, file to be copied must be a newly created file or file with added changes. A file without modifications must not be copied.

Database set-up

Here, perform the Database set-up.

In this tutorial, the H2 Database is used to save the database setup time.

Modification in todo-infra.properties

Modify the `src/main/resources/META-INF/spring/todo-infra.properties` settings for creating tables into H2 Database while booting the AP server.

```
database=H2
# (1)
```

```
database.url=jdbc:h2:mem:todo;DB_CLOSE_DELAY=-1;INIT=create table if not exists␣
↪todo(todo_id varchar(36) primary key, todo_title varchar(30), finished boolean,␣
↪created_at timestamp)
database.username=sa
database.password=
database.driverClassName=org.h2.Driver
# connection pool
cp.maxActive=96
cp.maxIdle=16
cp.minIdle=0
cp.maxWait=60000
```

Sr. No.	Description
(1)	Specify the DDL statements to create tables into INIT parameter of the URL connection.

Note: If you format the DDL statement that is set to INIT parameter, it will looks like follows.

```
create table if not exists todo (
    todo_id varchar(36) primary key,
    todo_title varchar(30),
    finished boolean,
    created_at timestamp
)
```

Creating infrastructure layer with MyBatis3

Here, How to create RepositoryImpl of infrastructure layer using MyBatis3 is explained.

If you want to use the Spring Data JPA, you can skip this section and may proceed to the *Creating infrastructure layer with Spring Data JPA*.

Create TodoRepository

`TodoRepository` is created by the same way as it is created for without O/R Mapper. For creation method, refer [*Repository creation*].

Create TodoRepositoryImpl

If MyBatis3 is used, RepositoryImpl is automatically generated from the Repository interface (Mapper interface). Therefore, the creation of `TodoRepositoryImpl` is not required. Remove if it is created.

Create Mapper file

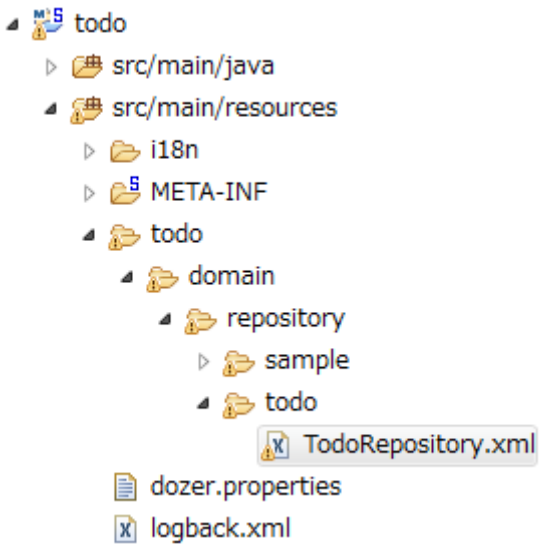
Create a Mapper file for defining SQL to be executed when the `TodoRepository` interface methods are called.

Right click on the Package Explorer, select -> [New] -> [File] -> [New File] dialog box appears,

Sr. No.	Item	Input value
1	Enter or select the parent folder	`todo/src/main/resources/todo/domain/repository/to`
2	File name	`TodoRepository.xml`

and click [Finish] after entering above information.

Created file stored in the following directory.



Describe the SQL to be executed when the `TodoRepository` methods defined in the interfaces are called.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
    "http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<!-- (1) -->
<mapper namespace="todo.domain.repository.todo.TodoRepository">

    <!-- (2) -->
    <resultMap id="todoResultMap" type="Todo">
        <id property="todoId" column="todo_id" />
        <result property="todoTitle" column="todo_title" />
```

```
        <result property="finished" column="finished" />
        <result property="createdAt" column="created_at" />
    </resultMap>

    <!-- (3) -->
    <select id="findOne" parameterType="String" resultMap="todoResultMap">
    <![CDATA[
        SELECT
            todo_id,
            todo_title,
            finished,
            created_at
        FROM
            todo
        WHERE
            todo_id = #{todoId}
    ]]>
    </select>

    <!-- (4) -->
    <select id="findAll" resultMap="todoResultMap">
    <![CDATA[
        SELECT
            todo_id,
            todo_title,
            finished,
            created_at
        FROM
            todo
    ]]>
    </select>

    <!-- (5) -->
    <insert id="create" parameterType="Todo">
    <![CDATA[
        INSERT INTO todo
        (
            todo_id,
            todo_title,
            finished,
            created_at
        )
        VALUES
        (
            #{todoId},
            #{todoTitle},
            #{finished},
            #{createdAt}
        )
    ]]>
```

```
    </insert>

    <!-- (6) -->
    <update id="update" parameterType="Todo">
    <![CDATA[
        UPDATE todo
        SET
            todo_title = #{todoTitle},
            finished = #{finished},
            created_at = #{createdAt}
        WHERE
            todo_id = #{todoId}
    ]]>
    </update>

    <!-- (7) -->
    <delete id="delete" parameterType="Todo">
    <![CDATA[
        DELETE FROM
            todo
        WHERE
            todo_id = #{todoId}
    ]]>
    </delete>

    <!-- (8) -->
    <select id="countByFinished" parameterType="Boolean"
        resultType="Long">
    <![CDATA[
        SELECT
            COUNT(*)
        FROM
            todo
        WHERE
            finished = #{finished}
    ]]>
    </select>

</mapper>
```

Sr. No.	Description
(1)	Specify the fully qualified class name of the Repository interfaces (FQCN) in the `namespace` attribute of `mapper` element.
(2)	Define JavaBean mapping with search result (`ResultSet`) in the `<resultMap>` element. For the mapping file details, Refer *Database Access (MyBatis3)*.
(3)	Implement the SQL to get one record which matches with the `todoId` (PK). Specify the ID of applicable mapping definition in the `resultMap` attribute of `<select>` element.
(4)	Implement the SQL to retrieve all records. Specify the ID of applicable mapping definition in the `resultMap` attribute of `<select>` element. Although it is not described in the application requirement, records are rearranged as the most recent TODO is displayed at the top.
(5)	Implement the SQL to insert the Todo object specified in the argument. Specify the parameter of the class name (FQCN or alias name) in the `parameterType` attribute of `<insert>` element.
(6)	Implement the SQL to update the Todo object specified in the argument. Specify the parameter of the class name (FQCN or alias name) in the `parameterType` attribute of `<update>` element.
(7)	Implement the SQL to delete the Todo object specified in the argument. Specify the parameter of the class name (FQCN or alias name) in the `parameterType` attribute of `<delete>` element.
(8)	Implement the SQL to get all records matches with the `finished` status specified in the argument.

Since the creation of infrastructure layer using MyBatis3 has been completed, create the application layer components and services.

Once creation of the Services and application layer are completed, execute the Todo application after starting the AP server, SQL and transaction log output is as following.

```
date:2016-02-17 13:18:54    thread:tomcat-http--5   X-Track:
↪390066c43aa94b6588e5bac6a54812b2        level:TRACE     logger:o.t.gfw.web.
↪logging.TraceLoggingInterceptor      message:[START CONTROLLER] TodoController.
↪list(Model)
date:2016-02-17 13:18:54    thread:tomcat-http--5   X-Track:
↪390066c43aa94b6588e5bac6a54812b2        level:DEBUG     logger:o.s.jdbc.
↪datasource.DataSourceTransactionManager message:Creating new transaction with␣
↪name [todo.domain.service.todo.TodoServiceImpl.findAll]: PROPAGATION_REQUIRED,
↪ISOLATION_DEFAULT,readOnly; ''
date:2016-02-17 13:18:55    thread:tomcat-http--5   X-Track:
↪390066c43aa94b6588e5bac6a54812b2        level:DEBUG     logger:o.s.jdbc.
↪datasource.DataSourceTransactionManager message:Acquired Connection [net.sf.
↪log4jdbc.ConnectionSpy@4e53de7c] for JDBC transaction
date:2016-02-17 13:18:55    thread:tomcat-http--5   X-Track:
↪390066c43aa94b6588e5bac6a54812b2        level:DEBUG     logger:t.domain.
↪repository.todo.TodoRepository.findAll  message:==>  Preparing: SELECT todo_id,␣
↪todo_title, finished, created_at FROM todo
date:2016-02-17 13:18:55    thread:tomcat-http--5   X-Track:
↪390066c43aa94b6588e5bac6a54812b2        level:DEBUG     logger:t.domain.
↪repository.todo.TodoRepository.findAll  message:==> Parameters:
date:2016-02-17 13:18:55    thread:tomcat-http--5   X-Track:
↪390066c43aa94b6588e5bac6a54812b2        level:DEBUG     logger:jdbc.sqltiming
                             message: sun.reflect.NativeMethodAccessorImpl.
↪invoke0(NativeMethodAccessorImpl.java:-2)
1. SELECT
            todo_id,
            todo_title,
            finished,
            created_at
        FROM
            todo {executed in 0 msec}
date:2016-02-17 13:18:55    thread:tomcat-http--5   X-Track:
↪390066c43aa94b6588e5bac6a54812b2        level:INFO      logger:jdbc.
↪resultsettable                              message:|--------|-----------|-------
↪--|-----------|
date:2016-02-17 13:18:55    thread:tomcat-http--5   X-Track:
↪390066c43aa94b6588e5bac6a54812b2        level:INFO      logger:jdbc.
↪resultsettable                              message:|TODO_ID |TODO_TITLE␣
↪|FINISHED |CREATED_AT |
date:2016-02-17 13:18:55    thread:tomcat-http--5   X-Track:
↪390066c43aa94b6588e5bac6a54812b2        level:INFO      logger:jdbc.
↪resultsettable                              message:|--------|-----------|-------
↪--|-----------|
```

```
date:2016-02-17 13:18:55    thread:tomcat-http--5   X-Track:
↪390066c43aa94b6588e5bac6a54812b2        level:INFO      logger:jdbc.
↪resultsettable                              message:|--------|-----------|-------
↪--|-----------|
date:2016-02-17 13:18:55    thread:tomcat-http--5   X-Track:
↪390066c43aa94b6588e5bac6a54812b2        level:DEBUG     logger:t.domain.
↪repository.todo.TodoRepository.findAll  message:<==      Total: 0
date:2016-02-17 13:18:55    thread:tomcat-http--5   X-Track:
↪390066c43aa94b6588e5bac6a54812b2        level:DEBUG     logger:o.s.jdbc.
↪datasource.DataSourceTransactionManager message:Initiating transaction commit
date:2016-02-17 13:18:55    thread:tomcat-http--5   X-Track:
↪390066c43aa94b6588e5bac6a54812b2        level:DEBUG     logger:o.s.jdbc.
↪datasource.DataSourceTransactionManager message:Committing JDBC transaction on␣
↪Connection [net.sf.log4jdbc.ConnectionSpy@4e53de7c]
date:2016-02-17 13:18:55    thread:tomcat-http--5   X-Track:
↪390066c43aa94b6588e5bac6a54812b2        level:DEBUG     logger:o.s.jdbc.
↪datasource.DataSourceTransactionManager message:Releasing JDBC Connection [net.
↪sf.log4jdbc.ConnectionSpy@4e53de7c] after transaction
date:2016-02-17 13:18:55    thread:tomcat-http--5   X-Track:
↪390066c43aa94b6588e5bac6a54812b2        level:TRACE     logger:o.t.gfw.web.
↪logging.TraceLoggingInterceptor      message:[END CONTROLLER  ] TodoController.
↪list(Model)-> view=todo/list, model={todoForm=todo.app.todo.TodoForm@2a075f1d,␣
↪todos=[], org.springframework.validation.BindingResult.todoForm=org.
↪springframework.validation.BeanPropertyBindingResult: 0 errors}
date:2016-02-17 13:18:55    thread:tomcat-http--5   X-Track:
↪390066c43aa94b6588e5bac6a54812b2        level:TRACE     logger:o.t.gfw.web.
↪logging.TraceLoggingInterceptor      message:[HANDLING TIME   ] TodoController.
↪list(Model)-> 756,709,153 ns
```

Creating infrastructure layer with Spring Data JPA

Here, How to create RepositoryImpl of infrastructure layer using Spring Data JPA is explained.

Modification in Entity

Configure the JPA annotation for mapping the Todo class with TODO table of Database.

```
package todo.domain.model;

import java.io.Serializable;
import java.util.Date;

import javax.persistence.Column;
```

```
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;
import javax.persistence.Temporal;
import javax.persistence.TemporalType;

// (1)
@Entity
@Table(name = "todo")
public class Todo implements Serializable {
    private static final long serialVersionUID = 1L;

    // (2)
    @Id
    @Column(name = "todo_id")
    private String todoId;

    @Column(name = "todo_title")
    private String todoTitle;

    @Column(name = "finished")
    private boolean finished;

    // (3)
    @Temporal(TemporalType.TIMESTAMP)
    @Column(name = "created_at")
    private Date createdAt;

    public String getTodoId() {
        return todoId;
    }

    public void setTodoId(String todoId) {
        this.todoId = todoId;
    }

    public String getTodoTitle() {
        return todoTitle;
    }

    public void setTodoTitle(String todoTitle) {
        this.todoTitle = todoTitle;
    }

    public boolean isFinished() {
        return finished;
    }

    public void setFinished(boolean finished) {
        this.finished = finished;
```

```
    }

    public Date getCreatedAt() {
        return createdAt;
    }

    public void setCreatedAt(Date createdAt) {
        this.createdAt = createdAt;
    }
}
```

Sr. No.	Description
(1)	Add `@Entity` showing that it is JPA entity and set the corresponding table name using `@Table`.
(2)	Add `@Id` to the field corresponding to primary key column.
(3)	It should be clearly specified that `java.util.Date` type corresponds to which of `java.sql.Date`, `java.sql.Time`, `java.sql.Timestamp`. Here `Timestamp` is specified to `createdAt` property.

Creat TodoRepository

Create `TodoRepository` using Repository feature of Spring Data JPA.

Right click on the Package Explorer, select -> [New] -> [Interface] -> [New Java Interface] dialog box appears,

Sr. No.	Item	Input value
1	Package	`todo.domain.repository.todo`
2	Name	`TodoRepository`
3	Extended interfaces	`org.springframework.data.jpa.repository.JpaReposit`

and click [Finish] after entering above information.

```
package todo.domain.repository.todo;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;

import todo.domain.model.Todo;
```

```
// (1)
public interface TodoRepository extends JpaRepository<Todo, String> {

    @Query("SELECT COUNT(t) FROM Todo t WHERE t.finished = :finished") // (2)
    long countByFinished(@Param("finished") boolean finished); // (3)

}
```

Sr. No.	Description
(1)	Specify generics parameter of `JpaRepository`. From left to right, Entity Classes (`Todo`), primary key class (`String`) is specified. Since basic CRUD operations (`findOne`, `findAll`, `save`, `delete` etc.) are defined in `JpaRepository` interface, only `countByFinished` can be defined in `TodoRepository`.
(2)	Specify `@Query` annotation to the JPQL executed at the time of calling `countByFinished` method
(3)	Set bind variable of JPQL specified in (2) by `@Param` annotation. Here, `@Param(“finished”)` is added before the method argument `finished` to embed the value with ”`:finished`” of JPQL.

Create TodoRepositoryImpl

If Spring Data JPA is used, RepositoryImpl is automatically generated from the Repository interface. Therefore, the creation of `TodoRepositoryImpl` is not required. Remove if it is created.

Since the creation of infrastructure layer using Spring Data JPA has been completed, create the application layer components and services.

Once creation of the Services and application layer are completed, execute the Todo application after starting the AP server, SQL and transaction log output is as following.

```
date:2016-02-17 13:32:44    thread:tomcat-http--5   X-Track:
↪7c34263e0a2143639f3ffd191b35c135        level:TRACE     logger:o.t.gfw.web.
↪logging.TraceLoggingInterceptor      message:[START CONTROLLER] TodoController.
↪list(Model)
date:2016-02-17 13:32:44    thread:tomcat-http--5   X-Track:
↪7c34263e0a2143639f3ffd191b35c135        level:DEBUG     logger:o.h.e.transaction.
↪spi.AbstractTransactionImpl    message:begin
date:2016-02-17 13:32:44    thread:tomcat-http--5   X-Track:
↪7c34263e0a2143639f3ffd191b35c135        level:DEBUG     logger:o.h.e.transaction.
↪internal.jdbc.JdbcTransaction  message:initial autocommit status: false
date:2016-02-17 13:32:45    thread:tomcat-http--5   X-Track:
↪7c34263e0a2143639f3ffd191b35c135        level:DEBUG     logger:jdbc.sqltiming
                            message: org.hibernate.engine.jdbc.internal.
↪ResultSetReturnImpl.extract(ResultSetReturnImpl.java:82)
5. /* select generatedAlias0 from Todo as generatedAlias0 */ select todo0_.todo_
↪id as todo_id1_0_, todo0_.created_at as created_2_0_, todo0_.finished as␣
↪finished3_0_, todo0_.todo_title as todo_tit4_0_ from todo todo0_ {executed in 1␣
↪msec}
date:2016-02-17 13:32:45    thread:tomcat-http--5   X-Track:
↪7c34263e0a2143639f3ffd191b35c135        level:INFO      logger:jdbc.
↪resultsettable                             message:|--------|-----------|-------
↪--|-----------|
date:2016-02-17 13:32:45    thread:tomcat-http--5   X-Track:
↪7c34263e0a2143639f3ffd191b35c135        level:INFO      logger:jdbc.
↪resultsettable                             message:|TODO_ID |CREATED_AT␣
↪|FINISHED |TODO_TITLE |
date:2016-02-17 13:32:45    thread:tomcat-http--5   X-Track:
↪7c34263e0a2143639f3ffd191b35c135        level:INFO      logger:jdbc.
↪resultsettable                             message:|--------|-----------|-------
↪--|-----------|
date:2016-02-17 13:32:45    thread:tomcat-http--5   X-Track:
↪7c34263e0a2143639f3ffd191b35c135        level:INFO      logger:jdbc.
↪resultsettable                             message:|--------|-----------|-------
↪--|-----------|
date:2016-02-17 13:32:45    thread:tomcat-http--5   X-Track:
↪7c34263e0a2143639f3ffd191b35c135        level:DEBUG     logger:o.h.e.transaction.
↪spi.AbstractTransactionImpl    message:committing
date:2016-02-17 13:32:45    thread:tomcat-http--5   X-Track:
↪7c34263e0a2143639f3ffd191b35c135        level:DEBUG     logger:o.h.e.transaction.
↪internal.jdbc.JdbcTransaction  message:committed JDBC Connection
date:2016-02-17 13:32:45    thread:tomcat-http--5   X-Track:
↪7c34263e0a2143639f3ffd191b35c135        level:TRACE     logger:o.t.gfw.web.
↪logging.TraceLoggingInterceptor      message:[END CONTROLLER  ] TodoController.
↪list(Model)-> view=todo/list, model={todoForm=todo.app.todo.TodoForm@5dac2c75,␣
↪todos=[], org.springframework.validation.BindingResult.todoForm=org.
↪springframework.validation.BeanPropertyBindingResult: 0 errors}
date:2016-02-17 13:32:45    thread:tomcat-http--5   X-Track:
↪7c34263e0a2143639f3ffd191b35c135        level:TRACE     logger:o.t.gfw.web.
↪logging.TraceLoggingInterceptor      message:[HANDLING TIME   ] TodoController.
↪list(Model)-> 320,129,237 ns
```

10.1.6 In the end...

In this tutorial, following contents have been learnt.

- How to develop basic applications by TERASOLUNA Server Framework for Java (5.x)
- How to build Maven as well as STS (Eclipse) project
- Way of development using application layering of TERASOLUNA Server Framework for Java (5.x).
- Implementation of domain layer with POJO(+ Spring)
- Implementation of application layer with the use of JSP tag libraries and POJO(+ Spring MVC)
- Development of Infrastructure layer with the use of MyBatis3
- Development of Infrastructure layer with the use of Spring Data JPA
- Development of Infrastructure layer without use of O/R Mapper

The following improvement can be done in the TODO management application. As a learning challenge of the application improvement refer the appropriate description of the guidelines.

- To externalize the property (Maximum number of uncompleted TODO) -> *Properties Management*
- To externalize the messages -> *Message Management*
- To add pagination function -> *Pagination*
- To add exception handling -> *Exception Handling*
- To add double submit protection (Support the transaction token check) -> *Double Submit Protection*
- To change how to get the system date time -> *System Date*

10.1.7 Appendix

Description of the configuration file

Description of the configuration files are done for giving an understanding of what type of settings are required to run an application. Settings that are not used in created Todo tutorial application are ignored here.

web.xml

In `web.xml`, the settings are done for deploying the Todo application as a Web application.

Following settings are done in created blank project `src/main/webapp/WEB-INF/web.xml`.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- (1) -->
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/
↪ns/javaee/web-app_3_0.xsd"
    version="3.0">

    <!-- (2) -->
    <listener>
        <listener-class>org.springframework.web.context.ContextLoaderListener</
↪listener-class>
    </listener>
    <context-param>
        <param-name>contextConfigLocation</param-name>
        <!-- Root ApplicationContext -->
        <param-value>
            classpath*:META-INF/spring/applicationContext.xml
            classpath*:META-INF/spring/spring-security.xml
        </param-value>
    </context-param>

    <listener>
        <listener-class>org.terasoluna.gfw.web.logging.
↪HttpSessionEventLoggingListener</listener-class>
    </listener>

    <!-- (3) -->
    <filter>
        <filter-name>MDCClearFilter</filter-name>
        <filter-class>org.terasoluna.gfw.web.logging.mdc.MDCClearFilter</filter-
↪class>
    </filter>
    <filter-mapping>
        <filter-name>MDCClearFilter</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>

    <filter>
        <filter-name>exceptionLoggingFilter</filter-name>
        <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-
↪class>
    </filter>
    <filter-mapping>
```

```
        <filter-name>exceptionLoggingFilter</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>

    <filter>
        <filter-name>XTrackMDCPutFilter</filter-name>
        <filter-class>org.terasoluna.gfw.web.logging.mdc.XTrackMDCPutFilter</
↪filter-class>
    </filter>
    <filter-mapping>
        <filter-name>XTrackMDCPutFilter</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>

    <filter>
        <filter-name>CharacterEncodingFilter</filter-name>
        <filter-class>org.springframework.web.filter.CharacterEncodingFilter</
↪filter-class>
        <init-param>
            <param-name>encoding</param-name>
            <param-value>UTF-8</param-value>
        </init-param>
        <init-param>
            <param-name>forceEncoding</param-name>
            <param-value>true</param-value>
        </init-param>
    </filter>
    <filter-mapping>
        <filter-name>CharacterEncodingFilter</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>

    <filter>
        <filter-name>springSecurityFilterChain</filter-name>
        <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-
↪class>
    </filter>
    <filter-mapping>
        <filter-name>springSecurityFilterChain</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>

    <!-- (4) -->
    <servlet>
        <servlet-name>appServlet</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-
↪class>
        <init-param>
            <param-name>contextConfigLocation</param-name>
            <!-- ApplicationContext for Spring MVC -->
```

```
            <param-value>classpath*:META-INF/spring/spring-mvc.xml</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>appServlet</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>

    <!-- (5) -->
    <jsp-config>
        <jsp-property-group>
            <url-pattern>*.jsp</url-pattern>
            <el-ignored>false</el-ignored>
            <page-encoding>UTF-8</page-encoding>
            <scripting-invalid>false</scripting-invalid>
            <include-prelude>/WEB-INF/views/common/include.jsp</include-prelude>
        </jsp-property-group>
    </jsp-config>

    <!-- (6) -->
    <error-page>
        <error-code>500</error-code>
        <location>/WEB-INF/views/common/error/systemError.jsp</location>
    </error-page>

    <error-page>
        <error-code>404</error-code>
        <location>/WEB-INF/views/common/error/resourceNotFoundError.jsp</location>
    </error-page>

    <error-page>
        <exception-type>java.lang.Exception</exception-type>
        <location>/WEB-INF/views/common/error/unhandledSystemError.html</location>
    </error-page>

    <!-- (7) -->
    <session-config>
        <!-- 30min -->
        <session-timeout>30</session-timeout>
        <cookie-config>
            <http-only>true</http-only>
            <!-- <secure>true</secure> -->
        </cookie-config>
        <tracking-mode>COOKIE</tracking-mode>
    </session-config>

</web-app>
```

Sr. No.	Description
(1)	Declaration to use Servlet3.0.
(2)	Definition of servlet context listener. In the blank project, • `ContextLoaderListener` is used to create an `ApplicationContext` for entire application • `HttpSessionEventLoggingListener` is used to log output the HTTPSession operations are configured.
(3)	Definition of servlet filter. In the blank project, • Servlet filter provided by the common library • `CharacterEncodingFilter` for specifing the character encoding provided by the Spring Framework • Servlet filter for authentication and authorization provided by the Spring Security are configured.
(4)	Definition of DispatcherServlet that is the entry point of Spring MVC. `ApplicationContext` used in the DispatcherServlet is the child of `ApplicationContext` created in step (2). It is also possible to use components loaded in (2) by setting `ApplicationContext` created at step (2) as a parent.
(5)	Common definition of JSP. In the blank project, • EL expression can be used in JSP • UTF-8 as the JSP page encoding • Scripting can be used in JSP • Included `/WEB-INF/views/common/include.jsp` at the beginning of each JSP are configured.

次のページに続く

Table. 10.3 – 前のページからの続き

Sr. No.	Description
(6)	Definition of error page. In the blank project, • Respond HTTP status code `404` or `500` to the servlet container • Exception notification to the servlet container related transition destination are defined.
(7)	Definition of session management. In the blank project, • 30 minutes as a session time-out are defined.

Common JSP

Tag library settings applied to all JSP are done in Include JSP.

Following settings are done in created blank project `src/main/webapp/WEB-INF/views/common/include.jsp`.

```
<!-- (1) -->
<%@ page session="false"%>
<!-- (2) -->
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt"%>
<!-- (3) -->
<%@ taglib uri="http://www.springframework.org/tags" prefix="spring"%>
<%@ taglib uri="http://www.springframework.org/tags/form" prefix="form"%>
<!-- (4) -->
<%@ taglib uri="http://www.springframework.org/security/tags" prefix="sec"%>
<!-- (5) -->
<%@ taglib uri="http://terasoluna.org/tags" prefix="t"%>
<%@ taglib uri="http://terasoluna.org/functions" prefix="f"%>
```

Sr. No.	Description
(1)	Definition to avoid session creation while JSP execution.
(2)	Definition standard tag library.
(3)	Definition tag library for Spring MVC.
(4)	Definition tag library for Spring Security.(However, it is not used in this tutorial)
(5)	Definition EL function and tag library provided in common library.

Bean definition file

Following Bean definition files and property files are generated in created blank project.

- `src/main/resources/META-INF/spring/applicationContext.xml`
- `src/main/resources/META-INF/spring/todo-domain.xml`
- `src/main/resources/META-INF/spring/todo-infra.xml`
- `src/main/resources/META-INF/spring/todo-infra.properties`
- `src/main/resources/META-INF/spring/todo-env.xml`
- `src/main/resources/META-INF/spring/spring-mvc.xml`
- `src/main/resources/META-INF/spring/spring-security.xml`

Note: The `todo-infra.properties` and `todo-env.xml` are not created while creating blank project that does not dependent on the O/R Mapper.

Note: In this guideline, it is recommended to split Bean definition file for each role (layer).

By this way, what is defined in which file can be easily understood and improves the ease of maintenance. It is not effective in case of small application like tutorial but more effective in case of large scale application.

---

applicationContext.xml

Perform entire Todo application related settings in the `applicationContext.xml`.

Following settings are done in created blank project
`src/main/resources/META-INF/spring/applicationContext.xml`.
In addition, a description of the components that are not used in the tutorial are omitted.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
```

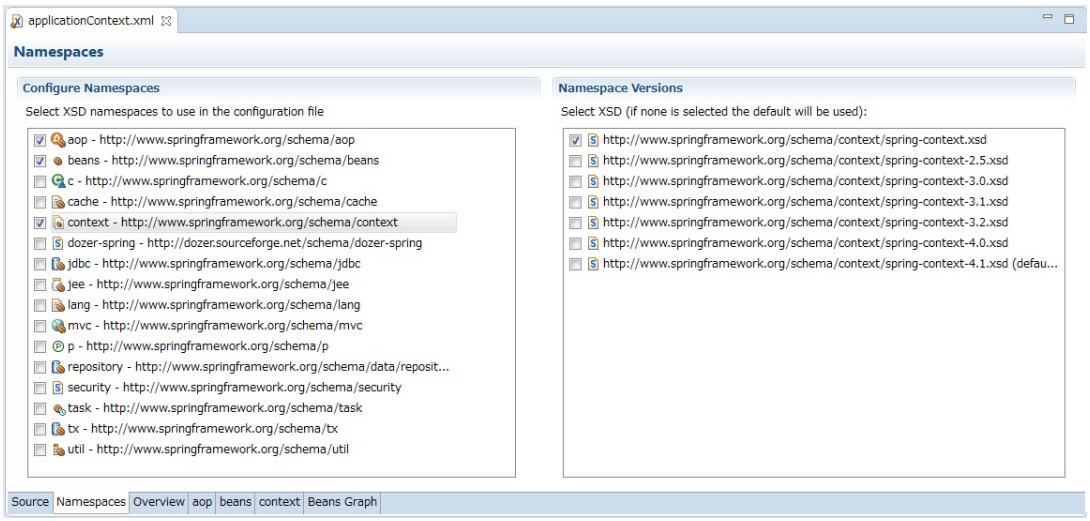
```
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans http://www.springframework.org/
↪schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/context http://www.springframework.
↪org/schema/context/spring-context.xsd
    ">

    <!-- (1) -->
    <import resource="classpath:/META-INF/spring/todo-domain.xml" />

    <bean id="passwordEncoder" class="org.springframework.security.crypto.bcrypt.
↪BCryptPasswordEncoder" />

    <!-- (2) -->
    <context:property-placeholder
        location="classpath*:/META-INF/spring/*.properties" />

    <!-- (3) -->
    <bean id="beanMapper" class="org.dozer.spring.DozerBeanMapperFactoryBean">
        <property name="mappingFiles"
            value="classpath*:/META-INF/dozer/**/*-mapping.xml" />
    </bean>

    <!-- Message -->
    <bean id="messageSource"
        class="org.springframework.context.support.ResourceBundleMessageSource">
        <property name="basenames">
            <list>
                <value>i18n/application-messages</value>
            </list>
        </property>
    </bean>

    <!-- Exception Code Resolver. -->
    <bean id="exceptionCodeResolver"
        class="org.terasoluna.gfw.common.exception.
↪SimpleMappingExceptionCodeResolver">
        <!-- Setting and Customization by project. -->
        <property name="exceptionMappings">
            <map>
                <entry key="ResourceNotFoundException" value="e.xx.fw.5001" />
                <entry key="InvalidTransactionTokenException" value="e.xx.fw.7001"␣
↪/>
                <entry key="BusinessException" value="e.xx.fw.8001" />
                <entry key=".DataAccessException" value="e.xx.fw.9002" />
            </map>
        </property>
        <property name="defaultExceptionCode" value="e.xx.fw.9001" />
```

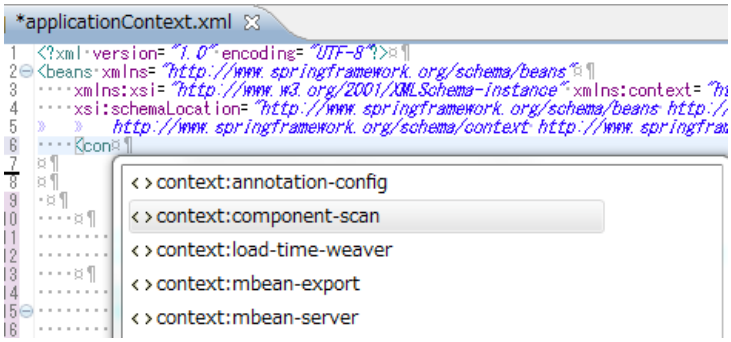
```
    </bean>

    <!-- Exception Logger. -->
    <bean id="exceptionLogger"
        class="org.terasoluna.gfw.common.exception.ExceptionLogger">
        <property name="exceptionCodeResolver" ref="exceptionCodeResolver" />
    </bean>

    <!-- Filter. -->
    <bean id="exceptionLoggingFilter"
        class="org.terasoluna.gfw.web.exception.ExceptionLoggingFilter" >
        <property name="exceptionLogger" ref="exceptionLogger" />
    </bean>

</beans>
```

Sr. No.	Description
(1)	Import Bean definition file related to domain layer.
(2)	Loading configuration of the property file. Load all property files stored under the `src/main/resources/META-INF/spring`. Using this setting, it is possible to insert property file value in `${propertyName}` format in Bean definition file and to inject using `@Value("${propertyName}")` in Java class.
(3)	Define Dozer Mapper of Bean conversion library. For the mapping file, refer to Dozer manual.

Tip: By inserting following type of check-mark in [Configure Namespace] tab of the editor, selected XML schema gets enabled and possible to supplement the input using Ctrl+Space at the time of XML editing.

It is recommended to select xsd file without version in [Namespace Versions]. By selecting the xsd file without version, always the latest xsd is used included in the jar therefore no need to concern about the Spring version up.





todo-domain.xml

Perform the domain layer related settings of the Todo application in `todo-domain.xml`.

Following settings are done in created blank project
`src/main/resources/META-INF/spring/todo-domain.xml`.

In addition, a description of the components that are not used in the tutorial are omitted.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:aop="http://www.springframework.org/schema/aop"
    xsi:schemaLocation="
        http://www.springframework.org/schema/aop http://www.springframework.org/
↪schema/aop/spring-aop.xsd
        http://www.springframework.org/schema/beans http://www.springframework.org/
↪schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/context http://www.springframework.
↪org/schema/context/spring-context.xsd
```

```
    ">

    <!-- (1) -->
    <import resource="classpath:META-INF/spring/todo-infra.xml" />
    <import resource="classpath*:META-INF/spring/**/*-codelist.xml" />

    <!-- (2) -->
    <context:component-scan base-package="todo.domain" />

    <!-- AOP. -->
    <bean id="resultMessagesLoggingInterceptor"
        class="org.terasoluna.gfw.common.exception.ResultMessagesLoggingInterceptor
↪">
        <property name="exceptionLogger" ref="exceptionLogger" />
    </bean>
    <aop:config>
        <aop:advisor advice-ref="resultMessagesLoggingInterceptor"
            pointcut="@within(org.springframework.stereotype.Service)" />
    </aop:config>

</beans>
```

Sr. No.	Description
(1)	Import Bean definition file related to infrastructure layer.
(2)	Components under todo.domain package are target of component scan. Thus, by attaching annotations like `@Repository` , `@Service` etc to the class under todo.domain package, it get registered as Bean of Spring Framework. It is possible to use registered classes (Bean) by doing DI in Controller and Service class.

Note: The `<tx:annotation-driven>` tag is set in order to enable the transaction management by `@Transactional` annotation while creating O/R Mapper dependent blank project.

```
<tx:annotation-driven />
```

todo-infra.xml

Perform infrastructure layer related settings of the TODO application in `todo-infra.xml`.

Following settings are done in created blank project `src/main/resources/META-INF/spring/todo-infra.xml`.

Since configuration of `todo-infra.xml` is significantly different by the infrastructure layer, description of each blank project done. You may skip the explanation other than the created blank project.

todo-infra.xml if creating blank project that does not dependent on the O/R Mapper

Empty definition file is created as follows while creating blank project that does not dependent on the O/R Mapper.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans http://www.springframework.org/
↪schema/beans/spring-beans.xsd
    ">

</beans>
```

todo-infra.xml of blank project created for MyBatis3

The following settings are done in created MyBatis3 project.

```
 <?xml version="1.0" encoding="UTF-8"?>
 <beans xmlns="http://www.springframework.org/schema/beans"
     xmlns:mybatis="http://mybatis.org/schema/mybatis-spring"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="
         http://www.springframework.org/schema/beans http://www.springframework.
↪org/schema/beans/spring-beans.xsd
         http://mybatis.org/schema/mybatis-spring http://mybatis.org/schema/
↪mybatis-spring.xsd
     ">

      <!-- (1) -->
     <import resource="classpath:/META-INF/spring/todo-env.xml" />

      <!-- (2) -->
     <!-- define the SqlSessionFactory -->
     <bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
          <!-- (3) -->
         <property name="dataSource" ref="dataSource" />
          <!-- (4) -->
         <property name="configLocation" value="classpath:/META-INF/mybatis/
↪mybatis-config.xml" />
```

```
    </bean>

     <!-- (5) -->
    <!-- scan for Mappers -->
    <mybatis:scan base-package="todo.domain.repository" />

</beans>
```

Sr. No.	Description
(1)	Import the Bean definition file that defines the environment dependent components(Such as DataSource and transaction manager).
(2)	The bean definition of `SqlSessionFactoryBean` as a component for generating a `SqlSessionFactory`.
(3)	Specify the bean configured DataSource in the `dataSource` property. The connection is retrieved from the DataSource specified here by executing the SQL in MyBatis3 process.
(4)	Specify the path of MyBatis configuration file in the `configLocation` property. The specified file loaded while generating a `SqlSessionFactory`
(5)	Define `<mybatis:scan>` for scanning the Mapper interface, Specify the base package in the `base-package` attribute that contains the Mapper interface. Scanned the Mapper interface that stored under the specified package and automatically generated the thread-safe Mapper object (Proxy object of Mapper interface).

Note: `mybatis-config.xml` is a configuration file that performs operational setting of MyBatis3.

The following settings are done by default in the blank project.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE configuration
    PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
    "http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>

    <!-- See http://mybatis.github.io/mybatis-3/configuration.html#settings --
↪>
    <settings>
        <setting name="mapUnderscoreToCamelCase" value="true" />
        <setting name="lazyLoadingEnabled" value="true" />
        <setting name="defaultFetchSize" value="100" />
<!--
        <setting name="defaultExecutorType" value="REUSE" />
        <setting name="jdbcTypeForNull" value="NULL" />
        <setting name="localCacheScope" value="STATEMENT" />
-->
    </settings>

    <typeAliases>
        <package name="todo.domain.model" />
        <package name="todo.domain.repository" />
<!--
        <package name="todo.infra.mybatis.typehandler" />
-->
    </typeAliases>

    <typeHandlers>
<!--
        <package name="todo.infra.mybatis.typehandler" />
-->
    </typeHandlers>

</configuration>
```

todo-infra.xml of blank project created for JPA

The following settings are done in created JPA blank project.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:jpa="http://www.springframework.org/schema/data/jpa"
    xmlns:util="http://www.springframework.org/schema/util"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans http://www.springframework.org/
↪schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/util http://www.springframework.org/
↪schema/util/spring-util.xsd
```

```
        http://www.springframework.org/schema/data/jpa http://www.springframework.
↪org/schema/data/jpa/spring-jpa.xsd
    ">

    <!-- (1) -->
    <import resource="classpath:/META-INF/spring/todo-env.xml" />

    <!-- (2) -->
    <jpa:repositories base-package="todo.domain.repository"></jpa:repositories>

    <!-- (3) -->
    <bean id="jpaVendorAdapter"
        class="org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter">
        <property name="showSql" value="false" />
        <property name="database" value="${database}" />
    </bean>

    <!-- (4) -->
    <bean id="entityManagerFactory"
        class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">
        <!-- (5) -->
        <property name="packagesToScan" value="todo.domain.model" />
        <property name="dataSource" ref="dataSource" />
        <property name="jpaVendorAdapter" ref="jpaVendorAdapter" />
        <!-- (6) -->
        <property name="jpaPropertyMap">
            <util:map>
                <entry key="hibernate.hbm2ddl.auto" value="" />
                <entry key="hibernate.ejb.naming_strategy"
                    value="org.hibernate.cfg.ImprovedNamingStrategy" />
                <entry key="hibernate.connection.charSet" value="UTF-8" />
                <entry key="hibernate.show_sql" value="false" />
                <entry key="hibernate.format_sql" value="false" />
                <entry key="hibernate.use_sql_comments" value="true" />
                <entry key="hibernate.jdbc.batch_size" value="30" />
                <entry key="hibernate.jdbc.fetch_size" value="100" />
            </util:map>
        </property>
    </bean>

</beans>
```

Sr. No.	Description
(1)	Import the Bean definition file that defines the environment dependent components(Such as DataSource and transaction manager).
(2)	Automatically generate the implementation class from Repository interface using the Spring Data JPA. Specify the package in `base-package` attribute of `<jpa:repository>` tag that contains the Repository.
(3)	Configure JPA implementation vendor. In order to use Hibernate, `HibernateJpaVendorAdapter` is defined as a JPA implementation.
(4)	Define the `EntityManager`.
(5)	Specify the package name that treated as a JPA entity classes.
(6)	Perform the Hibernate related detail settings.

todo-infra.properties

Perform environment dependent infrastructure layer configuration of Todo application in the `todo-infra.properties`.

The `todo-infra.properties` is not created while creating blank project that does not dependent on the O/R Mapper.

Following settings are done in created blank project `src/main/resources/META-INF/spring/todo-infra.properti`

```
# (1)
database=H2
database.url=jdbc:h2:mem:todo;DB_CLOSE_DELAY=-1
database.username=sa
database.password=
database.driverClassName=org.h2.Driver
# (2)
# connection pool
cp.maxActive=96
cp.maxIdle=16
cp.minIdle=0
cp.maxWait=60000
```

Sr. No.	Description
(1)	Database related settings. In this tutorial, H2 Database is used to save the setup time of the database.
(2)	Connection pool related settings.

Note: The configuration value of this is referred from `todo-env.xml`.

todo-env.xml

In the `todo-env.xml` file, define such components which are differing depending upon the environment.

Following settings are done in created blank project `src/main/resources/META-INF/spring/todo-env.xml`.

Here, the file stored in the blank project for MyBatis3 is described as an example. Furthermore, the `todo-env.xml` is not created while creating blank project that does not access the database.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:jdbc="http://www.springframework.org/schema/jdbc"
    xsi:schemaLocation="
        http://www.springframework.org/schema/jdbc http://www.springframework.org/
↪schema/jdbc/spring-jdbc.xsd
```

```
        http://www.springframework.org/schema/beans http://www.springframework.org/
↪schema/beans/spring-beans.xsd
    ">

    <bean id="dateFactory" class="org.terasoluna.gfw.common.date.jodatime.
↪DefaultJodaTimeDateFactory" />

    <!-- (1) -->
    <bean id="realDataSource" class="org.apache.commons.dbcp2.BasicDataSource"
        destroy-method="close">
        <property name="driverClassName" value="${database.driverClassName}" />
        <property name="url" value="${database.url}" />
        <property name="username" value="${database.username}" />
        <property name="password" value="${database.password}" />
        <property name="defaultAutoCommit" value="false" />
        <property name="maxTotal" value="${cp.maxActive}" />
        <property name="maxIdle" value="${cp.maxIdle}" />
        <property name="minIdle" value="${cp.minIdle}" />
        <property name="maxWaitMillis" value="${cp.maxWait}" />
    </bean>


    <!-- (2) -->
    <bean id="dataSource" class="net.sf.log4jdbc.Log4jdbcProxyDataSource">
        <constructor-arg index="0" ref="realDataSource" />
    </bean>

    <jdbc:initialize-database data-source="dataSource"
        ignore-failures="ALL">
        <jdbc:script location="classpath:/database/${database}-schema.sql"␣
↪encoding="UTF-8" />
        <jdbc:script location="classpath:/database/${database}-dataload.sql"␣
↪encoding="UTF-8" />
    </jdbc:initialize-database>

    <!--  REMOVE THIS LINE IF YOU USE JPA
    <bean id="transactionManager"
        class="org.springframework.orm.jpa.JpaTransactionManager">
        <property name="entityManagerFactory" ref="entityManagerFactory" />
    </bean>
          REMOVE THIS LINE IF YOU USE JPA  -->
    <!-- (3) -->
    <bean id="transactionManager"
        class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
        <property name="dataSource" ref="dataSource" />
        <property name="rollbackOnCommitFailure" value="true" />
    </bean>
</beans>
```

Sr. No.	Description
(1)	Setting of the actual DataSource.
(2)	Setting of the DataSource. Specify the DataSource that having JDBC-related logging feature. By use of `net.sf.log4jdbc.Log4jdbcProxyDataSource` , you can output JDBC related log such as SQL log which could be very useful for debugging.
(3)	Setting of the Transaction Manager. Specify the `transactionManager` in id attribute. In case of specifing different name, it is necessary to specify the transaction manager name in `<tx:annotation-driven>` tag also. In the blank project, transaction controlling class (`org.springframework.jdbc.datasource.DataSourceTransactionManager`) is configured using JDBC API.

Note: If blank project created for JPA, Transaction Manager, transaction controlling class (`org.springframework.orm.jpa.JpaTransactionManager`) is configured using JPA API.

```
<bean id="transactionManager"
    class="org.springframework.orm.jpa.JpaTransactionManager">
    <property name="entityManagerFactory" ref="entityManagerFactory" />
</bean>
```

spring-mvc.xml

The Spring MVC related definitions are done in `spring-mvc.xml`.

Following settings are done in created blank project

src/main/resources/META-INF/spring/spring-mvc.xml.

In addition, a description of the components that are not used in the tutorial are omitted.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:mvc="http://www.springframework.org/schema/mvc"
    xmlns:util="http://www.springframework.org/schema/util"
    xmlns:aop="http://www.springframework.org/schema/aop"
    xsi:schemaLocation="http://www.springframework.org/schema/mvc http://www.
↪springframework.org/schema/mvc/spring-mvc.xsd
        http://www.springframework.org/schema/beans http://www.springframework.org/
↪schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/util http://www.springframework.org/
↪schema/util/spring-util.xsd
        http://www.springframework.org/schema/context http://www.springframework.
↪org/schema/context/spring-context.xsd
        http://www.springframework.org/schema/aop http://www.springframework.org/
↪schema/aop/spring-aop.xsd
    ">

    <!-- (1) -->
    <context:property-placeholder
        location="classpath*:/META-INF/spring/*.properties" />

    <!-- (2) -->
    <mvc:annotation-driven>
        <mvc:argument-resolvers>
            <bean
                class="org.springframework.data.web.
↪PageableHandlerMethodArgumentResolver" />
            <bean
                class="org.springframework.security.web.method.annotation.
↪AuthenticationPrincipalArgumentResolver" />
        </mvc:argument-resolvers>
        <!-- workaround to CVE-2016-5007. -->
        <mvc:path-matching path-matcher="pathMatcher" />
    </mvc:annotation-driven>

    <mvc:default-servlet-handler />

    <!-- (3) -->
    <context:component-scan base-package="todo.app" />

    <!-- (4) -->
    <mvc:resources mapping="/resources/**"
        location="/resources/,classpath:META-INF/resources/"
        cache-period="#{60 * 60}" />
```

```
    <mvc:interceptors>
        <!-- (5) -->
        <mvc:interceptor>
            <mvc:mapping path="/**" />
            <mvc:exclude-mapping path="/resources/**" />
            <mvc:exclude-mapping path="/**/*.html" />
            <bean
                class="org.terasoluna.gfw.web.logging.TraceLoggingInterceptor" />
        </mvc:interceptor>
        <mvc:interceptor>
            <mvc:mapping path="/**" />
            <mvc:exclude-mapping path="/resources/**" />
            <mvc:exclude-mapping path="/**/*.html" />
            <bean
                class="org.terasoluna.gfw.web.token.transaction.
↪TransactionTokenInterceptor" />
        </mvc:interceptor>
        <mvc:interceptor>
            <mvc:mapping path="/**" />
            <mvc:exclude-mapping path="/resources/**" />
            <mvc:exclude-mapping path="/**/*.html" />
            <bean class="org.terasoluna.gfw.web.codelist.CodeListInterceptor">
                <property name="codeListIdPattern" value="CL_.+" />
            </bean>
        </mvc:interceptor>
        <!--  REMOVE THIS LINE IF YOU USE JPA
        <mvc:interceptor>
            <mvc:mapping path="/**" />
            <mvc:exclude-mapping path="/resources/**" />
            <mvc:exclude-mapping path="/**/*.html" />
            <bean
                class="org.springframework.orm.jpa.support.
↪OpenEntityManagerInViewInterceptor" />
        </mvc:interceptor>
            REMOVE THIS LINE IF YOU USE JPA  -->
    </mvc:interceptors>

    <!-- (6) -->
    <!-- Settings View Resolver. -->
    <mvc:view-resolvers>
        <mvc:jsp prefix="/WEB-INF/views/" />
    </mvc:view-resolvers>

    <bean id="requestDataValueProcessor"
        class="org.terasoluna.gfw.web.mvc.support.
↪CompositeRequestDataValueProcessor">
        <constructor-arg>
            <util:list>
                <bean
```

```
                    class="org.springframework.security.web.servlet.support.csrf.
↪CsrfRequestDataValueProcessor" />
                <bean
                    class="org.terasoluna.gfw.web.token.transaction.
↪TransactionTokenRequestDataValueProcessor" />
            </util:list>
        </constructor-arg>
    </bean>

    <!-- Setting Exception Handling. -->
    <!-- Exception Resolver. -->
    <bean id="systemExceptionResolver"
        class="org.terasoluna.gfw.web.exception.SystemExceptionResolver">
        <property name="exceptionCodeResolver" ref="exceptionCodeResolver" />
        <!-- Setting and Customization by project. -->
        <property name="order" value="3" />
        <property name="exceptionMappings">
            <map>
                <entry key="ResourceNotFoundException" value="common/error/
↪resourceNotFoundError" />
                <entry key="BusinessException" value="common/error/businessError" /
↪>
                <entry key="InvalidTransactionTokenException" value="common/error/
↪transactionTokenError" />
                <entry key=".DataAccessException" value="common/error/
↪dataAccessError" />
            </map>
        </property>
        <property name="statusCodes">
            <map>
                <entry key="common/error/resourceNotFoundError" value="404" />
                <entry key="common/error/businessError" value="409" />
                <entry key="common/error/transactionTokenError" value="409" />
                <entry key="common/error/dataAccessError" value="500" />
            </map>
        </property>
        <property name="defaultErrorView" value="common/error/systemError" />
        <property name="defaultStatusCode" value="500" />
    </bean>
    <!-- Setting AOP. -->
    <bean id="handlerExceptionResolverLoggingInterceptor"
        class="org.terasoluna.gfw.web.exception.
↪HandlerExceptionResolverLoggingInterceptor">
        <property name="exceptionLogger" ref="exceptionLogger" />
    </bean>
    <aop:config>
        <aop:advisor advice-ref="handlerExceptionResolverLoggingInterceptor"
            pointcut="execution(* org.springframework.web.servlet.
↪HandlerExceptionResolver.resolveException(..))" />
    </aop:config>
```

```
    <!-- Setting PathMatcher. -->
    <bean id="pathMatcher" class="org.springframework.util.AntPathMatcher">
        <property name="trimTokens" value="false" />
    </bean>

</beans>
```

Sr. No.	Description
(1)	Configuration of reading property file done Load all property files stored under the src/main/resources/META-INF/spring. Using this setting, it is possible to insert property file value in `${propertyName}` format in Bean definition file and to inject using `@Value("${propertyName}")` in Java class.
(2)	Carry out annotation based default settings of Spring MVC.
(3)	Components under todo.app package that holds classes of application layer are made target of component-scan.
(4)	Carry out the settings for accessing the static resource (css, images, js etc.). Set URL path to `mapping` attribute and physical path to `location` attribute. In case of this setting, when there is a request for `<contextPath>/rerources/app/css/styles.css`, the `WEB-INF/resources/app/css/styles.css` is searched. If not found, `META-INF/resources/app/css/styles.css` is searched in classpath (`src/main/resources` and jar). If `styles.css` is not stored anywhere, 404 error is returned. Here, cache period (3600 seconds = 60 minutes) of static resources is set in `cache-period` attribute. `cache-period="3600"` is also correct, however, in order to demonstrate that it is 60 minutes, it is better to write as `cache-period="#{60 * 60}"` which uses SpEL.

次のページに続く

Table. 10.4 – 前のページからの続き

Sr. No.	Description
(5)	Set interceptor that outputs trace log of controller processing. Set so that it excludes the path under `/resources` from mapping.

(6)	Carry out the settings of `ViewResolver`.
	Using these settings, for example, when view name `"hello"` is returned from controller, `/WEB-INF/views/hello.jsp` is executed.
	Tip: `<mvc:view-resolvers>` element is a XML element that added from Spring Framework 4.1 By using `<mvc:view-resolvers>` element, it is possible to define `ViewResolver` simply. The definition example of using the conventional `<bean>` element is shown below.
	`<bean id="viewResolver" class="org.springframework.web.servlet.view.InternalResourceViewReso`
	`<property name="prefix" value="/WEB-INF/views/" />`
	`<property name="suffix" value=".jsp" />`
	`</bean>`

Note: In blank project created for JPA, as a `<mvc:interceptors>` definition, `OpenEntityManagerInViewInterceptor` definition is in enable state.

```
<mvc:interceptor>
    <mvc:mapping path="/**" />
    <mvc:exclude-mapping path="/resources/**" />
    <mvc:exclude-mapping path="/**/*.html" />
    <bean
        class="org.springframework.orm.jpa.support.
↪OpenEntityManagerInViewInterceptor" />
</mvc:interceptor>
```

`OpenEntityManagerInViewInterceptor` is a Interceptor that performs start and end of the life cycle of the `EntityManager`. By adding this setting, the application layer (Controller or View class) Lazy Load is supported.

spring-security.xml

The Spring Security related definitions are done in `spring-security.xml`.

Following settings are done in created blank project `src/main/resources/META-INF/spring/spring-security.xml`.
In addition, a description of the Spring Security configuration file is omitted in the tutorial. For the Spring Security configuration file, Refer [*Spring Security Tutorial*].

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:sec="http://www.springframework.org/schema/security"
    xsi:schemaLocation="
        http://www.springframework.org/schema/security http://www.springframework.
↪org/schema/security/spring-security.xsd
        http://www.springframework.org/schema/beans http://www.springframework.org/
↪schema/beans/spring-beans.xsd
    ">

    <sec:http pattern="/resources/**" security="none"/>
```

```
    <sec:http>
        <sec:form-login/>
        <sec:logout/>
        <sec:access-denied-handler ref="accessDeniedHandler"/>
        <sec:custom-filter ref="userIdMDCPutFilter" after="ANONYMOUS_FILTER"/>
        <sec:session-management />
    </sec:http>

    <sec:authentication-manager />

    <!-- CSRF Protection -->
    <bean id="accessDeniedHandler"
        class="org.springframework.security.web.access.
↪DelegatingAccessDeniedHandler">
        <constructor-arg index="0">
            <map>
                <entry
                    key="org.springframework.security.web.csrf.
↪InvalidCsrfTokenException">
                    <bean
                        class="org.springframework.security.web.access.
↪AccessDeniedHandlerImpl">
                        <property name="errorPage"
                            value="/WEB-INF/views/common/error/
↪invalidCsrfTokenError.jsp" />
                    </bean>
                </entry>
                <entry
                    key="org.springframework.security.web.csrf.
↪MissingCsrfTokenException">
                    <bean
                        class="org.springframework.security.web.access.
↪AccessDeniedHandlerImpl">
                        <property name="errorPage"
                            value="/WEB-INF/views/common/error/
↪missingCsrfTokenError.jsp" />
                    </bean>
                </entry>
            </map>
        </constructor-arg>
        <constructor-arg index="1">
            <bean
                class="org.springframework.security.web.access.
↪AccessDeniedHandlerImpl">
                <property name="errorPage"
                    value="/WEB-INF/views/common/error/accessDeniedError.jsp" />
            </bean>
        </constructor-arg>
    </bean>
```

```
    <!-- Put UserID into MDC -->
    <bean id="userIdMDCPutFilter" class="org.terasoluna.gfw.security.web.logging.
↪UserIdMDCPutFilter">
    </bean>

</beans>
```

logback.xml

The log output related definitions are done in `logback.xml`.

Following settings are done in created blank project `src/main/resources/logback.xml`.

In addition, a description of the log settings that are not used in the tutorial are omitted.

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>

    <!-- (1) -->
    <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
        <encoder>
            <pattern><![CDATA[date:%d{yyyy-MM-dd HH:mm:ss}\tthread:%thread\tX-
↪Track:%X{X-Track}\tlevel:%-5level\tlogger:%-48logger{48}\tmessage:%msg%n]]></
↪pattern>
        </encoder>
    </appender>

    <appender name="APPLICATION_LOG_FILE" class="ch.qos.logback.core.rolling.
↪RollingFileAppender">
        <file>${app.log.dir:-log}/todo-application.log</file>
        <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
            <fileNamePattern>${app.log.dir:-log}/todo-application-%d{yyyyMMdd}.log
↪</fileNamePattern>
            <maxHistory>7</maxHistory>
        </rollingPolicy>
        <encoder>
            <charset>UTF-8</charset>
            <pattern><![CDATA[date:%d{yyyy-MM-dd HH:mm:ss}\tthread:%thread\tX-
↪Track:%X{X-Track}\tlevel:%-5level\tlogger:%-48logger{48}\tmessage:%msg%n]]></
↪pattern>
        </encoder>
```

```
    </appender>

    <appender name="MONITORING_LOG_FILE" class="ch.qos.logback.core.rolling.
↪RollingFileAppender">
        <file>${app.log.dir:-log}/todo-monitoring.log</file>
        <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
            <fileNamePattern>${app.log.dir:-log}/todo-monitoring-%d{yyyyMMdd}.log</
↪fileNamePattern>
            <maxHistory>7</maxHistory>
        </rollingPolicy>
        <encoder>
            <charset>UTF-8</charset>
            <pattern><![CDATA[date:%d{yyyy-MM-dd HH:mm:ss}\tX-Track:%X{X-Track}
↪\tlevel:%-5level\tmessage:%msg%n]]></pattern>
        </encoder>
    </appender>

    <!-- Application Loggers -->
    <!-- (2) -->
    <logger name="todo">
        <level value="debug" />
    </logger>

    <!-- TERASOLUNA -->
    <logger name="org.terasoluna.gfw">
        <level value="info" />
    </logger>
    <!-- (3) -->
    <logger name="org.terasoluna.gfw.web.logging.TraceLoggingInterceptor">
        <level value="trace" />
    </logger>
    <logger name="org.terasoluna.gfw.common.exception.ExceptionLogger">
        <level value="info" />
    </logger>
    <logger name="org.terasoluna.gfw.common.exception.ExceptionLogger.Monitoring"␣
↪additivity="false">
        <level value="error" />
        <appender-ref ref="MONITORING_LOG_FILE" />
    </logger>

    <!-- 3rdparty Loggers -->
    <logger name="org.springframework">
        <level value="warn" />
    </logger>

    <logger name="org.springframework.web.servlet">
        <level value="info" />
    </logger>

    <!--  REMOVE THIS LINE IF YOU USE JPA
```

```
    <logger name="org.hibernate.engine.transaction">
        <level value="debug" />
    </logger>
          REMOVE THIS LINE IF YOU USE JPA  -->
    <!--  REMOVE THIS LINE IF YOU USE MyBatis3
    <logger name="org.springframework.jdbc.datasource.DataSourceTransactionManager
↪">
        <level value="debug" />
    </logger>
          REMOVE THIS LINE IF YOU USE MyBatis3  -->

    <logger name="jdbc.sqltiming">
        <level value="debug" />
    </logger>

    <!-- only for development -->
    <logger name="jdbc.resultsettable">
        <level value="debug" />
    </logger>

    <root level="warn">
        <appender-ref ref="STDOUT" />
        <appender-ref ref="APPLICATION_LOG_FILE" />
    </root>

</configuration>
```

Sr. No.	Description
(1)	Set appender that outputs the log in standard output.
(2)	Set so that log of debug level and above is output under todo package.
(3)	Set log level to ‘trace’ for TraceLoggingInterceptor which is defined in spring-mvc.xml.

Note: If you create a blank project that uses the O/R Mapper, the logger to output transaction control-related log has been in enable state.

- Blank project for the JPA

```
<logger name="org.hibernate.engine.transaction">
    <level value="debug" />
</logger>
```

- Blank project for the MyBatis3

```
<logger name="org.springframework.jdbc.datasource.DataSourceTransactionManager
↪">
    <level value="debug" />
</logger>
```

10.2 Tutorial (Todo Application REST)

10.2.1 Introduction

Points to study in this tutorial

- Basic RESTful web service development using TERASOLUNA Server Framework for Java (5.x)

Target readers

- Those who have Implemented the *Tutorial (Todo Application)*.

Verification environment

In this tutorial, operations are verified on following environment.

In order to use the advanced features of Google Chrome, Google Chrome used as a Web Browser for REST Client.

Type	Product
REST Client	DHC REST Client 1.2.3
Product other than the above	Similar to *Tutorial (Todo Application)*

10.2.2 Environment creation

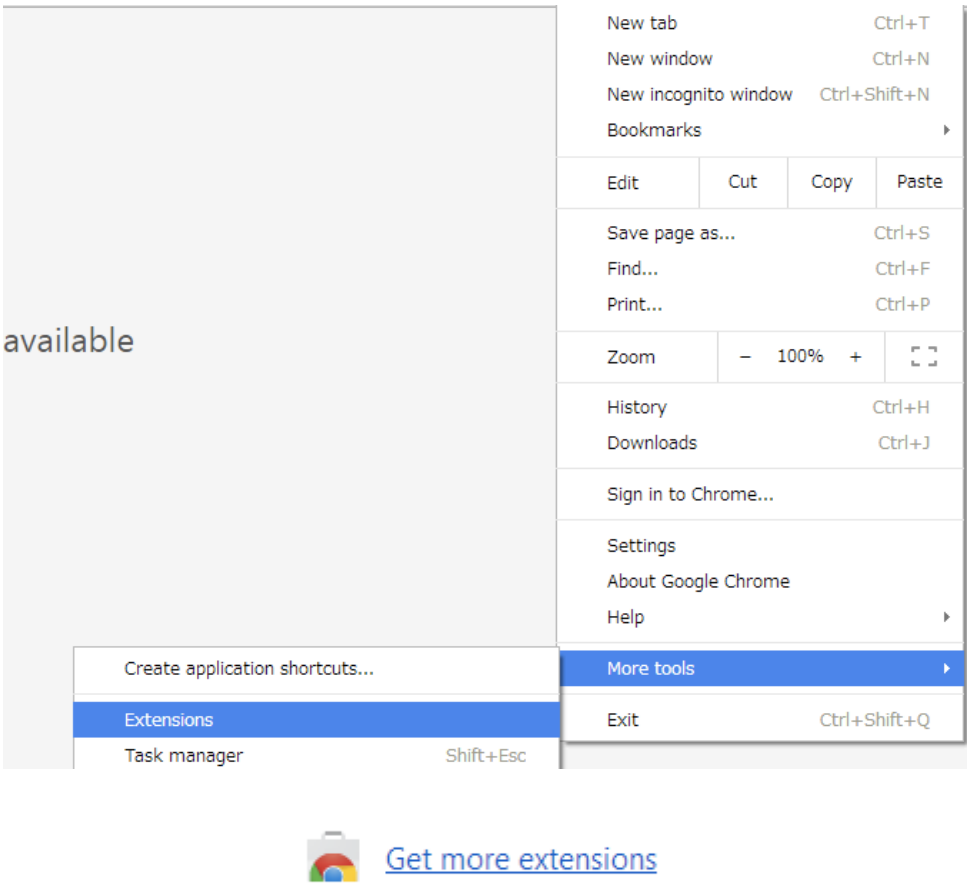
Assumes that Java, STS, Maven, Google Chrome are already installed since the *Tutorial (Todo Application)* is implemented.

Install DHC

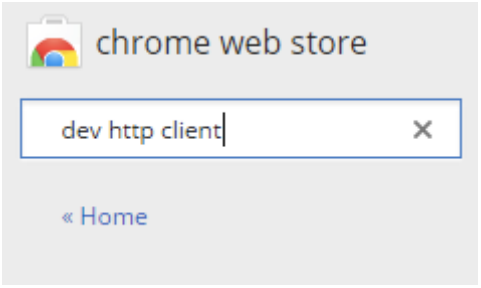
As a REST client, install [DHC] advanced features of Google Chrome.

Select [Tools] -> [Extensions] of Chrome browser.

Click [Get more extensions] link.



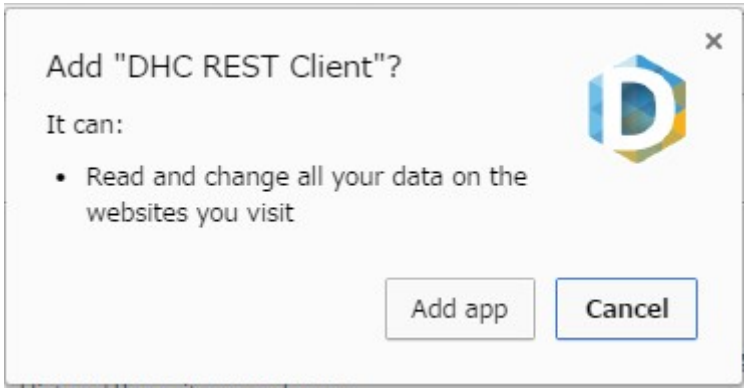
Search by entering [dev http client] in search form.



Click the [+ ADD TO CHROME] button of DHC REST Client.



Click [Add app] button.



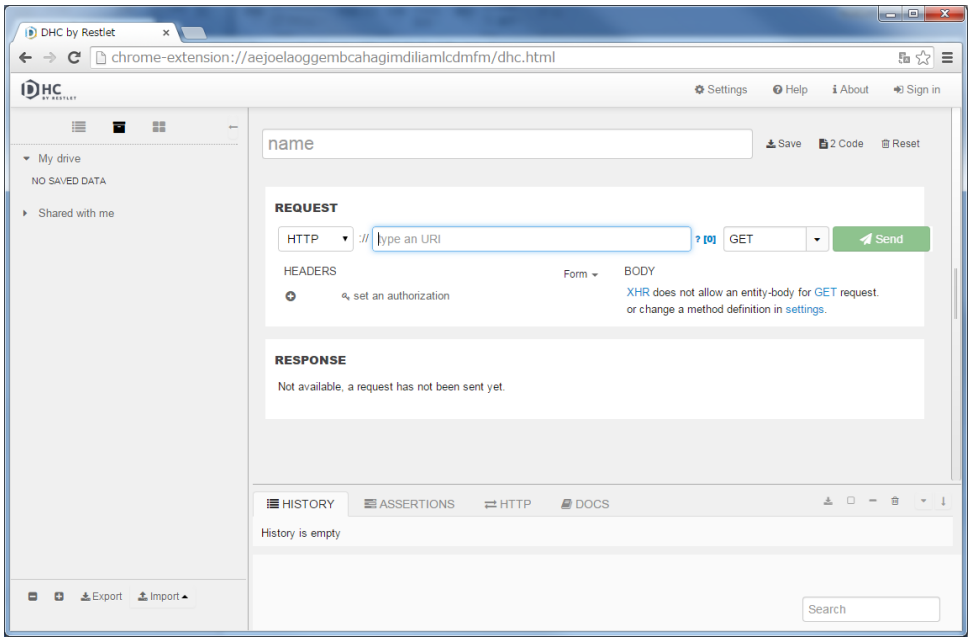
When you open the application list (Open by specifying [chrome://apps/] in your browser address bar) of Chrome, DHC has been added.



Click the DHC.

If the following screen appears, the installation is completed.

This screen can also be opened by entering the [chrome-extension://aejoelaoggembcahagimdiliamlcdmfm/dhc.html] in the address bar of the browser.



Project creation

In this tutorial, the RESTful Web Services are created for [*Tutorial (Todo Application)*].

Therefore, if [*Tutorial (Todo Application)*] project is not exists, re-create project by executing [*Tutorial (Todo Application)*].

Note: If project is re-created by executing [*Tutorial (Todo Application)*], it is possible to proceed further this tutorial by performing re-creation till the domain layer creation.

10.2.3 REST API creation

In this tutorial, creating REST API for publishing the data on Web which are managed in the todo table (here onwards called as [Todo Resources]).

API name	HTTP method	Path	Status code	Description
GET Todos	GET	`/api/v1/todos`	200 (OK)	Fetch all records of Todo Resource.
POST Todos	POST	`/api/v1/todos`	201 (Created)	Create new Todo Resource.
GET Todo	GET	`/api/v1/todos/{todoId}`	200 (OK)	Fetch one record of Todo Resource.
PUT Todo	PUT	`/api/v1/todos/{todoId}`	200 (OK)	Update Todo Resource in completed status
DELETE Todo	DELETE	`/api/v1/todos/{todoId}`	204 (No Content)	Delete Todo Resource.

Tip: The `{todoId}` included in path is called as path variable and can deal with any changeable value. The `GET /api/v1/todos/123` and `GET /api/v1/todos/456` can be handle with same API using the path variable.

In this tutorial, We are dealing with ID (Todo ID) as the path variable in order to uniquely identifying the Todo.

API specification

Indicated the REST API Interface specifications using specific example of the HTTP requests and responses in this tutorial.

HTTP headers which are not essential have been excluded from the example.

GET Todos

[Request]

```
> GET /todo/api/v1/todos HTTP/1.1
```

[Response]

Return list of created Todo Resource in JSON format.

```
< HTTP/1.1 200 OK
< Content-Type: application/json;charset=UTF-8
<
[{"todoId":"9aef3ee3-30d4-4a7c-be4a-bc184ca1d558","todoTitle":"Hello World!",
↪"finished":false,"createdAt":"2014-02-25T02:21:48.493+0000"}]
```

POST Todos

[Request]

Specify newly creation Todo Resource content (Title) in JSON format.

```
> POST /todo/api/v1/todos HTTP/1.1
> Content-Type: application/json
> Content-Length: 29
>
{"todoTitle": "Study Spring"}
```

[Response]

Return created Todo Resource in JSON format.

```
< HTTP/1.1 201 Created
< Content-Type: application/json;charset=UTF-8
```

```
<
{"todoId":"d6101d61-b22c-48ee-9110-e106af6a1404","todoTitle":"Study Spring",
↪"finished":false,"createdAt":"2014-02-25T04:05:58.752+0000"}
```

GET Todo

[Request]

Specify ID of the Todo Resource in [`todoId`] path variable that you want to fetch.

In below example, `9aef3ee3-30d4-4a7c-be4a-bc184ca1d558` is specified in [`todoId`] path variable.

```
> GET /todo/api/v1/todos/9aef3ee3-30d4-4a7c-be4a-bc184ca1d558 HTTP/1.1
```

[Response]

Return Todo Resource in JSON format that matches with the [`todoId`] path variable.

```
< HTTP/1.1 200 OK
< Content-Type: application/json;charset=UTF-8
<
{"todoId":"9aef3ee3-30d4-4a7c-be4a-bc184ca1d558","todoTitle":"Hello World!",
↪"finished":false,"createdAt":"2014-02-25T02:21:48.493+0000"}
```

PUT Todo

[Request]

Specify ID of the Todo Resource in [`todoId`] path variable that you want to update.

In the PUT Todo, interface specification does not receive the request BODY because Todo Resource is only updating into completion state.

```
> PUT /todo/api/v1/todos/9aef3ee3-30d4-4a7c-be4a-bc184ca1d558 HTTP/1.1
```

[Response]

Return Todo Resource in JSON format that matches with the [`todoId`] path variable after updating in completed status (`true` of `finished` field).

```
< HTTP/1.1 200 OK
< Content-Type: application/json;charset=UTF-8
<
{"todoId":"9aef3ee3-30d4-4a7c-be4a-bc184ca1d558","todoTitle":"Hello World!",
↪"finished":true,"createdAt":"2014-02-25T02:21:48.493+0000"}
```

DELETE Todo

[Request]

Specify ID of the Todo Resource in [`todoId`] path variable that you want to delete.

```
> DELETE /todo/api/v1/todos/9aef3ee3-30d4-4a7c-be4a-bc184ca1d558 HTTP/1.1
```

[Response]

In the DELETE Todo, since the Todo Resource is deleted and resource that can return is no longer exists, interface specification does not return the response BODY.

```
< HTTP/1.1 204 No Content
```

Error Response

Return error in JSON format in case of any error occurs in REST API.

The response specification of typical errors are described below.

Error patterns other than the below are also exists but description in the tutorial are omitted.

In the *Tutorial (Todo Application)*, error messages are hardcoded in the program but in this tutorial, it is modified such a way that the error messages are retrieved from the property file based on error code.

[Response specification at the time of input check error]

```
< HTTP/1.1 400 Bad Request
< Content-Type: application/json;charset=UTF-8
<
{"code":"E400","message":"[E400] The requested Todo contains invalid values.",
 ↪"details":[{"code":"NotNull","message":"todoTitle may not be null.",target:
 ↪"todoTitle"}]}
```

[Response specification at the time of business error]

```
< HTTP/1.1 409 Conflict
< Content-Type: application/json;charset=UTF-8
<
{"code":"E002","message":"[E002] The requested Todo is already finished.␣
 ↪(id=353fb5db-151a-4696-9b4a-b958358a5ab3)"}
```

[Response specification at the time of resources undetected]

```
< HTTP/1.1 404 Not Found
< Content-Type: application/json;charset=UTF-8
<
{"code":"E404","message":"[E404] The requested Todo is not found. (id=353fb5db-
 ↪151a-4696-9b4a-b958358a5ab2)"}
```

[Response specification at the time of system error]

```
< HTTP/1.1 500 Internal Server Error
< Content-Type: application/json;charset=UTF-8
<
{"code":"E500","message":"[E500] System error occurred."}
```

DispatcherServlet for REST API

First, add the definition of `DispatcherServlet` for processing the REST API request.

Modification of web.xml

Add configuration pertaining to REST API.

`src/main/webapp/WEB-INF/web.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/
↪ns/javaee/web-app_3_0.xsd"
    version="3.0">

    <listener>
        <listener-class>org.springframework.web.context.ContextLoaderListener</
↪listener-class>
    </listener>
    <context-param>
        <param-name>contextConfigLocation</param-name>
        <!-- Root ApplicationContext -->
        <param-value>
            classpath*:META-INF/spring/applicationContext.xml
            classpath*:META-INF/spring/spring-security.xml
        </param-value>
    </context-param>

    <listener>
        <listener-class>org.terasoluna.gfw.web.logging.
↪HttpSessionEventLoggingListener</listener-class>
    </listener>

    <filter>
        <filter-name>MDCClearFilter</filter-name>
        <filter-class>org.terasoluna.gfw.web.logging.mdc.MDCClearFilter</filter-
↪class>
    </filter>
    <filter-mapping>
        <filter-name>MDCClearFilter</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>

    <filter>
        <filter-name>exceptionLoggingFilter</filter-name>
        <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-
↪class>
    </filter>
    <filter-mapping>
        <filter-name>exceptionLoggingFilter</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>

    <filter>
        <filter-name>XTrackMDCPutFilter</filter-name>
        <filter-class>org.terasoluna.gfw.web.logging.mdc.XTrackMDCPutFilter</
↪filter-class>
    </filter>
```

```
    <filter-mapping>
        <filter-name>XTrackMDCPutFilter</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>

    <filter>
        <filter-name>CharacterEncodingFilter</filter-name>
        <filter-class>org.springframework.web.filter.CharacterEncodingFilter</
↪filter-class>
        <init-param>
            <param-name>encoding</param-name>
            <param-value>UTF-8</param-value>
        </init-param>
        <init-param>
            <param-name>forceEncoding</param-name>
            <param-value>true</param-value>
        </init-param>
    </filter>
    <filter-mapping>
        <filter-name>CharacterEncodingFilter</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>

    <filter>
        <filter-name>springSecurityFilterChain</filter-name>
        <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-
↪class>
    </filter>
    <filter-mapping>
        <filter-name>springSecurityFilterChain</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>

    <!-- (1) -->
    <servlet>
        <servlet-name>restApiServlet</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-
↪class>
        <init-param>
            <param-name>contextConfigLocation</param-name>
            <!-- ApplicationContext for Spring MVC (REST) -->
            <param-value>classpath*:META-INF/spring/spring-mvc-rest.xml</param-
↪value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>

    <!-- (2) -->
    <servlet-mapping>
        <servlet-name>restApiServlet</servlet-name>
```

```
        <url-pattern>/api/v1/*</url-pattern>
    </servlet-mapping>

    <servlet>
        <servlet-name>appServlet</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-
↪class>
        <init-param>
            <param-name>contextConfigLocation</param-name>
            <!-- ApplicationContext for Spring MVC -->
            <param-value>classpath*:META-INF/spring/spring-mvc.xml</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>appServlet</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>

    <jsp-config>
        <jsp-property-group>
            <url-pattern>*.jsp</url-pattern>
            <el-ignored>false</el-ignored>
            <page-encoding>UTF-8</page-encoding>
            <scripting-invalid>false</scripting-invalid>
            <include-prelude>/WEB-INF/views/common/include.jsp</include-prelude>
        </jsp-property-group>
    </jsp-config>

    <error-page>
        <error-code>500</error-code>
        <location>/WEB-INF/views/common/error/systemError.jsp</location>
    </error-page>

    <error-page>
        <error-code>404</error-code>
        <location>/WEB-INF/views/common/error/resourceNotFoundError.jsp</location>
    </error-page>

    <error-page>
        <exception-type>java.lang.Exception</exception-type>
        <location>/WEB-INF/views/common/error/unhandledSystemError.html</location>
    </error-page>

    <session-config>
        <!-- 30min -->
        <session-timeout>30</session-timeout>
        <cookie-config>
            <http-only>true</http-only>
            <!-- <secure>true</secure> -->
```

```
        </cookie-config>
        <tracking-mode>COOKIE</tracking-mode>
    </session-config>

</web-app>
```

Sr. No	Description
(1)	Specify the SpringMVC configuration file for REST at the initialization parameter [contextConfigLocation]. In this tutorial, [META-INF/spring/spring-mvc-rest.xml] is specified located at class path.
(2)	Specify the URL pattern that maps to the DispatcherServlet for REST API at <url-pattern> element. In this tutorial, if it starts from /api/v1/, request is considered as a REST API request and mapped with the DispatcherServlet for REST API.

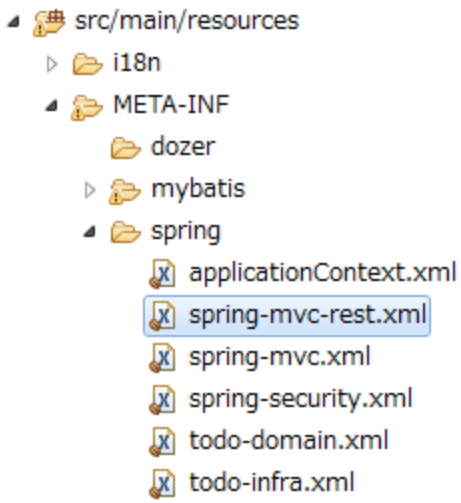
Creation of spring-mvc-rest.xml

Spring MVC configuration file for REST is created by copying the src/main/resources/META-INF/spring/spring-mvc.xmlfile.

The definition of SpringMVC configuration for REST file will be as follows.

src/main/resources/META-INF/spring/spring-mvc-rest.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:mvc="http://www.springframework.org/schema/mvc"
    xmlns:util="http://www.springframework.org/schema/util"
    xmlns:aop="http://www.springframework.org/schema/aop"
    xsi:schemaLocation="http://www.springframework.org/schema/mvc http://www.
↪springframework.org/schema/mvc/spring-mvc.xsd
        http://www.springframework.org/schema/beans http://www.springframework.org/
↪schema/beans/spring-beans.xsd
```




```
        http://www.springframework.org/schema/util http://www.springframework.org/
↪schema/util/spring-util.xsd
        http://www.springframework.org/schema/context http://www.springframework.
↪org/schema/context/spring-context.xsd
        http://www.springframework.org/schema/aop http://www.springframework.org/
↪schema/aop/spring-aop.xsd
    ">

    <context:property-placeholder
        location="classpath*:/META-INF/spring/*.properties" />

    <mvc:annotation-driven>
        <mvc:argument-resolvers>
            <bean
                class="org.springframework.data.web.
↪PageableHandlerMethodArgumentResolver" />
            <bean
                class="org.springframework.security.web.method.annotation.
↪AuthenticationPrincipalArgumentResolver" />
        </mvc:argument-resolvers>
        <!-- workaround to CVE-2016-5007. -->
        <mvc:path-matching path-matcher="pathMatcher" />
        <mvc:message-converters register-defaults="false">
            <!-- (1) -->
            <bean
                class="org.springframework.http.converter.json.
↪MappingJackson2HttpMessageConverter">
                <!-- (2) -->
                <property name="objectMapper">
                    <bean class="com.fasterxml.jackson.databind.ObjectMapper">
                        <property name="dateFormat">
                            <!-- (3) -->
                            <bean class="com.fasterxml.jackson.databind.util.
↪StdDateFormat"/>
```

```
                        </property>
                    </bean>
                </property>
            </bean>
        </mvc:message-converters>
    </mvc:annotation-driven>

    <mvc:default-servlet-handler />

    <context:component-scan base-package="todo.api" /> <!-- (3) -->

    <mvc:interceptors>
        <mvc:interceptor>
            <mvc:mapping path="/**" />
            <mvc:exclude-mapping path="/resources/**" />
            <mvc:exclude-mapping path="/**/*.html" />
            <bean
                class="org.terasoluna.gfw.web.logging.TraceLoggingInterceptor" />
        </mvc:interceptor>
        <mvc:interceptor>
            <mvc:mapping path="/**" />
            <mvc:exclude-mapping path="/resources/**" />
            <mvc:exclude-mapping path="/**/*.html" />
            <bean
                class="org.terasoluna.gfw.web.token.transaction.
↪TransactionTokenInterceptor" />
        </mvc:interceptor>
        <mvc:interceptor>
            <mvc:mapping path="/**" />
            <mvc:exclude-mapping path="/resources/**" />
            <mvc:exclude-mapping path="/**/*.html" />
            <bean class="org.terasoluna.gfw.web.codelist.CodeListInterceptor">
                <property name="codeListIdPattern" value="CL_.+" />
            </bean>
        </mvc:interceptor>
        <!--  REMOVE THIS LINE IF YOU USE JPA
        <mvc:interceptor>
            <mvc:mapping path="/**" />
            <mvc:exclude-mapping path="/resources/**" />
            <mvc:exclude-mapping path="/**/*.html" />
            <bean
                class="org.springframework.orm.jpa.support.
↪OpenEntityManagerInViewInterceptor" />
        </mvc:interceptor>
            REMOVE THIS LINE IF YOU USE JPA  -->
    </mvc:interceptors>

    <!-- Settings View Resolver. -->
    <mvc:view-resolvers>
        <mvc:jsp prefix="/WEB-INF/views/" />
```

```
    </mvc:view-resolvers>

    <bean id="requestDataValueProcessor"
        class="org.terasoluna.gfw.web.mvc.support.
↪CompositeRequestDataValueProcessor">
        <constructor-arg>
            <util:list>
                <bean
                    class="org.springframework.security.web.servlet.support.csrf.
↪CsrfRequestDataValueProcessor" />
                <bean
                    class="org.terasoluna.gfw.web.token.transaction.
↪TransactionTokenRequestDataValueProcessor" />
            </util:list>
        </constructor-arg>
    </bean>

    <!-- Setting Exception Handling. -->
    <!-- Exception Resolver. -->
    <bean id="systemExceptionResolver"
        class="org.terasoluna.gfw.web.exception.SystemExceptionResolver">
        <property name="exceptionCodeResolver" ref="exceptionCodeResolver" />
        <!-- Setting and Customization by project. -->
        <property name="order" value="3" />
        <property name="exceptionMappings">
            <map>
                <entry key="ResourceNotFoundException" value="common/error/
↪resourceNotFoundError" />
                <entry key="BusinessException" value="common/error/businessError" /
↪>
                <entry key="InvalidTransactionTokenException" value="common/error/
↪transactionTokenError" />
                <entry key=".DataAccessException" value="common/error/
↪dataAccessError" />
            </map>
        </property>
        <property name="statusCodes">
            <map>
                <entry key="common/error/resourceNotFoundError" value="404" />
                <entry key="common/error/businessError" value="409" />
                <entry key="common/error/transactionTokenError" value="409" />
                <entry key="common/error/dataAccessError" value="500" />
            </map>
        </property>
        <property name="defaultErrorView" value="common/error/systemError" />
        <property name="defaultStatusCode" value="500" />
    </bean>
    <!-- Setting AOP. -->
    <bean id="handlerExceptionResolverLoggingInterceptor"
        class="org.terasoluna.gfw.web.exception.
↪HandlerExceptionResolverLoggingInterceptor">
```

```
        <property name="exceptionLogger" ref="exceptionLogger" />
    </bean>
    <aop:config>
        <aop:advisor advice-ref="handlerExceptionResolverLoggingInterceptor"
            pointcut="execution(* org.springframework.web.servlet.
↪HandlerExceptionResolver.resolveException(..))" />
    </aop:config>

    <!-- Setting PathMatcher. -->
    <bean id="pathMatcher" class="org.springframework.util.AntPathMatcher">
        <property name="trimTokens" value="false" />
    </bean>

</beans>
```

Sr. No	Description
(1)	Set the class(`org.springframework.http.converter.HttpMessageConverter`) to serialize/de-serialize the JavaBean dealing with arguments and return values of the Controller at `<mvc:message-converters>`. Multiple `HttpMessageConverter` can be configured but, since only JSON is used in this tutorial only `MappingJackson2HttpMessageConverter` is specified.
(2)	Specify the `ObjectMapper`(Component for conversion of [JSON <-> JavaBean]) that is provided by Jackson into the `objectMapper` property of the `MappingJackson2HttpMessageConverter`. In this tutorial, Data format customized `ObjectMapper` is specified. `objectMapper`property can be omitted if customization is not required.
(3)	Specify format of the Date field into `dateFormat` property of `ObjectMapper`. In this tutorial, ISO-8601 format used while serializing `java.util.Date` object. If you want to use ISO-8601 format while serializing `Date` object, it can be implemented by configuring the `com.fasterxml.jackson.databind.util.StdDateFormat`.
(4)	Scan the components under the package of the REST API In this tutorial, the package of REST API is `todo.api`. Although the Controllers for the screen transition had been stored under `app` package, Controllers for REST API are recommended to store under `api` package.

Definition of Spring Security for REST API

Disabled the CSRF protection in REST API created in this tutorial.

CSRF protection is required even in Web application of the REST API. However, the purpose of this tutorial is not CSRF measures hence ignored the explanation.

If you disable the CSRF protection, the use of session is not required.

Therefore, in this tutorial, adopted an architecture that does not use the session(stateless architecture) and disable the CSRF measures.

The use of session and CSRF measures can be avoided by adding the following settings.

`src/main/resources/META-INF/spring/spring-security.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:sec="http://www.springframework.org/schema/security"
    xsi:schemaLocation="
        http://www.springframework.org/schema/security http://www.springframework.
↪org/schema/security/spring-security.xsd
        http://www.springframework.org/schema/beans http://www.springframework.org/
↪schema/beans/spring-beans.xsd
    ">

    <sec:http pattern="/resources/**" security="none"/>

    <!-- (1) -->
    <sec:http pattern="/api/v1/**" security="none"/>

    <sec:http>
        <sec:form-login/>
        <sec:logout/>
        <sec:access-denied-handler ref="accessDeniedHandler"/>
        <sec:custom-filter ref="userIdMDCPutFilter" after="ANONYMOUS_FILTER"/>
        <sec:session-management />
    </sec:http>

    <sec:authentication-manager />

    <!-- CSRF Protection -->
    <bean id="accessDeniedHandler"
        class="org.springframework.security.web.access.
↪DelegatingAccessDeniedHandler">
        <constructor-arg index="0">
            <map>
                <entry
                    key="org.springframework.security.web.csrf.
↪InvalidCsrfTokenException">
                    <bean
                        class="org.springframework.security.web.access.
↪AccessDeniedHandlerImpl">
                        <property name="errorPage"
```

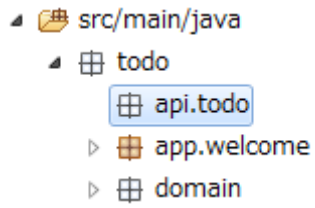
```
                            value="/WEB-INF/views/common/error/
↪invalidCsrfTokenError.jsp" />
                    </bean>
                </entry>
                <entry
                    key="org.springframework.security.web.csrf.
↪MissingCsrfTokenException">
                    <bean
                        class="org.springframework.security.web.access.
↪AccessDeniedHandlerImpl">
                        <property name="errorPage"
                            value="/WEB-INF/views/common/error/
↪missingCsrfTokenError.jsp" />
                    </bean>
                </entry>
            </map>
        </constructor-arg>
        <constructor-arg index="1">
            <bean
                class="org.springframework.security.web.access.
↪AccessDeniedHandlerImpl">
                <property name="errorPage"
                    value="/WEB-INF/views/common/error/accessDeniedError.jsp" />
            </bean>
        </constructor-arg>
    </bean>

    <!-- Put UserID into MDC -->
    <bean id="userIdMDCPutFilter" class="org.terasoluna.gfw.security.web.logging.
↪UserIdMDCPutFilter">
    </bean>

</beans>
```

Sr. No	Description
(1)	Add the definition of Spring Security for REST API. Specify the URL pattern of the REST API request path at `pattern` attribute of the `<sec:http>` element. In this tutorial, request path starts with `/api/v1/` is considered as a REST API request. Furthermore, session is no longer used in the processing of Spring Security by specifying `stateless` at `create-session` attribute. Specify `disabled="true"` in `<sec:csrf>` element for invalidating CSRF countermeasures.

Creation of REST API package

Create a package that stores the REST API classes.

The name of the root package is `api` that contains the REST API classes and recommended to create a package of each resource (lowercase resource name) under it.
Since the name of the resource is Todo in this tutorial, the `todo.api.todo` package is created.



Note: Usually following three types of classes are stored in the created package. The following naming rules are recommended for the classes.

- `[Resource name]Resource`
- `[Resource name]RestController`
- `[Resource name]Helper` (if required)

Since name of the resource is Todo in this tutorial,

- `TodoResource`
- `TodoRestController`

is created

The `TodoRestHelper` is not created in this tutorial.

Creation of Resource class

Create `TodoResource` class for implementing Todo Resource.

In this guide line, Java Bean that represent the JSON(or XML) for input and output of the REST API is called as **Resource class**.

src/main/java/todo/api/todo/TodoResource.java

```
package todo.api.todo;

import java.io.Serializable;
import java.util.Date;

import javax.validation.constraints.NotNull;
import javax.validation.constraints.Size;

public class TodoResource implements Serializable {

    private static final long serialVersionUID = 1L;

    private String todoId;

    @NotNull
    @Size(min = 1, max = 30)
    private String todoTitle;

    private boolean finished;

    private Date createdAt;

    public String getTodoId() {
        return todoId;
    }

    public void setTodoId(String todoId) {
        this.todoId = todoId;
    }

    public String getTodoTitle() {
        return todoTitle;
    }

    public void setTodoTitle(String todoTitle) {
        this.todoTitle = todoTitle;
    }

    public boolean isFinished() {
        return finished;
    }

    public void setFinished(boolean finished) {
        this.finished = finished;
```

```
    }

    public Date getCreatedAt() {
        return createdAt;
    }

    public void setCreatedAt(Date createdAt) {
        this.createdAt = createdAt;
    }
}
```

Note: The reason for creating a Resource class in spite of the existence of the DomainObject class (`Todo` class in this tutorial), business process is not consistent with the interface to be used in the input and output of the client.

If it is used wrongly, application layer will be impacted to the domain layer and also decrease the maintainability. It is recommended to perform the data conversion using BeanMapper such as Dozer by creating DomainObject and Resource class separately.

The role of the Resource class is similar to the Form class but eventually it is differing like Form class represent the `<form>` tag of HTML in JavaBean and Resource class is the input and output of the REST API in JavaBean.

However, it is a JavaBean having annotation of the Bean Validation and the Controller class is approximately the same as the Form class because stored in the same package.

Creation of Controller class

Create a `TodoRestController` class that provides a REST API of `TodoResource`.

`src/main/java/todo/api/todo/TodoRestController.java`

```
package todo.api.todo;

import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController // (1)
@RequestMapping("todos") // (2)
public class TodoRestController {

}
```

Sr. No	Description
(1)	Specify the `@RestController` annotation. Refer to the *Creation of RestController class* for the details of `@RestController`.
(2)	Specify the resource path. Since `/api/v1/` is defined in web.xml, it is mapped with the `/<contextPath>/api/v1/todos` path by perform this setting.

Implementation of GET Todos

Implement the processing of API(GET Todos) into `getTodos` method of `TodoRestController` that fetches all records of created Todo Resource.

`src/main/java/todo/api/todo/TodoRestController.java`

```
package todo.api.todo;

import java.util.ArrayList;
import java.util.Collection;
import java.util.List;

import javax.inject.Inject;

import org.dozer.Mapper;
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.ResponseStatus;
import org.springframework.web.bind.annotation.RestController;

import todo.domain.model.Todo;
import todo.domain.service.todo.TodoService;

@RestController
@RequestMapping("todos")
public class TodoRestController {

    @Inject
    TodoService todoService;
```

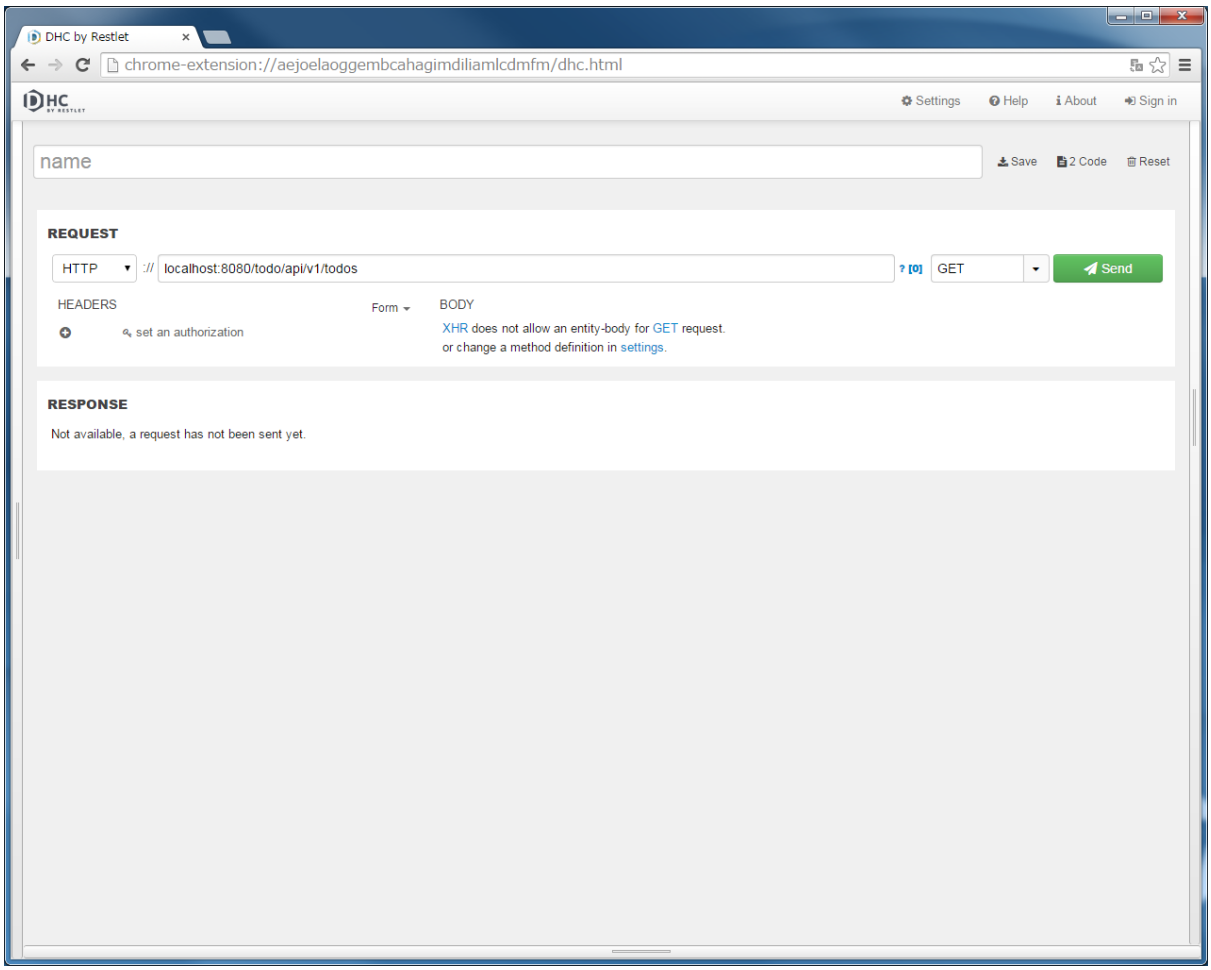
```
    @Inject
    Mapper beanMapper;

    @RequestMapping(method = RequestMethod.GET) // (1)
    @ResponseStatus(HttpStatus.OK) // (2)
    public List<TodoResource> getTodos() {
        Collection<Todo> todos = todoService.findAll();
        List<TodoResource> todoResources = new ArrayList<>();
        for (Todo todo : todos) {
            todoResources.add(beanMapper.map(todo, TodoResource.class)); // (3)
        }
        return todoResources; // (4)
    }

}
```

Sr. No	Description
(1)	Set the `RequestMethod.GET` to `method` attribute for handling the GET request.
(2)	Specify `@ResponseStatus` annotation to the HTTP status code for response. To set "200 OK" as a HTTP status, set the `HttpStatus.OK` to the `value` attribute.
(3)	Converting `Todo` object returned from `findAll` method of `TodoService` into `TodoResource` object type that represent JSON response. It is convenient to use the `org.dozer.Mapper` interface of Dozer for converting `Todo` and `TodoResource`.
(4)	By returning the `List<TodoResource>` object, it is serialized into JSON by `MappingJackson2HttpMessageConverter` defined in `spring-mvc-rest.xml`.

Check the operation of the implemented API by booting Application Server.

Access the REST API(Get Todos).

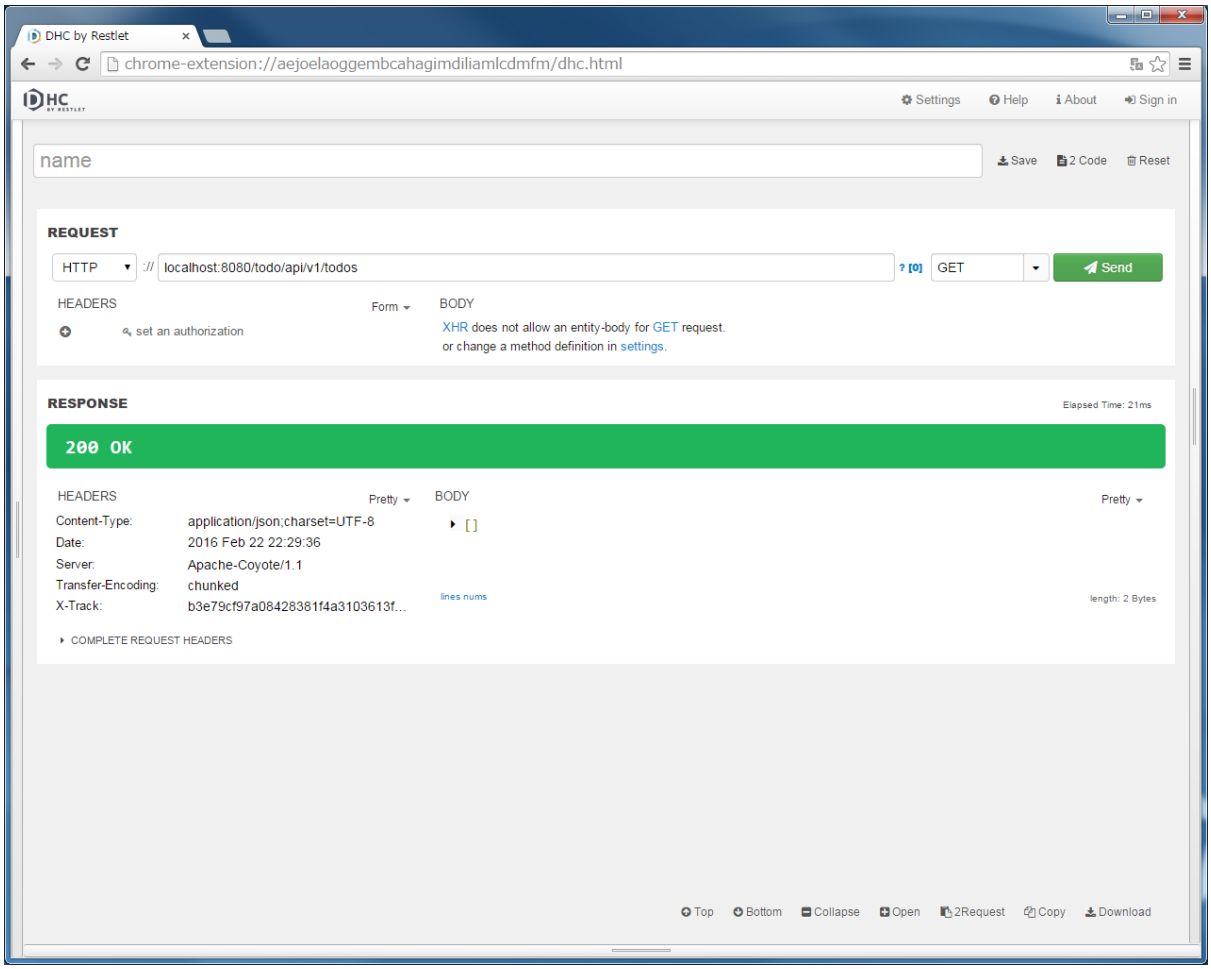
Open the DHC, enter "`localhost:8080/todo/api/v1/todos`" in the URL, specify GET in method and click the “Send” button.



Displays JSON execution results in [BODY] of the [RESPONSE] as follows.

Since data is not registered at present, an empty array `[]` is returned.

Since the Spring Security setting has been changed to not use the session therefore want to focus on the point that "`Set-Cookie:  JSESSIONID=xxxx`" is not exists in the [RESPONSE] [HEADERS].



Implementation of POST Todos

Implement the processing of API(GET Todos) into `postTodos` method of `TodoRestController` that create new Todo Resource.

`src/main/java/todo/api/todo/TodoRestController.java`

```
package todo.api.todo;

import java.util.ArrayList;
import java.util.Collection;
import java.util.List;

import javax.inject.Inject;

import org.dozer.Mapper;
import org.springframework.http.HttpStatus;
```

```
import org.springframework.validation.annotation.Validated;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.ResponseStatus;
import org.springframework.web.bind.annotation.RestController;

import todo.domain.model.Todo;
import todo.domain.service.todo.TodoService;

@RestController
@RequestMapping("todos")
public class TodoRestController {

    @Inject
    TodoService todoService;
    @Inject
    Mapper beanMapper;

    @RequestMapping(method = RequestMethod.GET)
    @ResponseStatus(HttpStatus.OK)
    public List<TodoResource> getTodos() {
        Collection<Todo> todos = todoService.findAll();
        List<TodoResource> todoResources = new ArrayList<>();
        for (Todo todo : todos) {
            todoResources.add(beanMapper.map(todo, TodoResource.class));
        }
        return todoResources;
    }

    @RequestMapping(method = RequestMethod.POST) // (1)
    @ResponseStatus(HttpStatus.CREATED) // (2)
    public TodoResource postTodos(@RequestBody @Validated TodoResource␣
↪todoResource) { // (3)
        Todo createdTodo = todoService.create(beanMapper.map(todoResource, Todo.
↪class)); // (4)
        TodoResource createdTodoResponse = beanMapper.map(createdTodo,␣
↪TodoResource.class); // (5)
        return createdTodoResponse; // (6)
    }

}
```

Sr. No	Description
(1)	Set the `RequestMethod.POST` to `method` attribute for handling the POST request.
(2)	Specify `@ResponseStatus` annotation to the HTTP status code for response. To set "201 Created" as a HTTP status, set the `HttpStatus.CREATED` to the `value` attribute.
(3)	In order to map the HTTP request Body(JSON) with JavaBean, grant `@RequestBody` annotation to the mapping targeted `TodoResource` class. Furthermore, grant `@Validated` annotation for input check. It is necessary to handle exception separately.
(4)	Create new Todo resource by executing `create` method of `TodoService` after converting `TodoResource` into `Todo` class.
(5)	Converting `Todo` object created by `create` method of `TodoService` into `TodoResource` type that represent JSON response.
(6)	By returning the `TodoResource` object, it is serialized into JSON by `MappingJackson2HttpMessageConverter` defined in `spring-mvc-rest.xml`.

Check the operation of the implemented API using DHC.

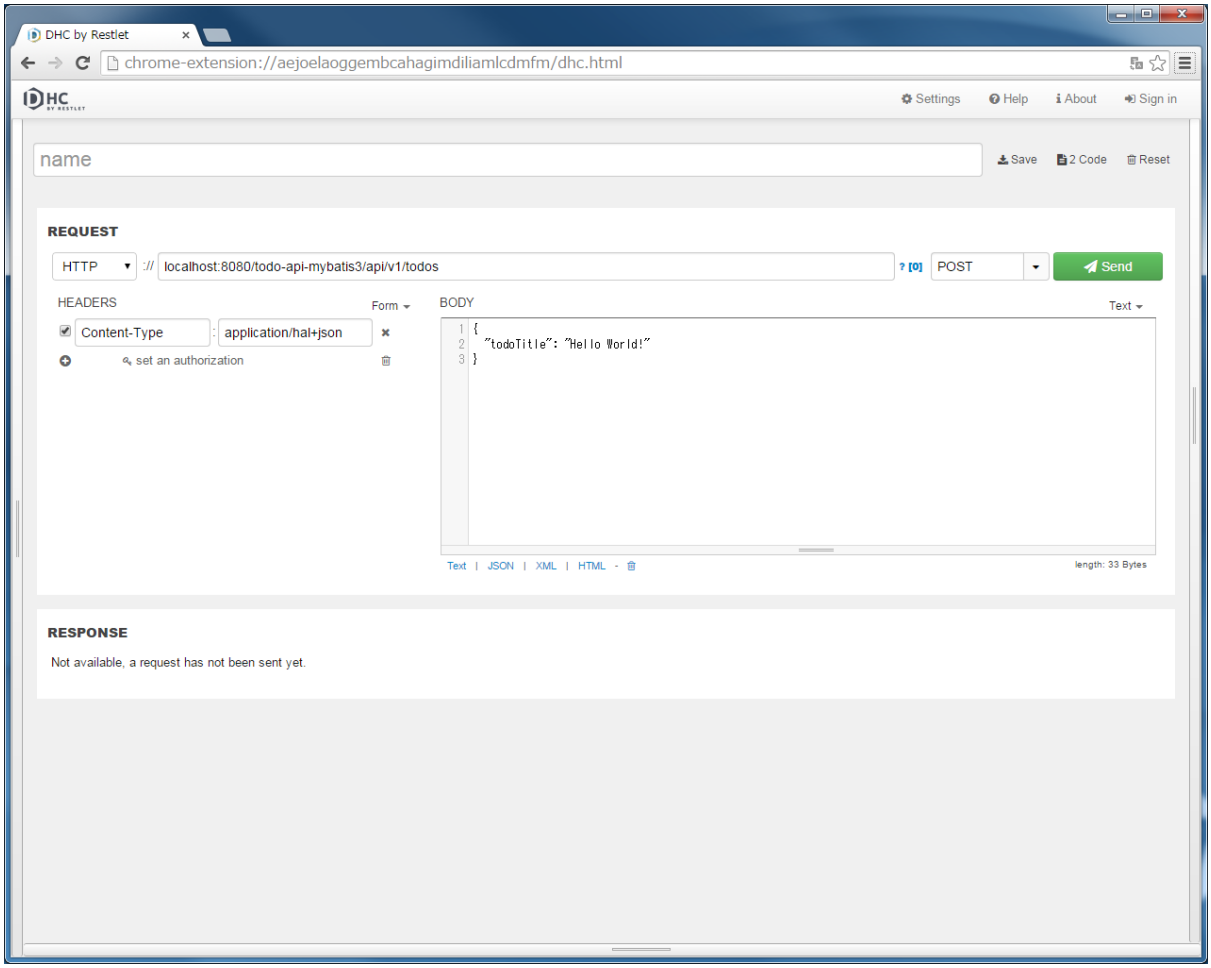
Open the DHC, enter "`localhost:8080/todo/api/v1/todos`" in the URL and specify POST in method.

Enter the following JSON into [BODY] of the [REQUEST].

```
{
  "todoTitle": "Hello World!"
```

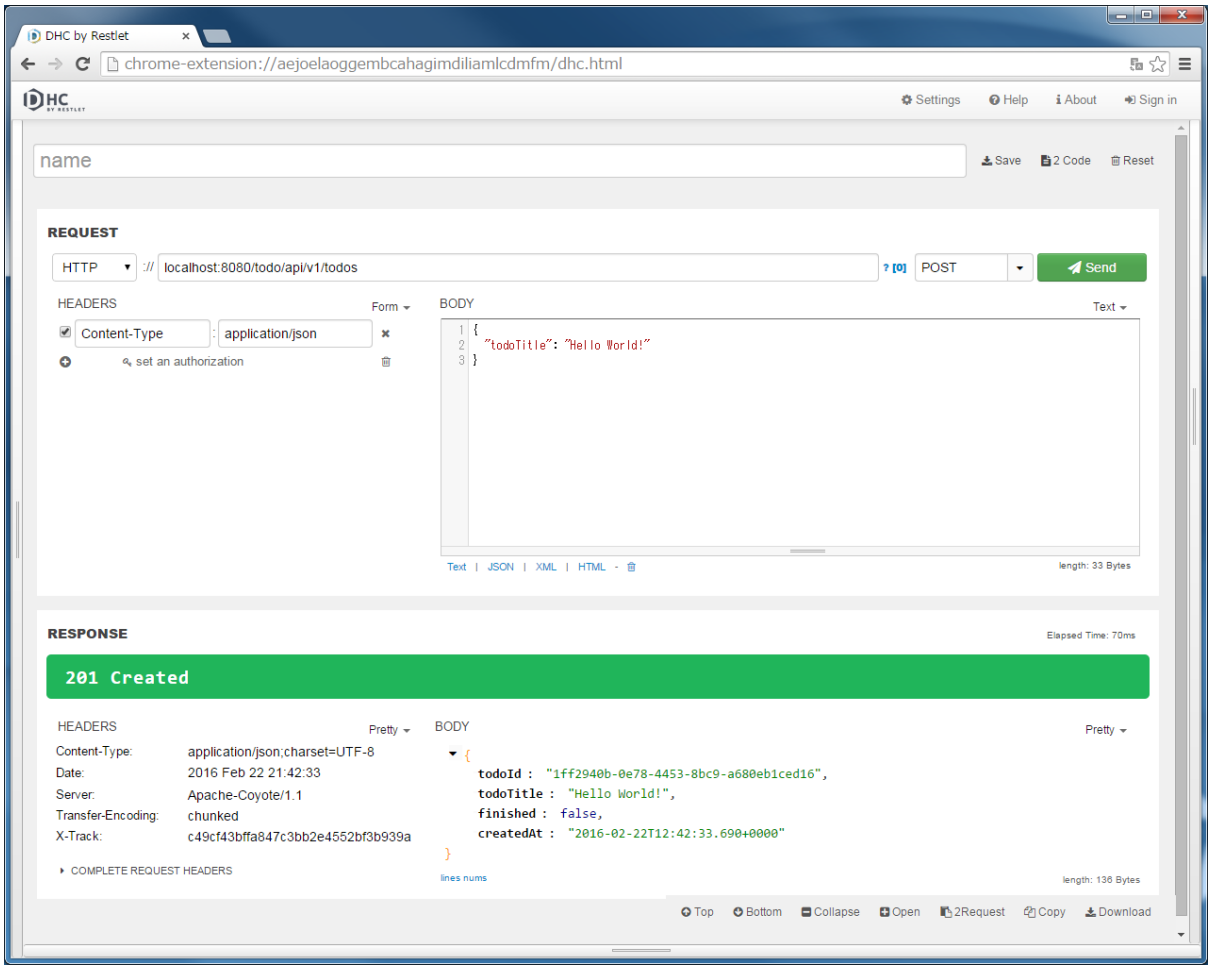
```
}
```

Furthermore, Add HTTP header by [+] button of [REQUEST] [HEADERS] and click “Send” button after setting [`application/json`] in the [`Content-Type`].



HTTP status returned “201 Created” and JSON of the newly created Todo resource displays in [Body] of [RESPONSE] part.

If GET Todos gets executed now, newly created Todo Resource returns as an array.



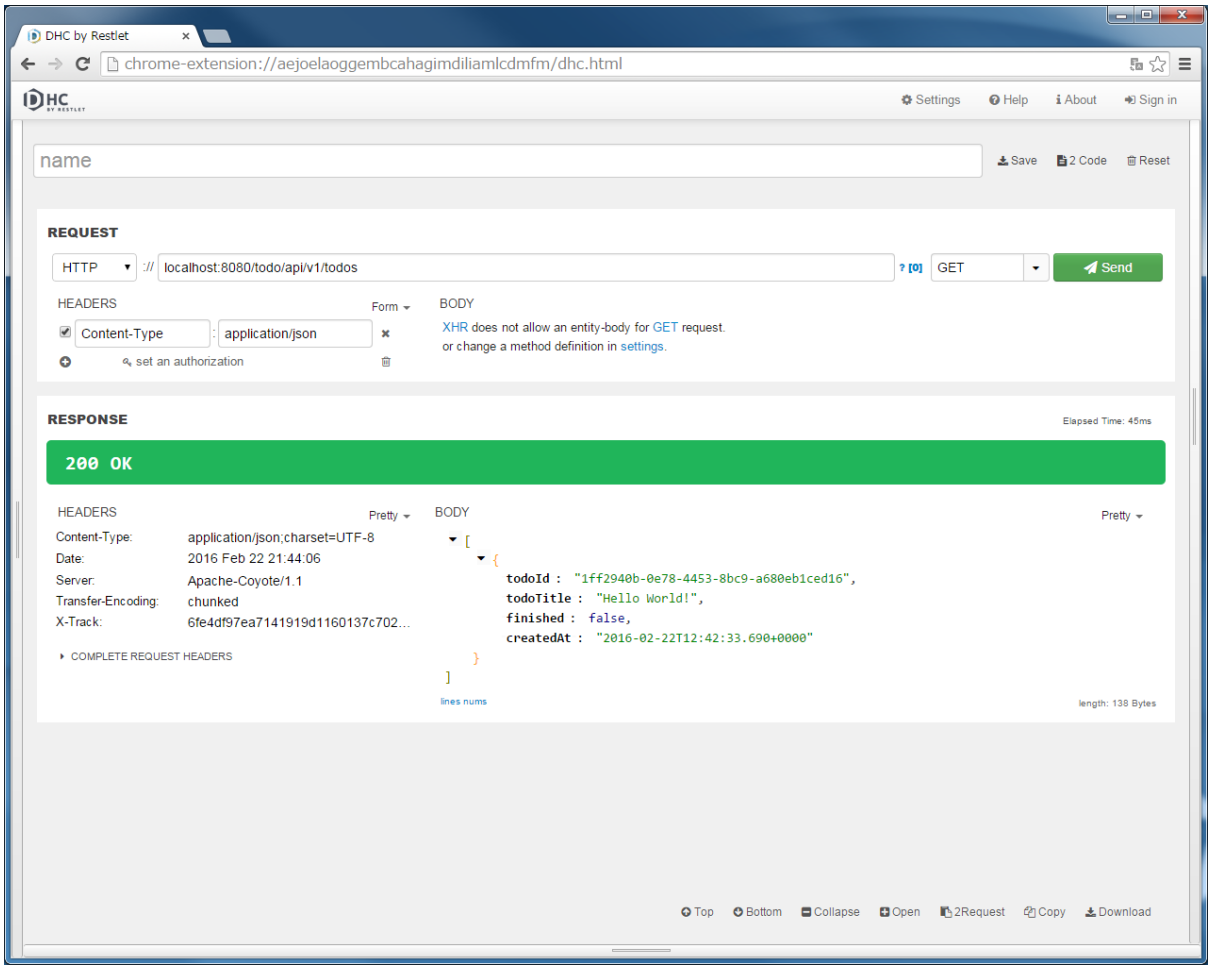
Implementation of GET Todo

Since method (findOne) for retrieving single item is not created in `TodoService` of the *Tutorial (Todo Application)*, add the following highlighted parts in `TodoService` and `TodoServiceImpl`.

Add the definition of `findOne` method.

`src/main/java/todo/domain/service/todo/TodoService.java`

```
package todo.domain.service.todo;

import java.util.Collection;

import todo.domain.model.Todo;
```




```
public interface TodoService {
    Collection<Todo> findAll();

    Todo findOne(String todoId);

    Todo create(Todo todo);

    Todo finish(String todoId);

    void delete(String todoId);
}
```

Set a read-only transaction that is initiated at the time of calling `findOne` method.

`src/main/java/todo/domain/service/todo/TodoServiceImpl.java`

```
package todo.domain.service.todo;

import java.util.Collection;
import java.util.Date;
import java.util.UUID;

import javax.inject.Inject;

import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import org.terasoluna.gfw.common.exception.BusinessException;
import org.terasoluna.gfw.common.exception.ResourceNotFoundException;
import org.terasoluna.gfw.common.message.ResultMessage;
import org.terasoluna.gfw.common.message.ResultMessages;

import todo.domain.model.Todo;
import todo.domain.repository.todo.TodoRepository;

@Service
@Transactional
public class TodoServiceImpl implements TodoService {

    private static final long MAX_UNFINISHED_COUNT = 5;

    @Inject
    TodoRepository todoRepository;

    @Override
    @Transactional(readOnly = true)
    public Todo findOne(String todoId) {
        Todo todo = todoRepository.findOne(todoId);
        if (todo == null) {
            ResultMessages messages = ResultMessages.error();
            messages.add(ResultMessage
                    .fromText("[E404] The requested Todo is not found. (id="
                            + todoId + ")"));
            throw new ResourceNotFoundException(messages);
        }
        return todo;
    }

    @Override
    @Transactional(readOnly = true)
    public Collection<Todo> findAll() {
        return todoRepository.findAll();
    }

    @Override
    public Todo create(Todo todo) {
        long unfinishedCount = todoRepository.countByFinished(false);
```

```
        if (unfinishedCount >= MAX_UNFINISHED_COUNT) {
            ResultMessages messages = ResultMessages.error();
            messages.add(ResultMessage
                    .fromText("[E001] The count of un-finished Todo must not be␣
↪over "
                            + MAX_UNFINISHED_COUNT + "."));
            throw new BusinessException(messages);
        }

        String todoId = UUID.randomUUID().toString();
        Date createdAt = new Date();

        todo.setTodoId(todoId);
        todo.setCreatedAt(createdAt);
        todo.setFinished(false);

        todoRepository.create(todo);
        /* REMOVE THIS LINE IF YOU USE JPA
            todoRepository.save(todo);
           REMOVE THIS LINE IF YOU USE JPA */

        return todo;
    }

    @Override
    public Todo finish(String todoId) {
        Todo todo = findOne(todoId);
        if (todo.isFinished()) {
            ResultMessages messages = ResultMessages.error();
            messages.add(ResultMessage
                    .fromText("[E002] The requested Todo is already finished. (id="
                            + todoId + ")"));
            throw new BusinessException(messages);
        }
        todo.setFinished(true);
        todoRepository.update(todo);
        /* REMOVE THIS LINE IF YOU USE JPA
            todoRepository.save(todo);
           REMOVE THIS LINE IF YOU USE JPA */
        return todo;
    }

    @Override
    public void delete(String todoId) {
        Todo todo = findOne(todoId);
        todoRepository.delete(todo);
    }
}
```

Implement the processing of retrieving single Todo Resource API(GET Todo) into `getTodo` method of `TodoRestController`.

`src/main/java/todo/api/todo/TodoRestController.java`

```
package todo.api.todo;

import java.util.ArrayList;
import java.util.Collection;
import java.util.List;

import javax.inject.Inject;

import org.dozer.Mapper;
import org.springframework.http.HttpStatus;
import org.springframework.validation.annotation.Validated;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.ResponseStatus;
import org.springframework.web.bind.annotation.RestController;

import todo.domain.model.Todo;
import todo.domain.service.todo.TodoService;

@RestController
@RequestMapping("todos")
public class TodoRestController {

    @Inject
    TodoService todoService;
    @Inject
    Mapper beanMapper;

    @RequestMapping(method = RequestMethod.GET)
    @ResponseStatus(HttpStatus.OK)
    public List<TodoResource> getTodos() {
        Collection<Todo> todos = todoService.findAll();
        List<TodoResource> todoResources = new ArrayList<>();
        for (Todo todo : todos) {
            todoResources.add(beanMapper.map(todo, TodoResource.class));
        }
        return todoResources;
    }
```

```
    @RequestMapping(method = RequestMethod.POST)
    @ResponseStatus(HttpStatus.CREATED)
    public TodoResource postTodos(@RequestBody @Validated TodoResource␣
↪todoResource) {
        Todo createdTodo = todoService.create(beanMapper.map(todoResource, Todo.
↪class));
        TodoResource createdTodoResponse = beanMapper.map(createdTodo,␣
↪TodoResource.class);
        return createdTodoResponse;
    }

    @RequestMapping(value="{todoId}", method = RequestMethod.GET) // (1)
    @ResponseStatus(HttpStatus.OK)
    public TodoResource getTodo(@PathVariable("todoId") String todoId) { // (2)
        Todo todo = todoService.findOne(todoId); // (3)
        TodoResource todoResource = beanMapper.map(todo, TodoResource.class);
        return todoResource;
    }

}
```

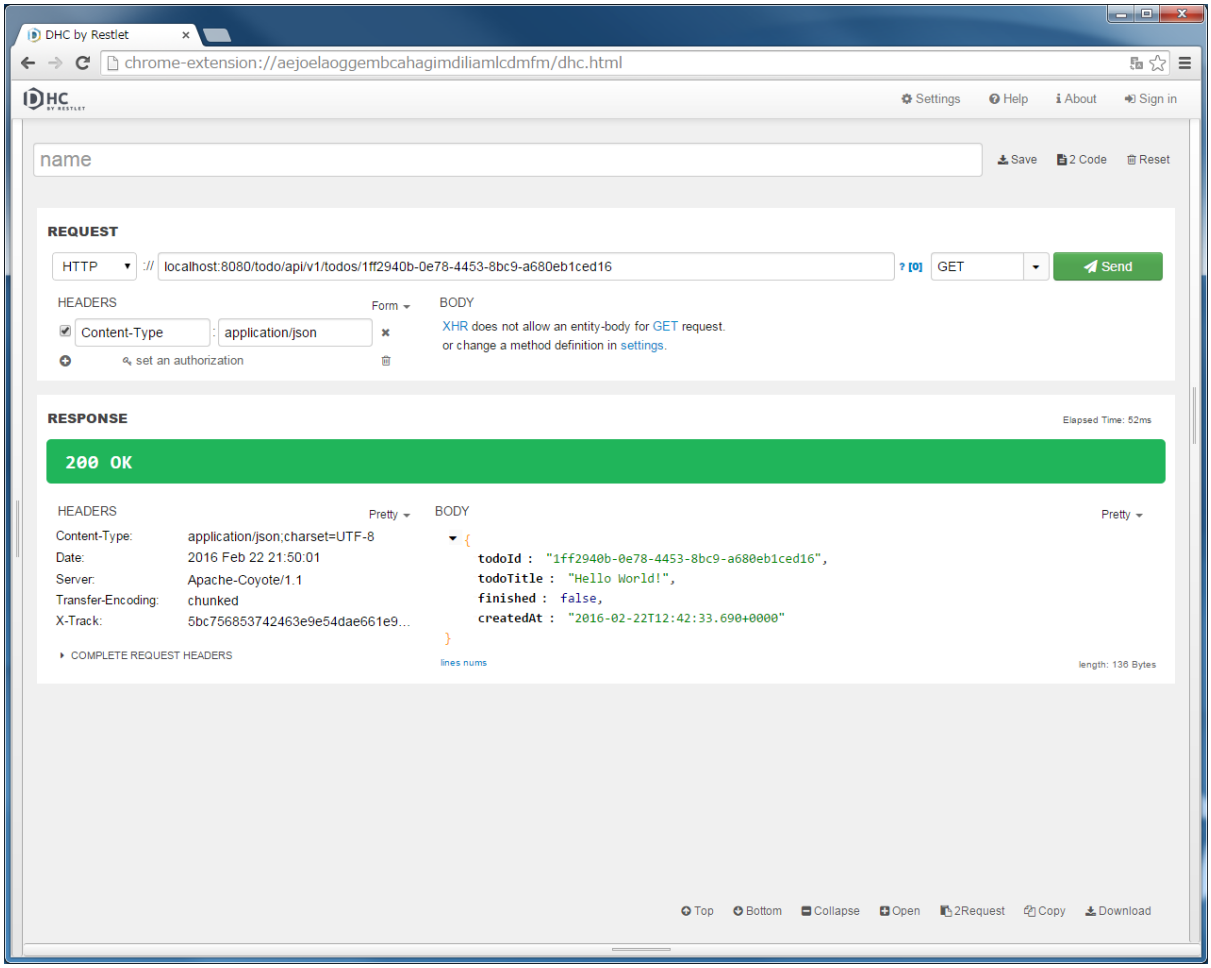
Sr. No	Description
(1)	In order to get the `todoId` from path, specify the path variable in the `value` attribute of the `@RequestMapping` annotation. Set the `RequestMethod.GET` to `method` attribute for handling the GET request.
(2)	Specify the path variable name to retrieve `todoId` in the `value` attribute of the `@PathVariable` annotation.
(3)	You can use the `todoId` obtained from path variable to get one Todo resource.

Check the operation of the implemented API using DHC.

Open the DHC, enter "`localhost:8080/todo/api/v1/todos/{todoId}`" in the URL and specify GET in method.

Since it is necessary to enter the actual ID at `{todoId}`, run the POST Todos or GET Todos to get actual ID, copy & paste the `todoId` from the Response, and click the "Send" button.

HTTP status returned “200 OK” and JSON of the indicated Todo resource displays in [Body] of [RESPONSE] part.



Implementation of PUT Todo

Implement the processing of API(PUT Todo) into `putTodo` method of `TodoRestController` that updates(updating into completed status) one record of Todo Resource.

`src/main/java/todo/api/todo/TodoRestController.java`

```
package todo.api.todo;

import java.util.ArrayList;
import java.util.Collection;
import java.util.List;
```

```
import javax.inject.Inject;

import org.dozer.Mapper;
import org.springframework.http.HttpStatus;
import org.springframework.validation.annotation.Validated;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.ResponseStatus;
import org.springframework.web.bind.annotation.RestController;

import todo.domain.model.Todo;
import todo.domain.service.todo.TodoService;

@RestController
@RequestMapping("todos")
public class TodoRestController {

    @Inject
    TodoService todoService;
    @Inject
    Mapper beanMapper;

    @RequestMapping(method = RequestMethod.GET)
    @ResponseStatus(HttpStatus.OK)
    public List<TodoResource> getTodos() {
        Collection<Todo> todos = todoService.findAll();
        List<TodoResource> todoResources = new ArrayList<>();
        for (Todo todo : todos) {
            todoResources.add(beanMapper.map(todo, TodoResource.class));
        }
        return todoResources;
    }

    @RequestMapping(method = RequestMethod.POST)
    @ResponseStatus(HttpStatus.CREATED)
    public TodoResource postTodos(@RequestBody @Validated TodoResource␣
↪todoResource) {
        Todo createdTodo = todoService.create(beanMapper.map(todoResource, Todo.
↪class));
        TodoResource createdTodoResponse = beanMapper.map(createdTodo,␣
↪TodoResource.class);
        return createdTodoResponse;
    }

    @RequestMapping(value="{todoId}", method = RequestMethod.GET)
    @ResponseStatus(HttpStatus.OK)
    public TodoResource getTodo(@PathVariable("todoId") String todoId) {
```

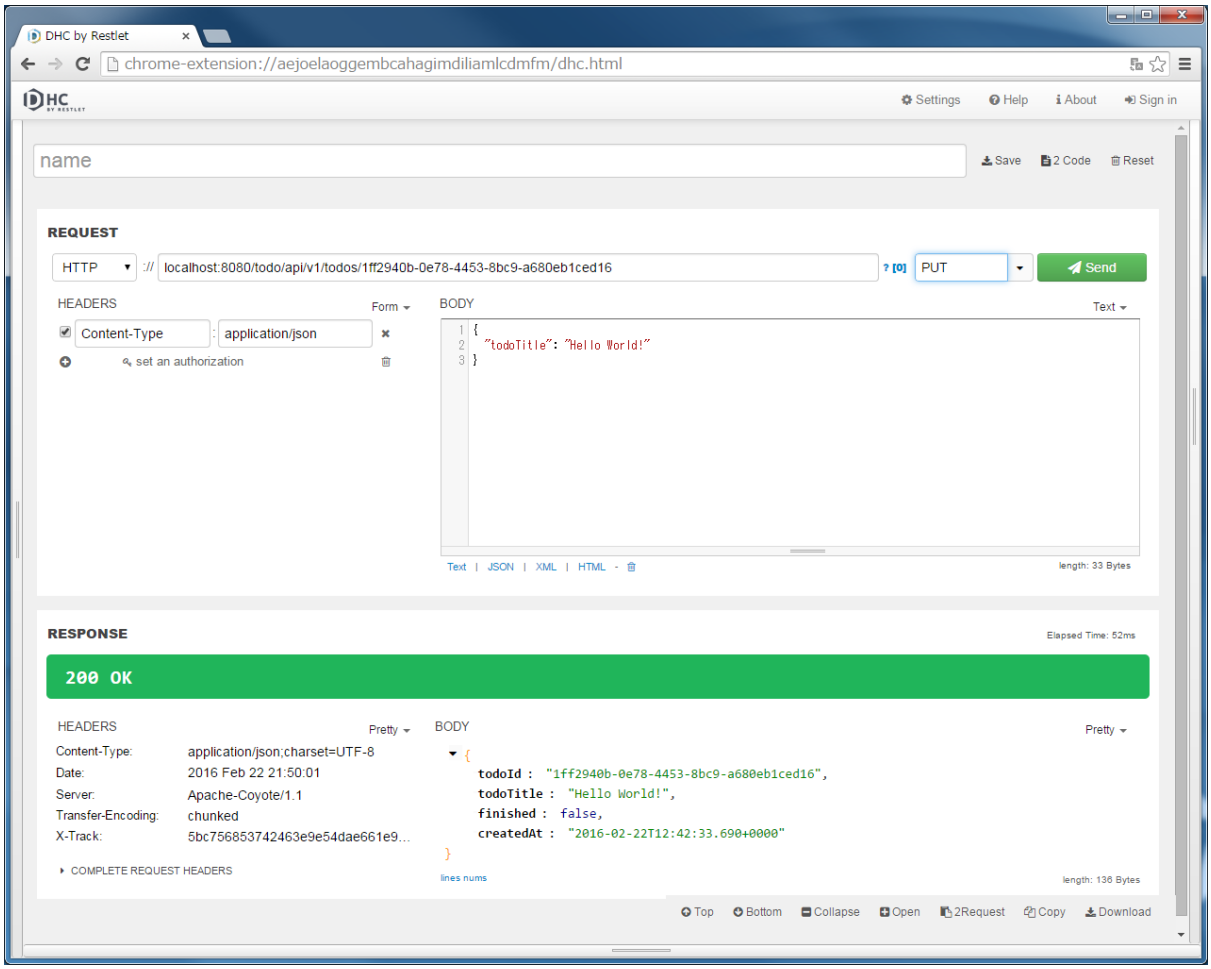
```
        Todo todo = todoService.findOne(todoId);
        TodoResource todoResource = beanMapper.map(todo, TodoResource.class);
        return todoResource;
    }

    @RequestMapping(value="{todoId}", method = RequestMethod.PUT) // (1)
    @ResponseStatus(HttpStatus.OK)
    public TodoResource putTodo(@PathVariable("todoId") String todoId) { // (2)
        Todo finishedTodo = todoService.finish(todoId); // (3)
        TodoResource finishedTodoResource = beanMapper.map(finishedTodo,␣
↪TodoResource.class);
        return finishedTodoResource;
    }

}
```

Sr. No	Description
(1)	In order to get the `todoId` from path, specify the path variable in the `value` attribute of the `@RequestMapping`annotation. Set the `RequestMethod.PUT` to `method` attribute for handling the PUT request.
(2)	Specify the path variable name to retrieve `todoId` in the `value` attribute of the `@PathVariable` annotation.
(3)	You can use the `todoId` obtained from path variable to update the Todo resource in completed status.

Check the operation of the implemented API using DHC.

Open the DHC, enter "`localhost:8080/todo/api/v1/todos/{todoId}`" in the URL and specify PUT in method.

Since it is necessary to enter the actual ID at `{todoId}`, run the POST Todos or GET Todos to get actual ID, copy & paste the `todoId` from the Response, and click the "Send" button.

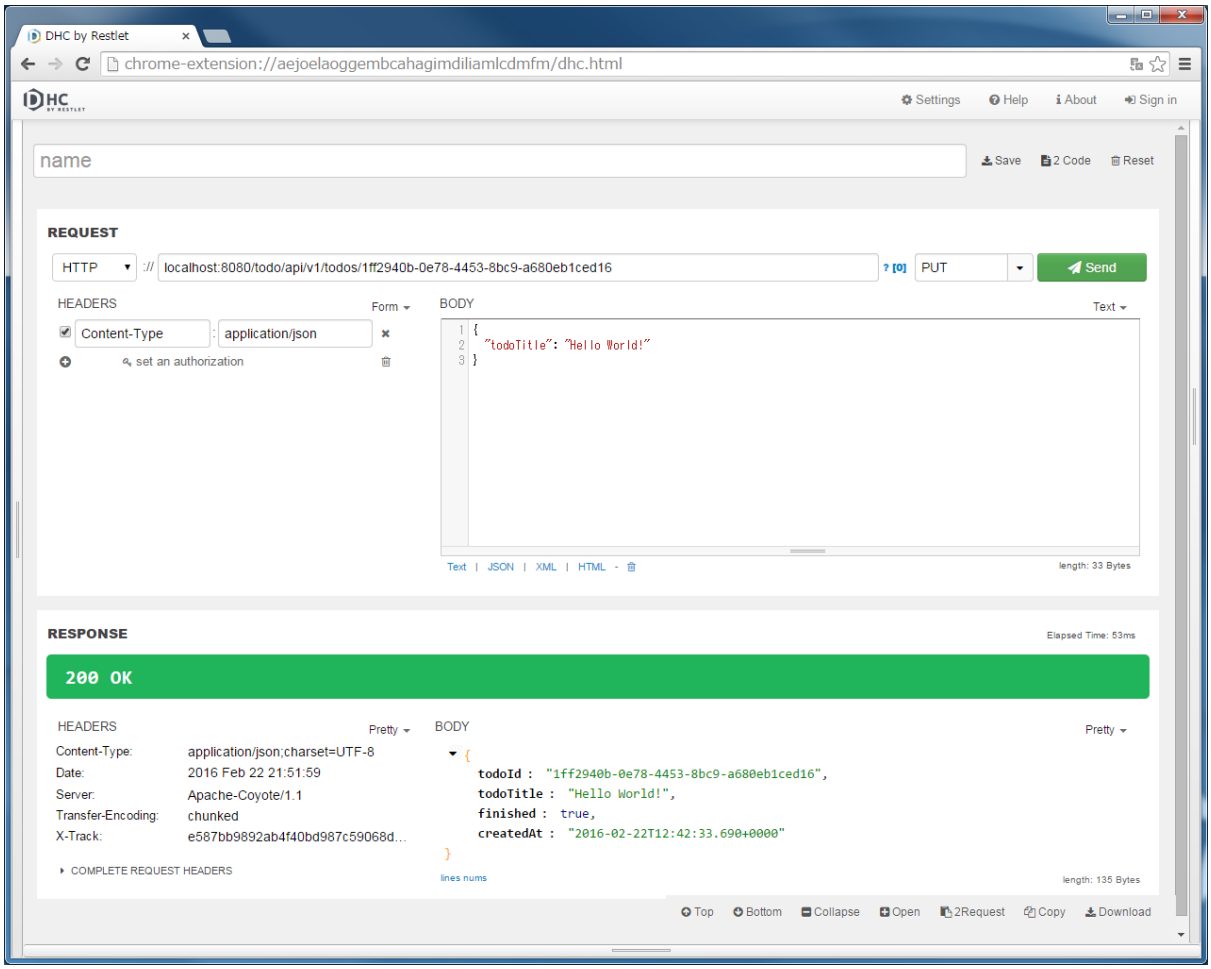


HTTP status returned “200 OK” and JSON of the modified Todo resource displays in [Body] of [RESPONSE] part.

`finished` is updated to `true`.

Implementation of DELETE Todo

Lastly, implement the processing of API(DELETE Todo) into `deleteTodo` method of `TodoRestController` that delete one record of Todo Resource.



src/main/java/todo/api/todo/TodoRestController.java

```
package todo.api.todo;

import java.util.ArrayList;
import java.util.Collection;
import java.util.List;

import javax.inject.Inject;

import org.dozer.Mapper;
import org.springframework.http.HttpStatus;
import org.springframework.validation.annotation.Validated;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.ResponseStatus;
import org.springframework.web.bind.annotation.RestController;

import todo.domain.model.Todo;
```

```
import todo.domain.service.todo.TodoService;

@RestController
@RequestMapping("todos")
public class TodoRestController {

    @Inject
    TodoService todoService;
    @Inject
    Mapper beanMapper;

    @RequestMapping(method = RequestMethod.GET)
    @ResponseStatus(HttpStatus.OK)
    public List<TodoResource> getTodos() {
        Collection<Todo> todos = todoService.findAll();
        List<TodoResource> todoResources = new ArrayList<>();
        for (Todo todo : todos) {
            todoResources.add(beanMapper.map(todo, TodoResource.class));
        }
        return todoResources;
    }

    @RequestMapping(method = RequestMethod.POST)
    @ResponseStatus(HttpStatus.CREATED)
    public TodoResource postTodos(@RequestBody @Validated TodoResource␣
↪todoResource) {
        Todo createdTodo = todoService.create(beanMapper.map(todoResource, Todo.
↪class));
        TodoResource createdTodoResponse = beanMapper.map(createdTodo,␣
↪TodoResource.class);
        return createdTodoResponse;
    }

    @RequestMapping(value="{todoId}", method = RequestMethod.GET)
    @ResponseStatus(HttpStatus.OK)
    public TodoResource getTodo(@PathVariable("todoId") String todoId) {
        Todo todo = todoService.findOne(todoId);
        TodoResource todoResource = beanMapper.map(todo, TodoResource.class);
        return todoResource;
    }

    @RequestMapping(value="{todoId}", method = RequestMethod.PUT)
    @ResponseStatus(HttpStatus.OK)
    public TodoResource putTodo(@PathVariable("todoId") String todoId) {
        Todo finishedTodo = todoService.finish(todoId);
        TodoResource finishedTodoResource = beanMapper.map(finishedTodo,␣
↪TodoResource.class);
        return finishedTodoResource;
    }
```

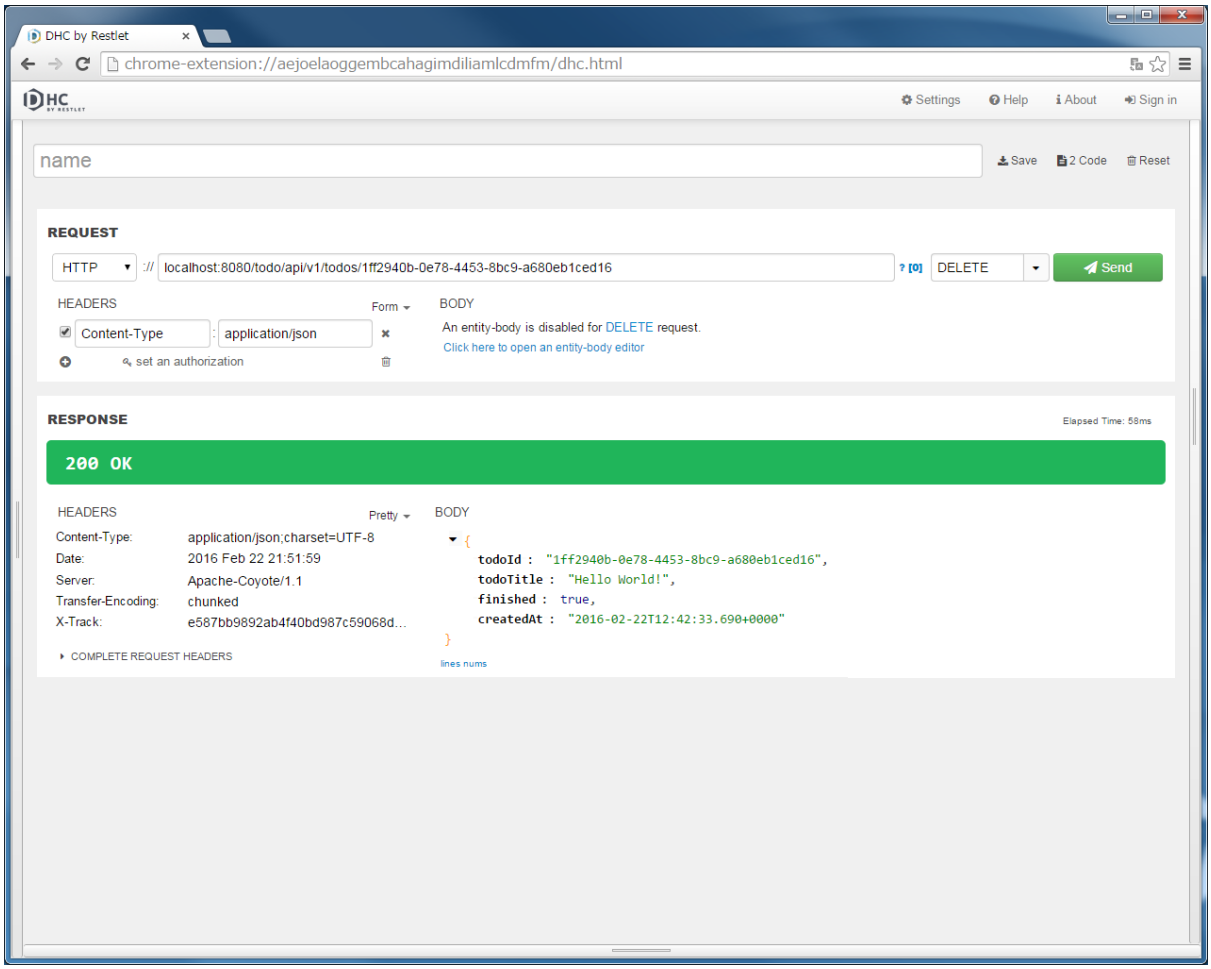
```
    @RequestMapping(value="{todoId}", method = RequestMethod.DELETE) // (1)
    @ResponseStatus(HttpStatus.NO_CONTENT) // (2)
    public void deleteTodo(@PathVariable("todoId") String todoId) { // (3)
        todoService.delete(todoId); // (4)
    }

}
```

Sr. No	Description
(1)	In order to get the `todoId` from path, specify the path variable in the `value` attribute of the `@RequestMapping` annotation. Set the `RequestMethod.DELETE` to `method` attribute for handling the DELETE request.
(2)	Specify `@ResponseStatus` annotation to the HTTP status code for response. To set "204 No Content" as a HTTP status, set the `HttpStatus.NO_CONTENT` to the `value` attribute.
(3)	The type of return value is a `void` because there is no content to be returned in the case of DELETE.
(4)	You can use the `todoId` obtained from path variable to delete the Todo resource.

Check the operation of the implemented API using DHC.

Open the DHC, enter "`localhost:8080/todo/api/v1/todos/{todoId}`" in the URL and specify DELETE in method.

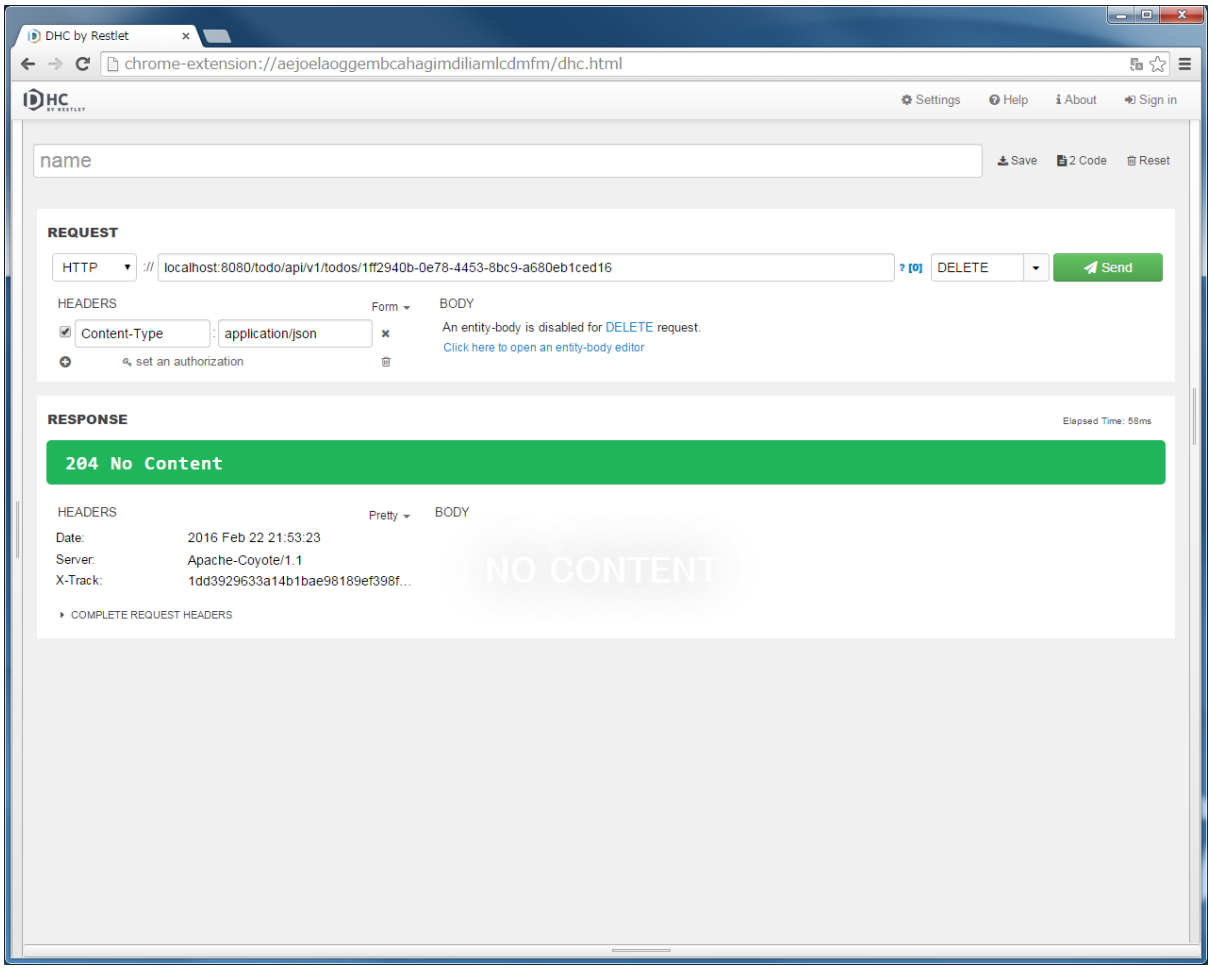
Since it is necessary to enter the actual ID at `{todoId}`, run the POST Todos or GET Todos to get actual ID, copy & paste the `todoId` from the Response, and click the "Send" button.



HTTP status returned “204 No Content” and [Body] of [RESPONSE] is empty.

Open the DHC, enter "`localhost:8080/todo/api/v1/todos`" in the URL and click the “Send” button by specify GET in method.

you can confirm that the Todo resource has been removed.



Implementation of exception handling

In this tutorial, for easy understanding, the implementation of exception handling made a simpler than that are recommended in this guideline.

It is strongly recommended that the actual exception handling should be handled in a way described in the *RESTful Web Service*.

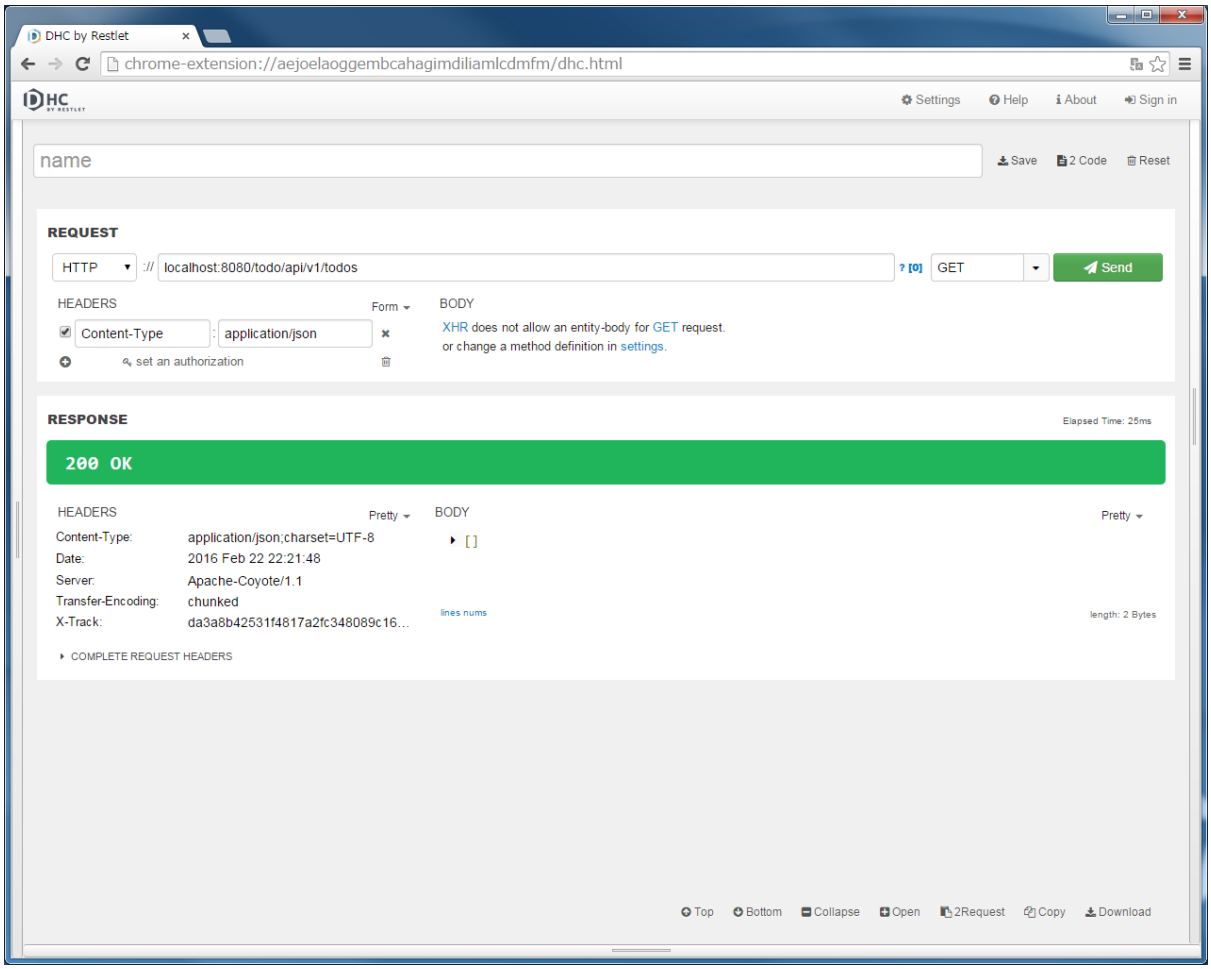
Change Domain layer implementation

In this tutorial, the error messages are retrieved from the property file based on error code.

Therefore, modify the implementation of the Service class as follows which is created at *Tutorial (Todo Application)* before implementing the exception handling.

Specify the error code instead of hard-coded error message.

`src/main/java/todo/domain/service/todo/TodoServiceImpl.java`




```
package todo.domain.service.todo;

import java.util.Collection;
import java.util.Date;
import java.util.UUID;

import javax.inject.Inject;

import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import org.terasoluna.gfw.common.exception.BusinessException;
import org.terasoluna.gfw.common.exception.ResourceNotFoundException;
import org.terasoluna.gfw.common.message.ResultMessages;

import todo.domain.model.Todo;
import todo.domain.repository.todo.TodoRepository;

@Service
@Transactional
public class TodoServiceImpl implements TodoService {
```

```
    private static final long MAX_UNFINISHED_COUNT = 5;

    @Inject
    TodoRepository todoRepository;

    @Override
    @Transactional(readOnly = true)
    public Todo findOne(String todoId) {
        Todo todo = todoRepository.findOne(todoId);
        if (todo == null) {
            ResultMessages messages = ResultMessages.error();
            messages.add("E404", todoId);
            throw new ResourceNotFoundException(messages);
        }
        return todo;
    }

    @Override
    @Transactional(readOnly = true)
    public Collection<Todo> findAll() {
        return todoRepository.findAll();
    }

    @Override
    public Todo create(Todo todo) {
        long unfinishedCount = todoRepository.countByFinished(false);
        if (unfinishedCount >= MAX_UNFINISHED_COUNT) {
            ResultMessages messages = ResultMessages.error();
            messages.add("E001", MAX_UNFINISHED_COUNT);
            throw new BusinessException(messages);
        }

        String todoId = UUID.randomUUID().toString();
        Date createdAt = new Date();

        todo.setTodoId(todoId);
        todo.setCreatedAt(createdAt);
        todo.setFinished(false);

        todoRepository.create(todo);
        /* REMOVE THIS LINE IF YOU USE JPA
            todoRepository.save(todo);
           REMOVE THIS LINE IF YOU USE JPA */

        return todo;
    }

    @Override
    public Todo finish(String todoId) {
        Todo todo = findOne(todoId);
```

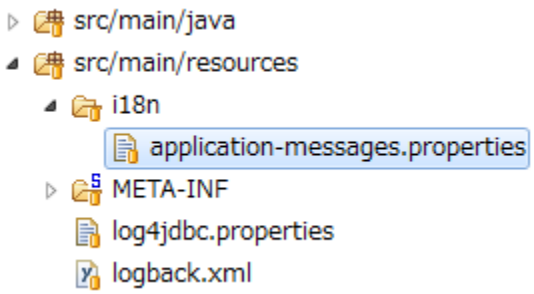
```
        if (todo.isFinished()) {
            ResultMessages messages = ResultMessages.error();
            messages.add("E002", todoId);
            throw new BusinessException(messages);
        }
        todo.setFinished(true);
        todoRepository.update(todo);
        /* REMOVE THIS LINE IF YOU USE JPA
            todoRepository.save(todo);
           REMOVE THIS LINE IF YOU USE JPA */
        return todo;
    }

    @Override
    public void delete(String todoId) {
        Todo todo = findOne(todoId);
        todoRepository.delete(todo);
    }
}
```

Error message definition

In this tutorial, the error messages are retrieved from the property file based on error code.

Therefore, define the error code corresponding to the error messages in the message property file before implementing the exception handling.

Define the error code corresponding to the error messages of the processing result in the message property file.



`src/main/resources/i18n/application-messages.properties`

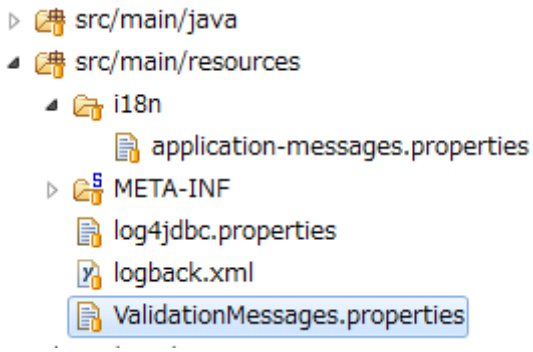
```
e.xx.fw.5001 = Resource not found.

e.xx.fw.7001 = Illegal screen flow detected!
e.xx.fw.7002 = CSRF attack detected!
e.xx.fw.7003 = Access Denied detected!
e.xx.fw.7004 = Missing CSRF detected!

e.xx.fw.8001 = Business error occurred!

e.xx.fw.9001 = System error occurred!
e.xx.fw.9002 = Data Access error!

# typemismatch
typeMismatch="{0}" is invalid.
typeMismatch.int="{0}" must be an integer.
typeMismatch.double="{0}" must be a double.
typeMismatch.float="{0}" must be a float.
typeMismatch.long="{0}" must be a long.
typeMismatch.short="{0}" must be a short.
typeMismatch.boolean="{0}" must be a boolean.
typeMismatch.java.lang.Integer="{0}" must be an integer.
typeMismatch.java.lang.Double="{0}" must be a double.
typeMismatch.java.lang.Float="{0}" must be a float.
typeMismatch.java.lang.Long="{0}" must be a long.
typeMismatch.java.lang.Short="{0}" must be a short.
typeMismatch.java.lang.Boolean="{0}" is not a boolean.
typeMismatch.java.util.Date="{0}" is not a date.
typeMismatch.java.lang.Enum="{0}" is not a valid value.

# For this tutorial
E001 = [E001] The count of un-finished Todo must not be over {0}.
E002 = [E002] The requested Todo is already finished. (id={0})
E400 = [E400] The requested Todo contains invalid values.
E404 = [E404] The requested Todo is not found. (id={0})
E500 = [E500] System error occurred.
E999 = [E999] Error occurred. Caused by : {0}
```

Define the error messages corresponding to input check error codes, in Bean Validation message properties file.

Change the default message definition because the default message does not include the item name in the message

In this tutorial, only define the message corresponding to the rules (`@NotNull` and `@Size`) that are used in

`TodoResource` class.



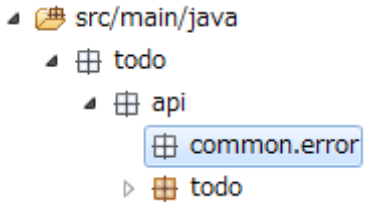
`src/main/resources/ValidationMessages.properties`

```
javax.validation.constraints.NotNull.message = {0} may not be null.
javax.validation.constraints.Size.message    = {0} size must be between {min} and
↪{max}.
```

Create a package that contains the error handling class

Create a package for storing the error handling classes.

In this tutorial, creating a package for storing the `todo.api.common.error` error handling class.

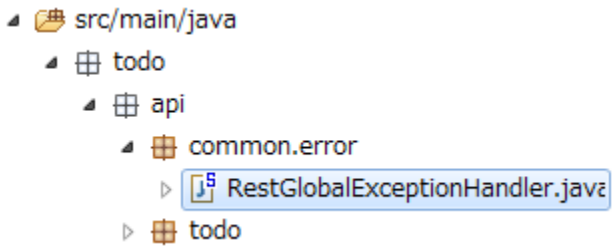


Creating REST API error handling class

The REST API error handling class is created by inheriting the `org.springframework.web.servlet.mvc.method.annotation.ResponseEntityExceptionHandler`

provided by Spring MVC, adding the `@ControllerAdvice` annotation, and recommended to add `(annotations = RestController.class)` attribute in order to restrict to the REST API processing.

Below created the `todo.api.common.error.RestGlobalExceptionHandler` class inherited from the `ResponseEntityExceptionHandler`.



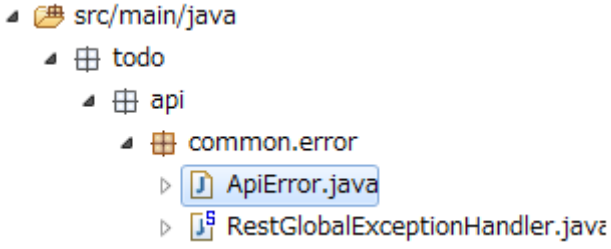
`src/main/java/todo/api/common/error/RestGlobalExceptionHandler.java`

```
package todo.api.common.error;

import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.servlet.mvc.method.annotation.
↪ResponseEntityExceptionHandler;

@ControllerAdvice
public class RestGlobalExceptionHandler extends ResponseEntityExceptionHandler {

}
```

Creating JavaBean for holding the REST API error information

Create `ApiError` class under the `todo.api.common.error` package for holding the error information generated by the REST API.

`ApiError` class converted into JSON and return to client.



src/main/java/todo/api/common/error/ApiError.java

```
package todo.api.common.error;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;

import com.fasterxml.jackson.annotation.JsonInclude;

public class ApiError implements Serializable {

    private static final long serialVersionUID = 1L;

    private final String code;

    private final String message;

    @JsonInclude(JsonInclude.Include.NON_EMPTY)
    private final String target;

    @JsonInclude(JsonInclude.Include.NON_EMPTY)
    private final List<ApiError> details = new ArrayList<>();

    public ApiError(String code, String message) {
        this(code, message, null);
    }

    public ApiError(String code, String message, String target) {
        this.code = code;
        this.message = message;
        this.target = target;
    }

    public String getCode() {
        return code;
    }

    public String getMessage() {
        return message;
    }

    public String getTarget() {
        return target;
    }

    public List<ApiError> getDetails() {
        return details;
    }

    public void addDetail(ApiError detail) {
```

```
            details.add(detail);
        }

}
```

Implementation of putting error information to the HTTP response BODY

By default `ResponseEntityExceptionHandler` configures only HTTP status (400 or 500 etc) but not configures the HTTP response BODY. Therefore, output the BODY by overidding the `handleExceptionInternal` method as follows.

`src/main/java/todo/api/common/error/RestGlobalExceptionHandler.java`

```
package todo.api.common.error;

import javax.inject.Inject;

import org.springframework.context.MessageSource;
import org.springframework.http.HttpHeaders;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.context.request.WebRequest;
import org.springframework.web.servlet.mvc.method.annotation.
↪ResponseEntityExceptionHandler;

@ControllerAdvice
public class RestGlobalExceptionHandler extends ResponseEntityExceptionHandler {

    @Inject
    MessageSource messageSource;

    @Override
    protected ResponseEntity<Object> handleExceptionInternal(Exception ex,
            Object body, HttpHeaders headers, HttpStatus status,
            WebRequest request) {
        Object responseBody = body;
        if (body == null) {
            responseBody = createApiError(request, "E999", ex.getMessage());
        }
        return ResponseEntity.status(status).headers(headers).body(responseBody);
    }
```

```
    private ApiError createApiError(WebRequest request, String errorCode,
            Object... args) {
        return new ApiError(errorCode, messageSource.getMessage(errorCode,
                args, request.getLocale()));
    }

}
```

By performing the above implementation, the error information is logged in to HTTP response BODY which was handled by the `ResponseEntityExceptionHandler` .

About the exception handled by `ResponseEntityExceptionHandler` , refer *HTTP response code set by DefaultHandlerExceptionResolver*.

Check the operation of the implemented error handling using DHC.

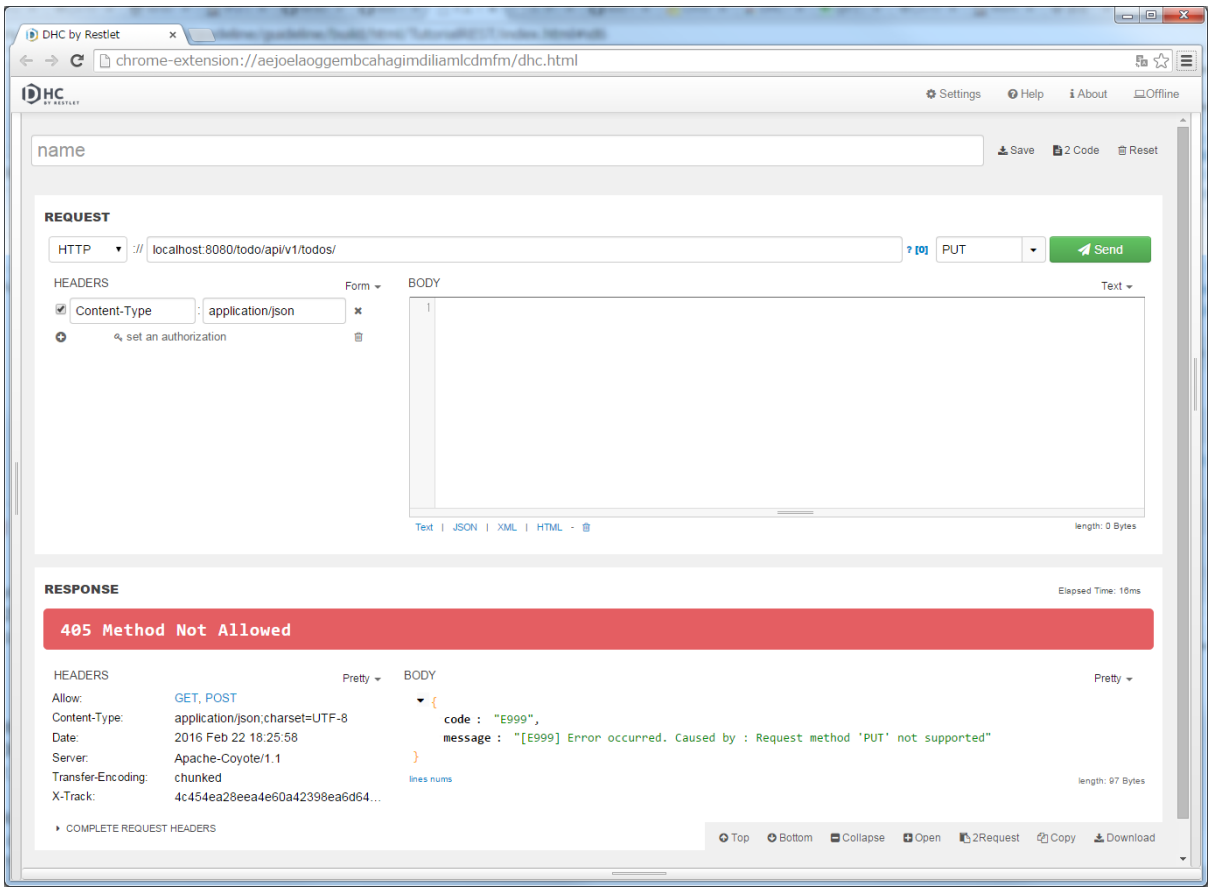
Open the DHC, enter `"localhost:8080/todo/api/v1/todos"` in the URL and click the “Send” button after specifying PUT in method.

HTTP status returned “405 Method Not Allowed” and the JSON error information displays in [Body] of [RESPONSE] part.

Error handling of Input errors

Type of input errors as follows.

- `org.springframework.web.bind.MethodArgumentNotValidException`
- `org.springframework.validation.BindException`
- `org.springframework.http.converter.HttpMessageNotReadableException`
- `org.springframework.beans.TypeMismatchException`



In this tutorial, implementing the `MethodArgumentNotValidException` error handling.

The `MethodArgumentNotValidException` is an exception that occurs if there is any input error in the data stored in HTTP request BODY.

`src/main/java/todo/api/common/error/RestGlobalExceptionHandler.java`

```
package todo.api.common.error;

import javax.inject.Inject;

import org.springframework.context.MessageSource;
import org.springframework.context.support.DefaultMessageSourceResolvable;
import org.springframework.http.HttpHeaders;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.validation.FieldError;
import org.springframework.validation.ObjectError;
import org.springframework.web.bind.MethodArgumentNotValidException;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.context.request.WebRequest;
import org.springframework.web.servlet.mvc.method.annotation.
 ↪ResponseEntityExceptionHandler;
```

```
@ControllerAdvice
public class RestGlobalExceptionHandler extends ResponseEntityExceptionHandler {

    @Inject
    MessageSource messageSource;

    @Override
    protected ResponseEntity<Object> handleExceptionInternal(Exception ex,
            Object body, HttpHeaders headers, HttpStatus status,
            WebRequest request) {
        Object responseBody = body;
        if (body == null) {
            responseBody = createApiError(request, "E999", ex.getMessage());
        }
        return ResponseEntity.status(status).headers(headers).body(responseBody);
    }

    private ApiError createApiError(WebRequest request, String errorCode,
            Object... args) {
        return new ApiError(errorCode, messageSource.getMessage(errorCode,
                args, request.getLocale()));
    }

    @Override
    protected ResponseEntity<Object> handleMethodArgumentNotValid(
            MethodArgumentNotValidException ex, HttpHeaders headers,
            HttpStatus status, WebRequest request) {
        ApiError apiError = createApiError(request, "E400");
        for (FieldError fieldError : ex.getBindingResult().getFieldErrors()) {
            apiError.addDetail(createApiError(request, fieldError, fieldError
                    .getField()));
        }
        for (ObjectError objectError : ex.getBindingResult().getGlobalErrors()) {
            apiError.addDetail(createApiError(request, objectError, objectError
                    .getObjectName()));
        }
        return handleExceptionInternal(ex, apiError, headers, status, request);
    }

    private ApiError createApiError(WebRequest request,
            DefaultMessageSourceResolvable messageSourceResolvable,
            String target) {
        return new ApiError(messageSourceResolvable.getCode(), messageSource
                .getMessage(messageSourceResolvable, request.getLocale()), target);
    }

}
```

Check the operation of the implemented error handling using DHC.

Open the DHC, enter "`localhost:8080/todo/api/v1/todos`" in the URL and specify POST in method.

Enter below JSON in [BODY] of the [REQUEST].

```
{
  "todoTitle": null
}
```

Furthermore, Add HTTP header by [+] button of [REQUEST] [HEADERS] and click "Send" button after setting [`application/json`] in the [`Content-Type`].

HTTP status returned "400 Bad Request" and JSON error information displays in [Body] of [RESPONSE] part.

Since `todoTitle` is required field, required error occurred.

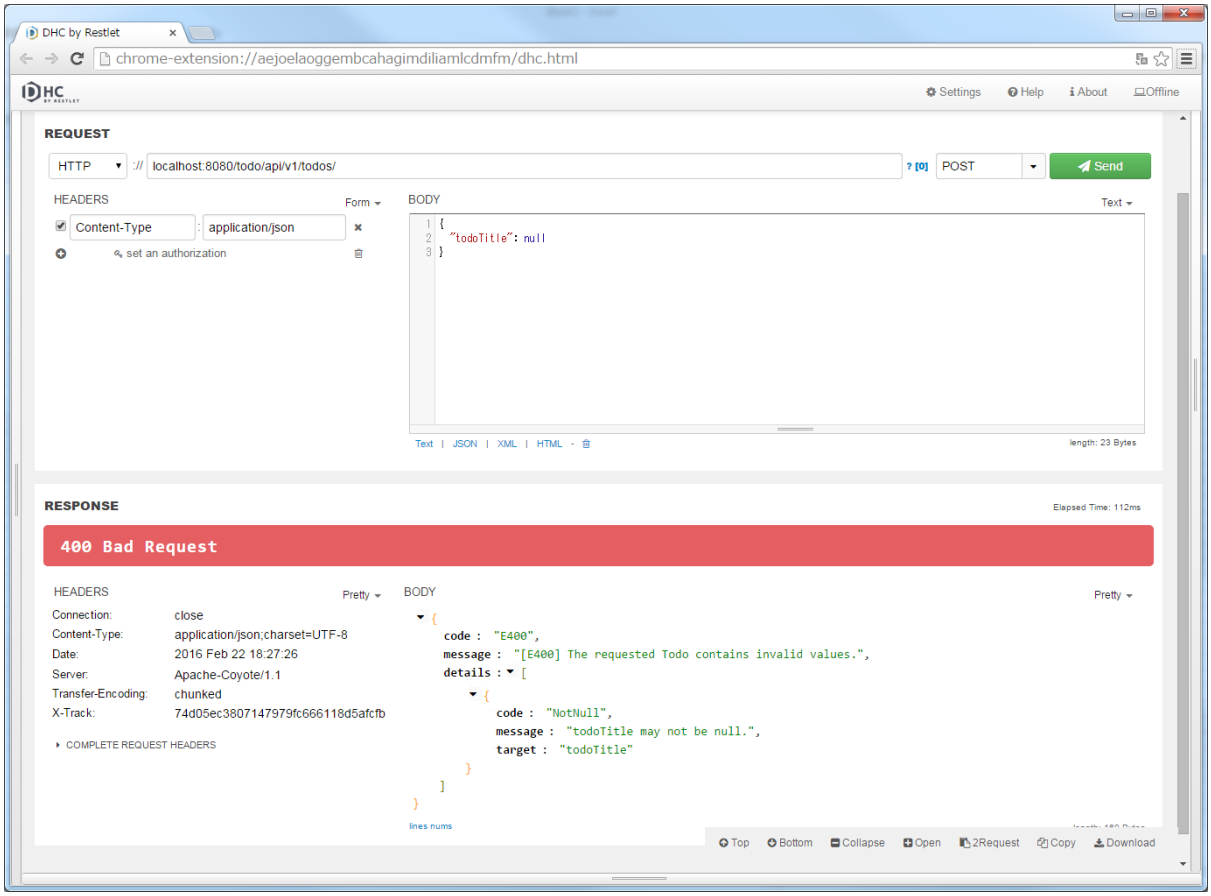
Business exception error handling

Handling a business exception by adding `org.terasoluna.gfw.common.exception.BusinessException` method in the `RestGlobalExceptionHandler`.

Set "409 Conflict" in HTTP status if business exception occurred.

`src/main/java/todo/api/common/error/RestGlobalExceptionHandler.java`

```
package todo.api.common.error;

import javax.inject.Inject;

import org.springframework.context.MessageSource;
import org.springframework.context.support.DefaultMessageSourceResolvable;
import org.springframework.http.HttpHeaders;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.validation.FieldError;
```




```
import org.springframework.validation.ObjectError;
import org.springframework.web.bind.MethodArgumentNotValidException;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.context.request.WebRequest;
import org.springframework.web.servlet.mvc.method.annotation.
 ↪ResponseEntityExceptionHandler;
import org.terasoluna.gfw.common.exception.BusinessException;
import org.terasoluna.gfw.common.exception.ResultMessagesNotificationException;
import org.terasoluna.gfw.common.message.ResultMessage;

@ControllerAdvice
public class RestGlobalExceptionHandler extends ResponseEntityExceptionHandler {

    @Inject
    MessageSource messageSource;

    @Override
    protected ResponseEntity<Object> handleExceptionInternal(Exception ex,
            Object body, HttpHeaders headers, HttpStatus status,
            WebRequest request) {
        Object responseBody = body;
        if (body == null) {
```

```
            responseBody = createApiError(request, "E999", ex.getMessage());
        }
        return ResponseEntity.status(status).headers(headers).body(responseBody);
    }

    private ApiError createApiError(WebRequest request, String errorCode,
            Object... args) {
        return new ApiError(errorCode, messageSource.getMessage(errorCode,
                args, request.getLocale()));
    }

    @Override
    protected ResponseEntity<Object> handleMethodArgumentNotValid(
            MethodArgumentNotValidException ex, HttpHeaders headers,
            HttpStatus status, WebRequest request) {
        ApiError apiError = createApiError(request, "E400");
        for (FieldError fieldError : ex.getBindingResult().getFieldErrors()) {
            apiError.addDetail(createApiError(request, fieldError, fieldError
                    .getField()));
        }
        for (ObjectError objectError : ex.getBindingResult().getGlobalErrors()) {
            apiError.addDetail(createApiError(request, objectError, objectError
                    .getObjectName()));
        }
        return handleExceptionInternal(ex, apiError, headers, status, request);
    }

    private ApiError createApiError(WebRequest request,
            DefaultMessageSourceResolvable messageSourceResolvable,
            String target) {
        return new ApiError(messageSourceResolvable.getCode(), messageSource
                .getMessage(messageSourceResolvable, request.getLocale()), target);
    }

    @ExceptionHandler(BusinessException.class)
    public ResponseEntity<Object> handleBusinessException(BusinessException ex,
            WebRequest request) {
        return handleResultMessagesNotificationException(ex, new HttpHeaders(),
                HttpStatus.CONFLICT, request);
    }

    private ResponseEntity<Object> handleResultMessagesNotificationException(
            ResultMessagesNotificationException ex, HttpHeaders headers,
            HttpStatus status, WebRequest request) {
        ResultMessage message = ex.getResultMessages().iterator().next();
        ApiError apiError = createApiError(request, message.getCode(), message
                .getArgs());
        return handleExceptionInternal(ex, apiError, headers, status, request);
    }
```

```
}
```

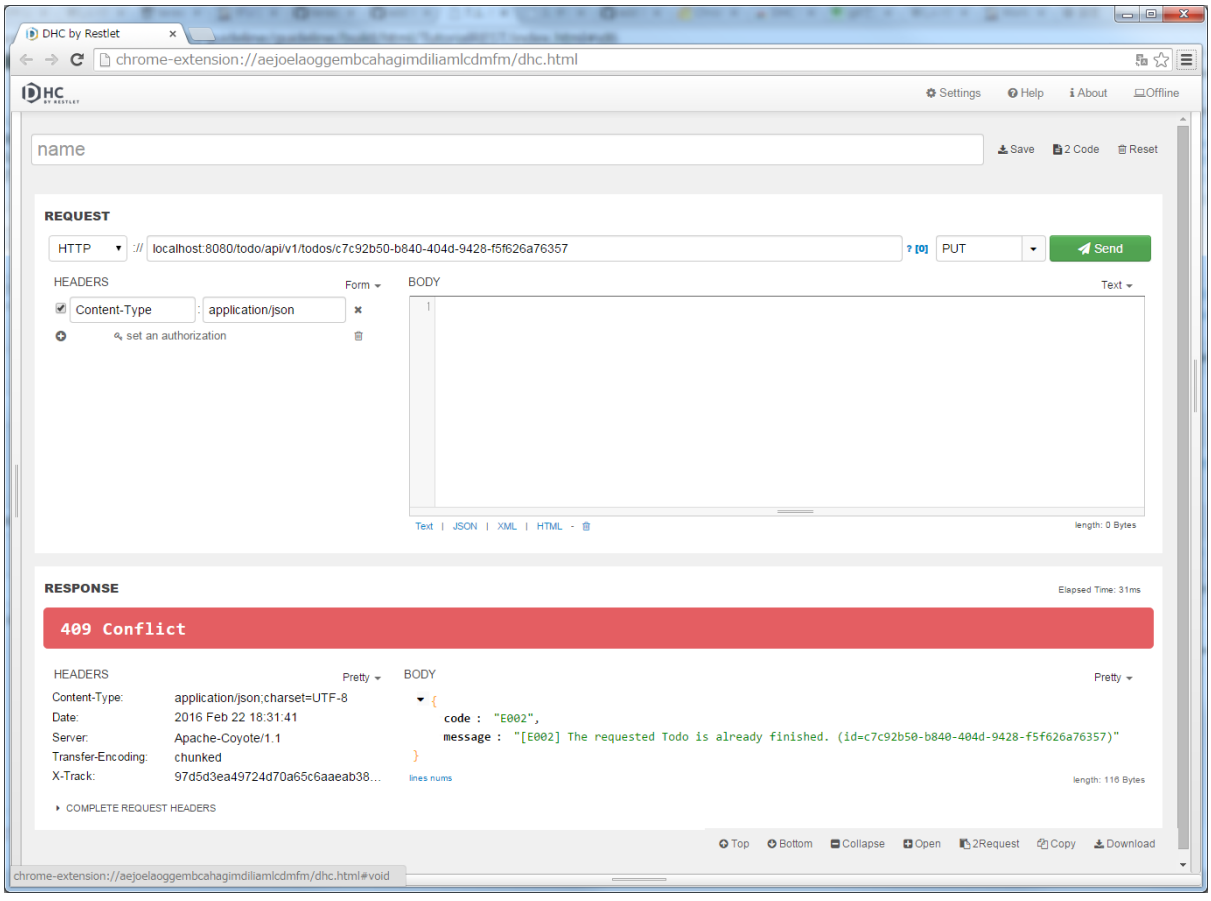
Check the operation of the implemented error handling using DHC.

Open the DHC, enter "`localhost:8080/todo/api/v1/todos/{todoId}`" in the URL and specify PUT in method.

Since it is necessary to enter the actual ID at `{todoId}`, run the POST Todos or GET Todos to get actual ID, copy & paste the `todoId` from the Response, and click the "Send" button twice.

Specify un-completed `todoId` of the Todos.

HTTP status returned "409 Conflict" as a response of the 2nd request and JSON error information displays in [Body] of [RESPONSE] part.



Resource not found exception error handling

Resource not found exception handles by adding `org.terasoluna.gfw.common.exception.ResourceNotFoundExce` method in the `RestGlobalExceptionHandler`.

Set "404 NotFound" in HTTP status if Resource not found exception occurred.

`src/main/java/todo/api/common/error/RestGlobalExceptionHandler.java`

```
package todo.api.common.error;

import javax.inject.Inject;

import org.springframework.context.MessageSource;
import org.springframework.context.support.DefaultMessageSourceResolvable;
import org.springframework.http.HttpHeaders;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.validation.FieldError;
import org.springframework.validation.ObjectError;
import org.springframework.web.bind.MethodArgumentNotValidException;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.context.request.WebRequest;
import org.springframework.web.servlet.mvc.method.annotation.
 ↪ResponseEntityExceptionHandler;
import org.terasoluna.gfw.common.exception.BusinessException;
import org.terasoluna.gfw.common.exception.ResourceNotFoundException;
import org.terasoluna.gfw.common.exception.ResultMessagesNotificationException;
import org.terasoluna.gfw.common.message.ResultMessage;

@ControllerAdvice
public class RestGlobalExceptionHandler extends ResponseEntityExceptionHandler {

    @Inject
    MessageSource messageSource;

    @Override
    protected ResponseEntity<Object> handleExceptionInternal(Exception ex,
            Object body, HttpHeaders headers, HttpStatus status,
            WebRequest request) {
        Object responseBody = body;
        if (body == null) {
            responseBody = createApiError(request, "E999", ex.getMessage());
        }
        return ResponseEntity.status(status).headers(headers).body(responseBody);
    }
```

```
    private ApiError createApiError(WebRequest request, String errorCode,
            Object... args) {
        return new ApiError(errorCode, messageSource.getMessage(errorCode,
                args, request.getLocale()));
    }

    @Override
    protected ResponseEntity<Object> handleMethodArgumentNotValid(
            MethodArgumentNotValidException ex, HttpHeaders headers,
            HttpStatus status, WebRequest request) {
        ApiError apiError = createApiError(request, "E400");
        for (FieldError fieldError : ex.getBindingResult().getFieldErrors()) {
            apiError.addDetail(createApiError(request, fieldError, fieldError
                    .getField()));
        }
        for (ObjectError objectError : ex.getBindingResult().getGlobalErrors()) {
            apiError.addDetail(createApiError(request, objectError, objectError
                    .getObjectName()));
        }
        return handleExceptionInternal(ex, apiError, headers, status, request);
    }

    private ApiError createApiError(WebRequest request,
            DefaultMessageSourceResolvable messageSourceResolvable,
            String target) {
        return new ApiError(messageSourceResolvable.getCode(), messageSource
                .getMessage(messageSourceResolvable, request.getLocale()), target);
    }

    @ExceptionHandler(BusinessException.class)
    public ResponseEntity<Object> handleBusinessException(BusinessException ex,
            WebRequest request) {
        return handleResultMessagesNotificationException(ex, new HttpHeaders(),
                HttpStatus.CONFLICT, request);
    }

    private ResponseEntity<Object> handleResultMessagesNotificationException(
            ResultMessagesNotificationException ex, HttpHeaders headers,
            HttpStatus status, WebRequest request) {
        ResultMessage message = ex.getResultMessages().iterator().next();
        ApiError apiError = createApiError(request, message.getCode(), message
                .getArgs());
        return handleExceptionInternal(ex, apiError, headers, status, request);
    }

    @ExceptionHandler(ResourceNotFoundException.class)
    public ResponseEntity<Object> handleResourceNotFoundException(
            ResourceNotFoundException ex, WebRequest request) {
        return handleResultMessagesNotificationException(ex, new HttpHeaders(),
```

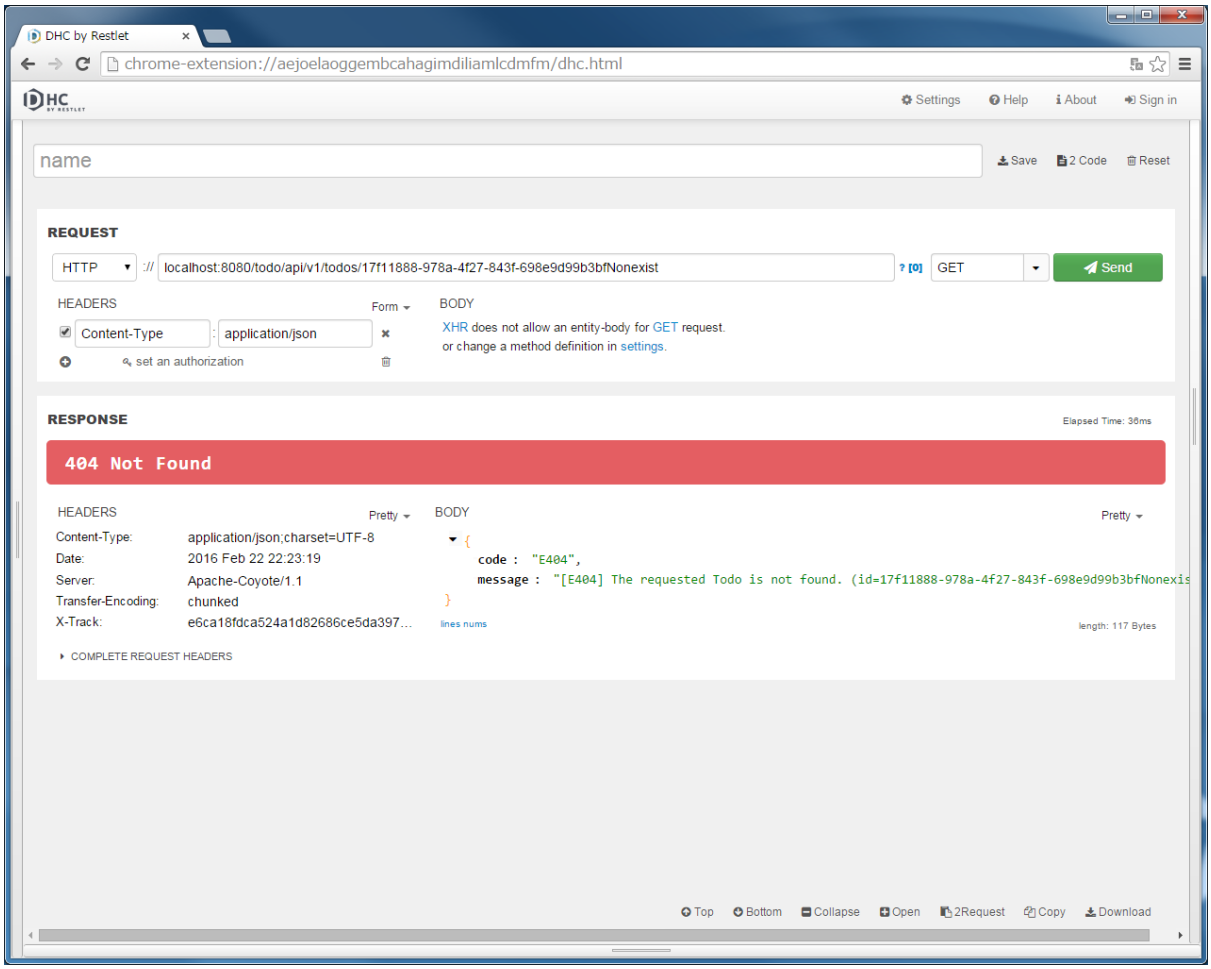
```
                HttpStatus.NOT_FOUND, request);
    }

}
```

Check the operation of the implemented error handling using DHC.

Open the DHC, enter "`localhost:8080/todo/api/v1/todos/{todoId}`" in the URL and specify GET in method.

Specifying the ID that does not exist in {todoId} portion and click “Send” button.

HTTP status returned “404 Not Found” and JSON error information displays in [Body] of [RESPONSE] part.



System exception error handling

Lastly, System exception handles by adding `java.lang.Exception` method in the `RestGlobalExceptionHandler`.

Set "500 InternalServerError" in HTTP status if System exception occurred.

`src/main/java/todo/api/common/error/RestGlobalExceptionHandler.java`

```
package todo.api.common.error;

import javax.inject.Inject;

import org.springframework.context.MessageSource;
import org.springframework.context.support.DefaultMessageSourceResolvable;
import org.springframework.http.HttpHeaders;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.validation.FieldError;
import org.springframework.validation.ObjectError;
import org.springframework.web.bind.MethodArgumentNotValidException;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.context.request.WebRequest;
import org.springframework.web.servlet.mvc.method.annotation.
↪ResponseEntityExceptionHandler;
import org.terasoluna.gfw.common.exception.BusinessException;
import org.terasoluna.gfw.common.exception.ResourceNotFoundException;
import org.terasoluna.gfw.common.exception.ResultMessagesNotificationException;
import org.terasoluna.gfw.common.message.ResultMessage;

@ControllerAdvice
public class RestGlobalExceptionHandler extends ResponseEntityExceptionHandler {

    @Inject
    MessageSource messageSource;

    @Override
    protected ResponseEntity<Object> handleExceptionInternal(Exception ex,
            Object body, HttpHeaders headers, HttpStatus status,
            WebRequest request) {
        Object responseBody = body;
        if (body == null) {
            responseBody = createApiError(request, "E999", ex.getMessage());
        }
        return ResponseEntity.status(status).headers(headers).body(responseBody);
    }
```

```
    private ApiError createApiError(WebRequest request, String errorCode,
            Object... args) {
        return new ApiError(errorCode, messageSource.getMessage(errorCode,
                args, request.getLocale()));
    }

    @Override
    protected ResponseEntity<Object> handleMethodArgumentNotValid(
            MethodArgumentNotValidException ex, HttpHeaders headers,
            HttpStatus status, WebRequest request) {
        ApiError apiError = createApiError(request, "E400");
        for (FieldError fieldError : ex.getBindingResult().getFieldErrors()) {
            apiError.addDetail(createApiError(request, fieldError, fieldError
                    .getField()));
        }
        for (ObjectError objectError : ex.getBindingResult().getGlobalErrors()) {
            apiError.addDetail(createApiError(request, objectError, objectError
                    .getObjectName()));
        }
        return handleExceptionInternal(ex, apiError, headers, status, request);
    }

    private ApiError createApiError(WebRequest request,
            DefaultMessageSourceResolvable messageSourceResolvable,
            String target) {
        return new ApiError(messageSourceResolvable.getCode(), messageSource
                .getMessage(messageSourceResolvable, request.getLocale()), target);
    }

    @ExceptionHandler(BusinessException.class)
    public ResponseEntity<Object> handleBusinessException(BusinessException ex,
            WebRequest request) {
        return handleResultMessagesNotificationException(ex, new HttpHeaders(),
                HttpStatus.CONFLICT, request);
    }

    private ResponseEntity<Object> handleResultMessagesNotificationException(
            ResultMessagesNotificationException ex, HttpHeaders headers,
            HttpStatus status, WebRequest request) {
        ResultMessage message = ex.getResultMessages().iterator().next();
        ApiError apiError = createApiError(request, message.getCode(), message
                .getArgs());
        return handleExceptionInternal(ex, apiError, headers, status, request);
    }

    @ExceptionHandler(ResourceNotFoundException.class)
    public ResponseEntity<Object> handleResourceNotFoundException(
            ResourceNotFoundException ex, WebRequest request) {
        return handleResultMessagesNotificationException(ex, new HttpHeaders(),
```

```
                HttpStatus.NOT_FOUND, request);
    }

    @ExceptionHandler(Exception.class)
    public ResponseEntity<Object> handleSystemError(Exception ex,
            WebRequest request) {
        ApiError apiError = createApiError(request, "E500");
        return handleExceptionInternal(ex, apiError, new HttpHeaders(),
                HttpStatus.INTERNAL_SERVER_ERROR, request);
    }

}
```

Check the operation of the implemented error handling using DHC.

In order to generate a system error, boot the application in the state of Database tables are not created.

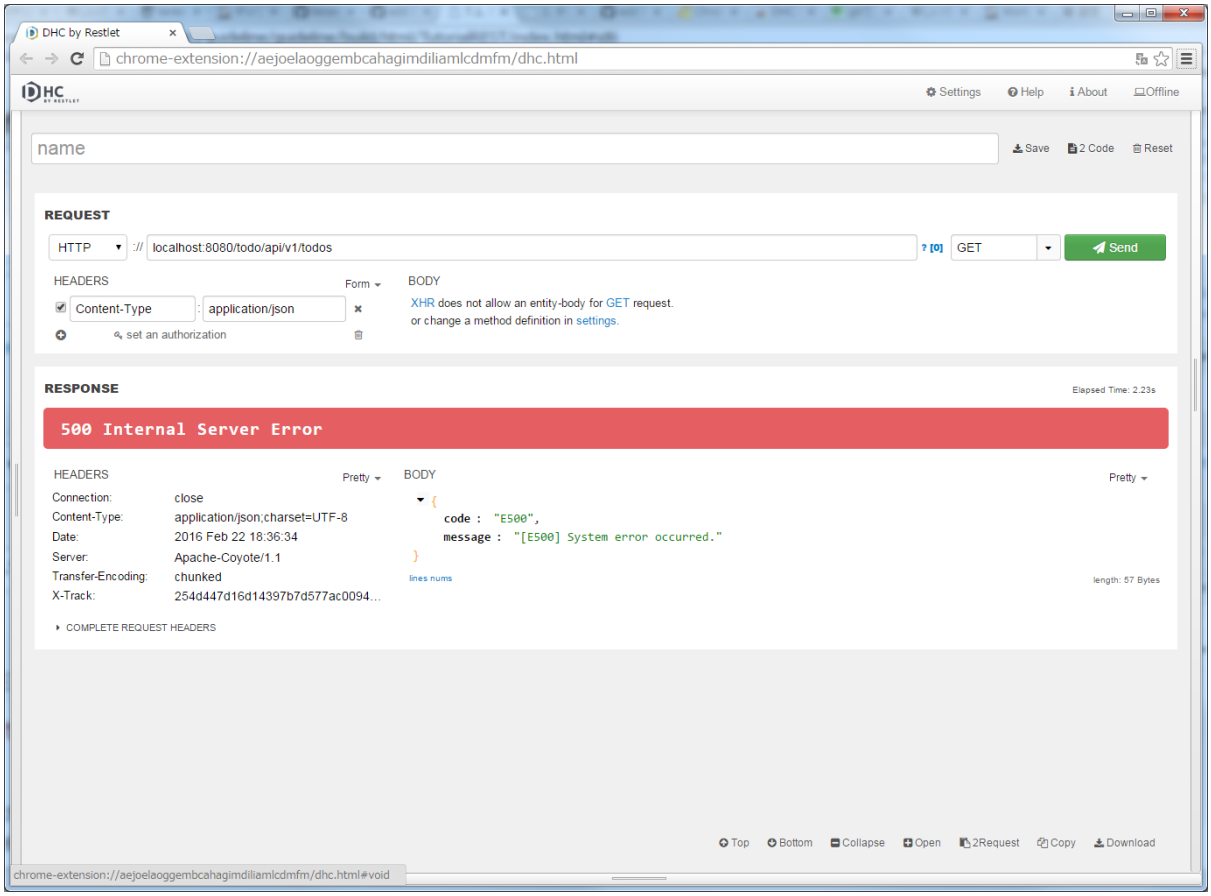
`src/main/resources/META-INF/spring/todo-infra.properties`

```
database=H2
#database.url=jdbc:h2:mem:todo;DB_CLOSE_DELAY=-1;INIT=create table if not exists␣
↪todo(todo_id varchar(36) primary key, todo_title varchar(30), finished boolean,␣
↪created_at timestamp)
database.url=jdbc:h2:mem:todo;DB_CLOSE_DELAY=-1
database.username=sa
database.password=
database.driverClassName=org.h2.Driver
# connection pool
cp.maxActive=96
cp.maxIdle=16
cp.minIdle=0
cp.maxWait=60000
```

Open the DHC, enter "`localhost:8080/todo/api/v1/todos/`" in the URL and click the "Send" button after specifying GET in method.

HTTP status returned "500 Internal Server Error" and JSON error information displays in [Body] of [RESPONSE]

part.



Note: In case of system error occurred, it is recommended to set a simple error message from which cause of error can not be identified while error message returning to the client. When you set the error message from which cause of error is identified, there is a possibility to exposes the vulnerability of the system to the client and may cause security issues.

It is good to flush the cause of an error into error analysis log. The default setting of Blank project has been outputting the log by `ExceptionLogger` provided in the common library therefore setting and implementation for outputting the log is not required.

The log output by `ExceptionLogger` is as follows.

The cause of the system error can be understood that the Todo table is not exist.

```
date:2015-01-19 02:08:47        thread:tomcat-http--4   X-Track:
↪aadf5822205d423c95a6531f2f76036f        level:ERROR     logger:o.t.gfw.
↪common.exception.ExceptionLogger         message:[e.xx.fw.9002]
### Error querying database.  Cause: org.h2.jdbc.JdbcSQLException: Table "TODO
↪" not found; SQL statement:
SELECT
            todo_id,
```

```
            todo_title,
            finished,
            created_at
        FROM
            todo [42102-182]
### The error may exist in todo/domain/repository/todo/TodoRepository.xml
### The error may involve todo.domain.repository.todo.TodoRepository.findAll
### The error occurred while executing a query

... (omitted)
```

10.2.4 In the end...

In this tutorial, following contents have been learnt.

- How to develop basic RESTful Web service by TERASOLUNA Server Framework for Java (5.x)
- Implementation of Controller class that offers REST API(GET, POST, PUT, DELETE)
- Cross conversion method of JavaBean and JSON
- Error message definition method
- Method of handling a variety of exception with Spring MVC

Here, explained how to implement the basic RESTful Web Services. To learn more about the architecture and design guidelines etc, Refer [*RESTful Web Service*].

10.3 Session tutorial

10.3.1 Introduction

Flow of learning

In this tutorial, users will learn the how to design data for the session management and the specific method of implementation to use the session by creating a simple web application. This tutorial will be implemented with the following flow.

1. To check the requirements of web application to be created
2. To check the method of implementation for the Controller and the procedure for designing data to meet the requirements
3. To implement on the basis of the design information

Lessons to be learnt in the tutorial

- How to design data for the session management
  - Selecting data to be stored in the session
  - Discarding data in the session
- Specific method of using the session in this FW
  - How to use @SessionAttributes
  - How to use the Bean for session scope

Target Readers

- Tutorial: Implements Todo application
- Tutorial: Implements Spring Security

Test environment

This tutorial verifies the operation in the following environments.

Type	Product
OS	Windows 7
JVM	Java 1.8
IDE	Spring Tool Suite 3.6.4.RELEASE (hereafter called as 'STS')
Build Tool	Apache Maven 3.3.3 (hereafter called as 'Maven')
Application Server	Pivotal tc Server Developer Edition v3.1 (Included in STS)
Web Browser	Google Chrome 42.0.2311.90 m

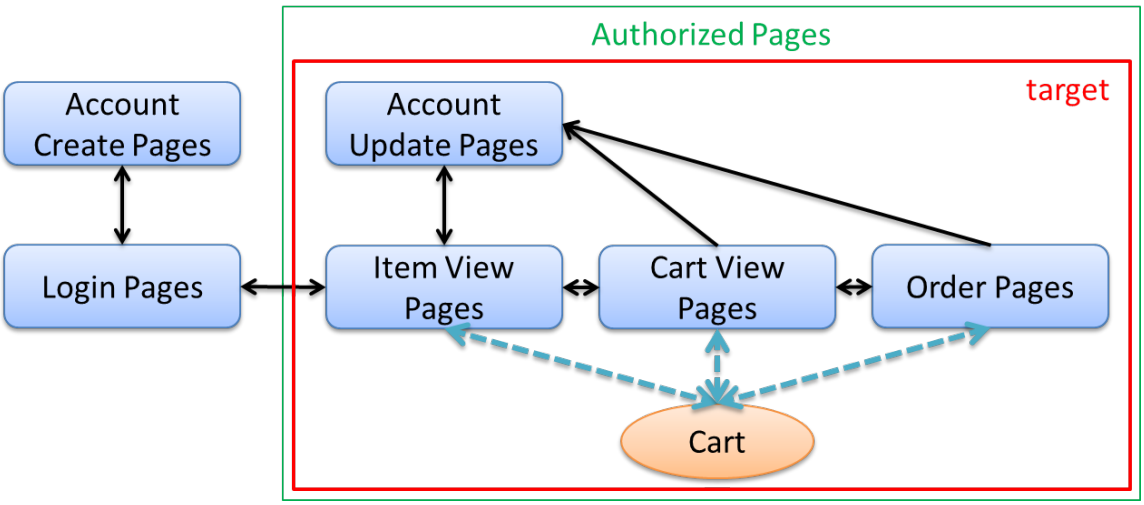
10.3.2 Application overview and requirements

Overview

Create a simple EC site. User can perform the following operations on the EC site.

- Login to the account
- Create an account
- Change the information of the account created
- View the list of products handled on the EC site
- View the product details
- Register the products to be purchased in a cart
- Remove the products registered in the cart from the cart
- Place order for the products in the cart

Application overview is shown in the following diagram. XxxPages in the diagram indicate a set of screens. In this tutorial, the interaction between the system and the user performed on 1 screen set is handled as 1 use case.



Requirements

Functional requirements

Implement the following function for each screen (use case) described earlier in this application.

Screen (use case)	Function
Login Pages	Login function **(created)**
Account Create Pages	Account creation function **(created)**
Account Update Pages	Account information change function
Item View Pages	Product list view function **(created)** Product details view function **(created)** Cart item registration function
Cart View Pages	Cart item deletion function
Order Pages	Product order function

Some functions have been created in advance in the project which are offered as the initial material for this tutorial. This is done to reduce the cost to create the part that is not directly related to the session management.

Create unfinished function in this tutorial. Further, implementation of domain layer/infrastructure layer has been created in unfinished function as well. Therefore, create screens for unfinished function and application layer in this tutorial.

Non-functional requirements

It is necessary to design and implement the application by considering the non-functional requirements required of that system while creating the real application. Design/create the application in this tutorial with the assumption of non-functional requirements in this tutorial. The specific numerical values for each requirement shown below

are the hypothetical values used for learning. It should be noted that it cannot be guaranteed that the application created in this tutorial will actually meet the requirements.

Availability

- Operation period: 24 hours
- A planned downtime of few days in a year
- Allow a downtime for about 1 hour
- Target the fault recovery within 1 business day
- Utilization: 99%

Usability

- Operation is not guaranteed on multiple browsers and tabs

Performance

- Number of users: 10,000
- Number of concurrent access users: 200
- Number of online process records: 10,000/month
- The number of users/number of concurrent access users/number of online process records together are expected to increase 1.2 times in 1 year

It is necessary to consider the above requirements while reviewing the following items to design the session management.

Requirement	Items to be reviewed
Availability	• Status of replication in multiple-server operation
Usability	• Retention of data integrity
Performance	• Status of replication in multiple-server operation • Memory utilization

Further, passing of important information including personal information/credit card information also should be considered in the design of the session management other than the items mentioned above.

Configuration of platform

Application created in this tutorial is to be operated on the following platforms. The specific numerical values for the configuration shown below are the hypothetical values used for learning.

- Consider 2 configurations for each server of Web/AP/DB.
- Memory load of AP server is 8GB and 2 slots are free

It is necessary to consider the above configuration while reviewing the memory utilization and replication status to design the session management.

10.3.3 Application design

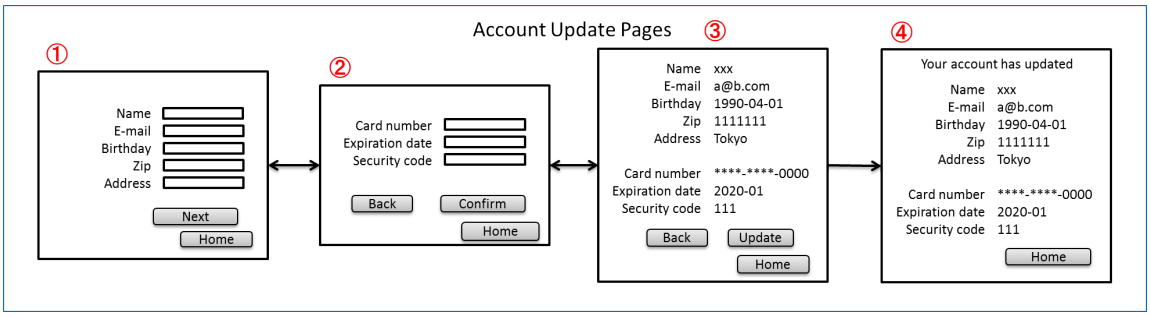
Determine the policy to create the application based on the requirements described earlier. Since the domain layer/infrastructure layer have been created in this tutorial, consider only the items related to the application layer as the target. Further, since this tutorial aims at learning the method to use the session, description for the items those are not directly related to the session management is omitted.

> **Warning:** Note that an example of the process that uses the session is shown in this chapter. It is required to follow the work instructions/operating procedures for each project in the real development.

Screen definition

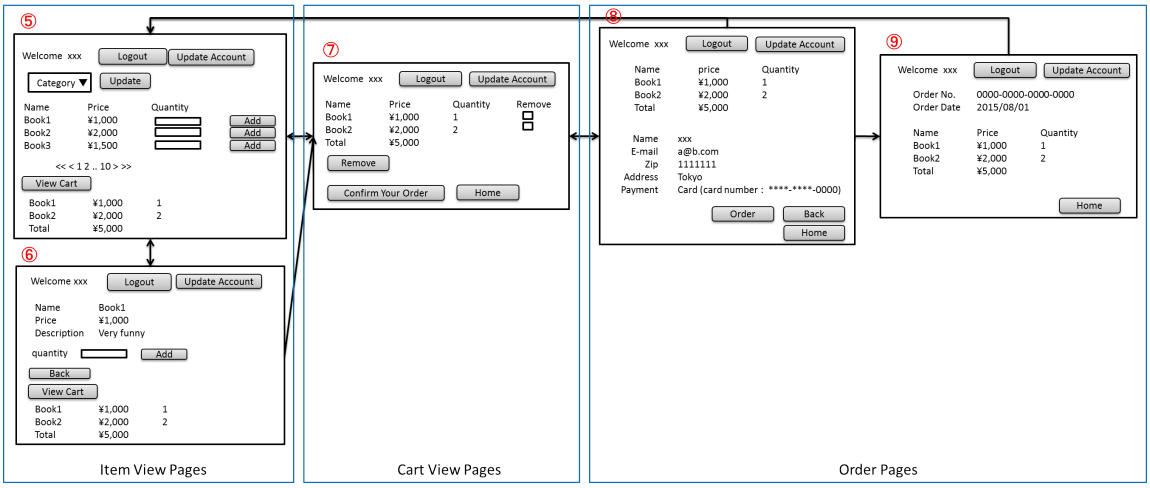
Define the screen on which the application is displayed, based on the requirements. Details for the screen definition process are omitted.

Image for the screen to be created in this tutorial defined in the end, is as follows.



Some of the transitions are given below which are omitted in the above diagram.

- When the user logins from the Login screen, the transition takes place to screen (5)
- When 'Home' button is clicked on each screen of Account Update Pages, the transition takes place to screen (5)



- When ‘Update Account’ button is clicked on each screen of Item View Pages, Cart View Pages and Order Pages, the transition takes place to screen (1)
- When ‘Logout’ button is clicked on each screen of Item View Pages, Cart View Pages and Order Pages, the transition takes place to Login screen

Extracting URL

The application determines the URL that does processing, based on the screen image.

Set the URL and parameters for each event occurred from each screen. Assign the respective names as per the following standards.

- URL:/<Use case name>
- Parameter:?<Process name>

Since the use cases are divided into account creation and update in this application, set the URLs as /account/create and /account/update respectively.

Further, also determine the Controller to process each URL. Basically process 1 use case in 1 Controller.

Finally the extracted URL can be arranged as follows. The Controller that is mentioned as “Created”, exists in the project provided as the initial material. Further, the process wherein the path mentioned as “Created” is accessed, is already described within the created Controller mentioned earlier.

Sr. No.	Process name	HTTP method	Path	Controller name	Screen
(1)	Account information change screen 1 display process	GET	/account/update?form1	AccountUpdateController	/account/updateForm1
(2)	Account information change screen 2 display process	POST	/account/update?form2	AccountUpdateController	/account/updateForm2
(3)	Account information change confirmation screen display process	POST	/account/update?confirm	AccountUpdateController	/account/updateConfirm
(4)	Account information change process	POST	/account/update	AccountUpdateController	Redirect to Account information change completion screen display process
(5)	Account information change completion screen display process	GET	/account/update?finish	AccountUpdateController	/account/updateFinish
(6)	Process to return to Account information change screen 1	POST	/account/update?redoform1	AccountUpdateController	/account/updateForm1
(7)	Process to return to Account information change screen 2	POST	/account/update?redoform2	AccountUpdateController	/account/updateForm2

次のページに続く

Table. 10.5 – 前のページからの続き

Sr. No.	Process name	HTTP method	Path	Controller name	Screen
(8)	Process to return to Home	GET	/account/update?home	AccountUpdateController	Redirect to Product list screen display process
(9)	Product list screen display process (default)	GET	/goods **(created)**	GoodsController **(created)**	/goods/showGoods
(10)	Product list screen display process (while selecting category)	GET	/goods?categoryId **(created)**	GoodsController **(created)**	/goods/showGoods
(11)	Product list screen display process (while selecting page)	GET	/goods?page **(created)**	GoodsController **(created)**	/goods/showGoods
(12)	Product details screen display process	GET	/goods?{goodsId} **(created)**	GoodsController **(created)**	/goods/showGoodsDetail
(13)	Process to add a product in the cart	GET	/addToCart	GoodsController **(created)**	Redirect to Product list screen display process
(14)	Cart screen view process	GET	/cart	CartController	cart/viewCart

次のページに続く

Table. 10.5 – 前のページからの続き

Sr. No.	Process name	HTTP method	Path	Controller name	Screen
(15)	Process to delete a product from the cart	POST	/cart	CartController	Redirect to cart screen display process
(16)	Order confirmation screen display process	GET	/order?confirm	OrderController	order/confirm
(17)	Order processing	POST	/order	OrderController	Redirect to order completion screen display process
(18)	Order completion screen display process	GET	/order?finish	OrderController	order/finish

Designing input/output data

Design the input/output data handled by the application based on the screen image.

Extracting data

Extract the input/output data handled by the application screen. The following data can be extracted based on the screen image mentioned earlier.

Sr. No.	Data item name	Data element
(1)	Update account information	Account name, mail address, birthdate, postal code, address, card number, validity, security code
(2)	Account information	Account name, mail address, password, birthdate, postal code, address, card number, validity, security code
(3)	Product search information	Selection category, page number
(4)	Product information	Product name, unit price, description, (product ID)
(5)	Register cart information	Quantity, (product ID)
(6)	Cart information	Product name, unit price, quantity, (product ID)
(7)	Delete cart information	Product ID list
(8)	Order information	Order ID, order date-time, (account ID), product name, unit price, quantity

Defining lifecycle

Define the lifecycle of the data extracted in the previous paragraph. Determine when the data will be generated and when it will be discarded in the definition of the lifecycle.

Also note that the data which is to be retained in multiple screens will have multiple discard timings as below.

- The task is terminated with the usual flow
- Task is cancelled during the process

If the above precautions are considered, the lifecycle of the data extracted in the previous paragraph can be defined as follows.

Sr. No.	Data item name	Lifecycle
(1)	Update account information	Generate the data with the input from screen (1) and retain it during the transition from (1) to (3). Discard it when there is a transition except from screen (1) to (3).
(2)	Account information	Generate the data at the time of login and discard at the time of logout.
(3)	Product search information	Generate the data when there is a transition to screen (5) and retain during the transition from (1) to (8). Discard when there is a transition to screen (9).
(4)	Product information	Generate the data when there is a transition to screen (5) or (6) and retain only during that request.
(5)	Register cart information	Generate the data by the input from screen (5) or (6) and retain only during that request.
(6)	Cart information	Generate an empty object when there is a transition to screen (5) and retain during the transition from (1) to (8). Discard when there is a transition to screen (9).
(7)	Delete cart information	Generate the data by the input from screen (7) and retain only during that request.
(8)	Order information	Generate the data when there is a transition to screen (9) and retain only during that request.

Determining the status of using a session

When the information needs to be retained on multiple screens, implementation is easy by using a session. At the same time, when the session is used, its demerits also need to be considered. Refer to the guideline :doc:'../ArchitectureInDetail/WebApplicationDetail/SessionManagement' and determine whether to use the session in this tutorial.

It is described in the guideline that it is recommended to use the session first and to store only the data that is necessary, in the session. This tutorial also considers not using the session.

Data item	Review contents
Update account information	The data needs to be passed using hidden since the Update account information is to be retained across 3 screens. However, the Update account information contains the important information like the card number. It is a security issue since the important information is written in the HTML source without masking the important information while passing data using hidden. Therefore, consider using the session in this tutorial.
Account information	Since it is retained on all screens after login, the data needs to be passed using hidden. The process of passing data must be described on almost all screens created in this case. Therefore, consider using the session in this tutorial to reduce the implementation cost for screens as well.
Product search information	The data needs to be passed using hidden since the Product search information is to be retained across 8 screens. The process of passing data must be described on almost all screens created in this case. Therefore, consider using the session in this tutorial to reduce the implementation cost for screens as well.
Product information	Since the Delete cart information is used only in 1 screen, the data should be handled in the request scope.
Register cart information	Since the Delete cart information is used only in 1 screen, the data should be handled in the request scope.

次のページに続く

Table. 10.6 – 前のページからの続き

Data item	Review contents
Cart information	The data needs to be passed using hidden since the Cart information is to be retained across 8 screens. The process of passing data must be described on almost all screens created in this case. Therefore, consider using the session in this tutorial to reduce the implementation cost for screens as well.
Delete cart information	Since the Delete cart information is used only in 1 screen, the data should be handled in the request scope.
Order information	Since the Order information is used only in 1 screen, the data should be handled in the request scope.

From the above-mentioned, consider using the session for these 4 i.e. the Update account information, Account information, Cart information and Product search information.

Next, verify the demerits of using the session. According to this verification, if it is determined that the impact of demerits cannot be ignored, do not use the session.

The following 3 major points can be mentioned as the demerits of using the session.

- When it is used on multiple tabs and multiple browsers, (it needs to be considered that) the integrity of data may be lost because of mutual operations.
- Since it is managed on the memory, the memory is likely to be exhausted because of the size of the data to be managed.
- The session replication needs to be considered when AP server multiplexing is done with the aim to implement scale-out or to achieve high availability. At that time, if large data is handled in the session, it may affect the performance etc.

Consider how to handle the respective risks or whether to allow the risks regarding the above-mentioned point of view.

Point of view	Review contents
Data integrity	The operation on multiple browsers and tabs is not guaranteed in this application. Therefore, a countermeasure to secure the data integrity is not required.
Memory utilization	Estimate the data size for which usage of session is considered. Assume maximum 100 characters 240 bytes (4 characters 8 bytes + initial 40 bytes) for the character string element, 24 bytes for data element and 16 bytes for the numerical value element. Further, the authentication information stored in the session at the time of login authentication also contains the size of “UserDetails”. “UserDetails” broadly contains ID, password and user rights. Multiple user rights can be specified, but assume here as 1. The result estimated for each item is as follows. • Account information (Character string: 7, Date: 2): maximum 1.7K bytes • Change account information (Character string: 8, Date: 2): maximum 2.0K bytes • Cart information (maximum 19 products x (Character string: 3, numerical value: 3)): maximum 14.6K bytes • Product search information (numerical value: 2): 32 bytes • “UserDetails”: (Character string: 3): 0.7K byte 1 user uses maximum 19KB in total. 1 user uses approximately 21KB if safety factor of 10% is considered. Since the usage is about 210MB even if it is considered that 10000 people are simultaneously connected and the memory load is considerably below 8GB even if other memory utilization is considered, it is less likely that the memory exhaustion will happen.
AP server multiplexing	Since high availability is not required in this application, use case continuation when failure happens is not required and redoing of use case because of re-login is allowed. Therefore, only take the countermeasure to set the load balancer so that all the requests occurred within the same session are distributed to the same AP server and do not implement the replication among the AP servers of the session.

Warning: Tool (for example, like SizeOf) needs to be used to measure the object size for estimation of the object size. The calculation formula in this tutorial refers the trend in actually measured values in SizeOf, but note that ultimately it is a temporary value. It should be separately considered how to calculate it at the time of sizing in the real system development.

Warning: The data to be stored in the session is basically restricted to the input data to avoid memory exhaustion. Since the size of the output data for search results tends to increase and on the other hand, often it is read-only that cannot be edited using screen operations, it is not suitable for storing in the session.

Increase in the management cost of session key is also 1 of the points to be considered other than the above-mentioned. However, since the data quantity to be stored in the session is not large in the application created this time, it can be said that the management cost of the session key is limited.

It can be said from this result that the impact of demerits happening because of using session is not large. Data to be stored in the session finally is as follows.

- Change account information
- Account information
- Product search information
- Cart information

This tutorial concludes that passing of data is implemented using the session. However, it is also considered as a result of the investigation that it concludes that session should not be used. Implement passing the data using hidden as an example when the session is not used.

Further, when the session is used, sometimes method to maintain the data integrity and replication settings are required.

The guideline mentions a method to avoid it using transaction token check. However, note that it becomes a low-usability application in this case. *Double Submit Protection* should be referred for the specific implementation method.

Since replication settings depend on AP server, when replication needs to be considered, configuration of AP server needs to be checked.

Warning: Sometimes there exists a data to be stored in the session other than the data determined here. Session is used when the following items among the items in the guideline are used.

- It uses authentication/authorization/CSRF countermeasures using Spring Security
- It uses transaction token check for prevention of double transmission

Method of implementation to use data during the session

Method of implementation to use the data during the session for each data is determined in this section.

The guideline provides 2 implementation methods corresponding to the locations of using data. *Session Management* classifies the methods to be used depending on whether the data is contained within 1 Controller. Therefore, the implementation method needs to be decided considering the lifecycle of data to be stored in the session and URL mapping. Further, when the data is associated with the authentication information, session management should be implemented by the Spring Security function.

Considering these, the final result for which the data handled in the session is organized is as follows.

Data	Characteristics	Method of using data during session
Change account information	Used only within 1 Controller	Method wherein @SessionAttributes annotation is used
Account information	Used among multiple Controllers Used in authentication process	Method wherein Spring Security function is used
Product search information	Used among multiple Controllers	Method wherein Bean for session scope of Spring is used
Cart information	Used among multiple Controllers	Method wherein Bean for session scope of Spring is used

Account information is already created in the project provided as initial material and is managed using the Spring Security function. Therefore, this tutorial does not describe any specific method of using. :doc:'../Security/Authentication' should be referred for the specific method of using.

Considerations while using session

Items mentioned hereafter need to be considered when it is decided to use the session. Review the respective items.

Session synchronization

Object stored in the session can be simultaneously accessed by means of multiple requests of the same user. Therefore, when session synchronization is not performed, it can cause unexpected error or operation.

Since the guideline mentions a method of implementing synchronization where BeanProcessor is used in *Session*

Management, use this in this tutorial.

Session time out

Session time out needs to be set when session is used. If timeout duration is too long, unnecessary resources are retained in the memory and if timeout duration is too small, user friendliness is lowered. Therefore, appropriate time needs to be set as per the requirement.

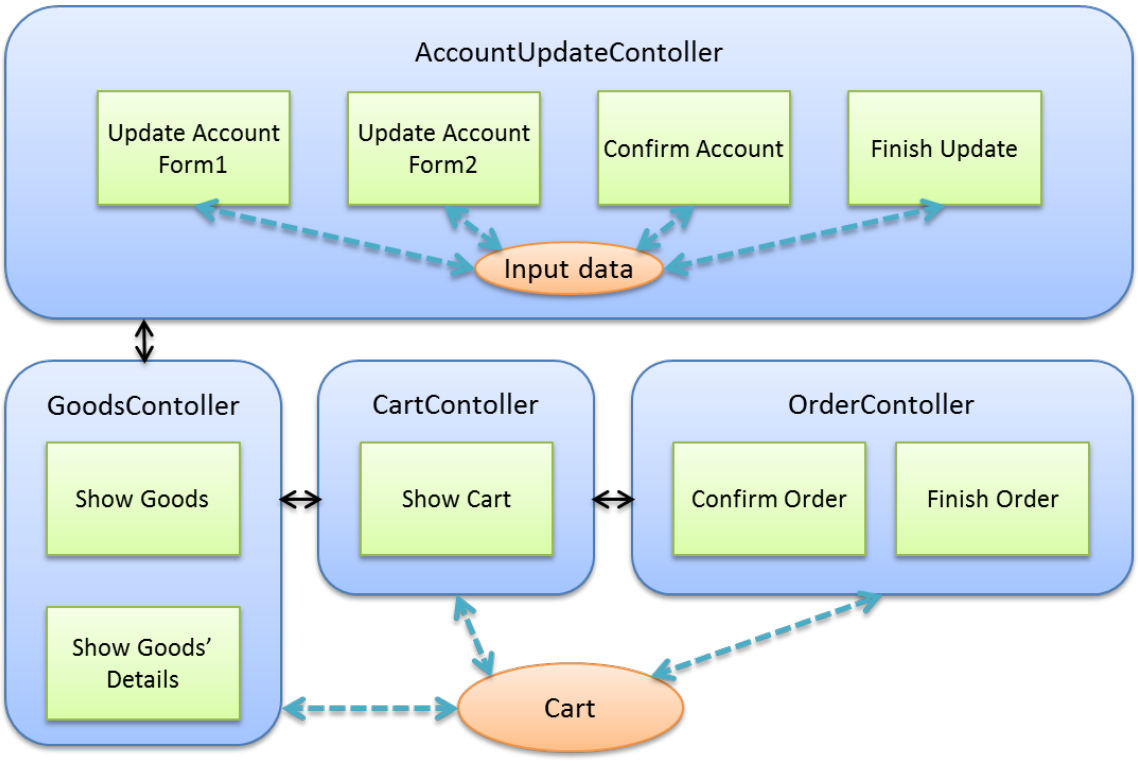
This tutorial is also well-equipped with memory resources, set the default value of AP server to 30 minutes.

Further, handling the requests after session timeout also needs to be reviewed. The guideline mentions a method to handle the requests after session timeout in *Session Management*.

Settings are done so as to transit to the login screen after timeout in this tutorial.

Overall application design

Overall image diagram of the final application design is shown below.



10.3.4 Project configuration

Project creation

As stated already, this tutorial starts with the status that some functions are created. Therefore, proceed with development using already created project.

Created project can be fetched with the following procedure.

1. Access to tutorial-apps.
2. Click 'Branch' button to select Branch of the required version and click 'Download ZIP' button to download zip file
3. Extract zip file and import the project in it.

Note that since the method to import the project is described in *Tutorial (Todo Application)*, the description is omitted in this tutorial.

Project configuration

It states about the configuration of the initial project fetched using git. Only the differences between the project fetched and the blank project are shown below.

```
session-tutorial-init-domain
    └── src
        └── main
             ├── java
             |   └── com
             |       └── example
             |           └── session
             |               └── domain
             |                   ├── model  ... (1)
             |                   |   ├── Account.java  ... (2)
             |                   |   ├── Cart.java  ... (3)
             |                   |   ├── CartItem.java  ... (3)
             |                   |   ├── Goods.java
             |                   |   ├── Order.java  ... (4)
             |                   |   └── OrderLine.java  ... (4)
             |                   ├── repository  ... (5)
             |                   |   ├── account
             |                   |   |   └── AccountRepository.java
             |                   |   ├── goods
             |                   |   |   └── GoodsRepository.java
             |                   |   └── order
             |                   |       └── OrderRepository.java
             |                   └── service  ... (6)
             |                       ├── account
             |                       |   └── AccountService.java
```

```
|                       ├-- goods
|                       |   └-- GoodsService.java
|                       ├-- order
|                       |   ├-- EmptyCartOrderException.java
|                       |   ├-- InvalidCartOrderException.java
|                       |   └-- OrderService.java
|                       └-- userdetails
|                           ├-- AccountDetails.java
|                           └-- AccountDetailsService.java
└-- resources
     ├-- com
     |   └-- example
     |       └-- session
     |           └-- domain
     |               └-- repository  ... (7)
     |                   ├-- account
     |                   |   └-- AccountRepository.xml
     |                   ├-- goods
     |                   |   └-- GoodsRepository.xml
     |                   └-- order
     |                       └-- OrderRepository.xml
     └-- META-INF
          ├-- dozer
          |   └-- order-mapping.xml  ... (8)
          └-- spring
              └-- session-tutorial-init-codelist.xml  ... (9)
```

Sr. No.	Description
(1)	Package to handle model used in this application. It describes the model required to be understood for proceeding with the tutorial below in detail.
(2)	Class to save user account information.
(3)	Class to save information of product registered in cart by user. 'Cart' manages the whole thing and 'CartItem' manages the individual products.
(4)	Class to save information of product ordered by user. 'Order' manages the whole thing and 'OrderLine' manages the individual products.
(5)	Package to handle repository used in this application.
(6)	Package to handle service used in this application.
(7)	Directory to store mapping file used in repository.
(8)	Mapping definition file of Dozer (Bean Mapper). Conversion from 'Cart' to 'Order' is defined.
(9)	Bean definition file in which code list used in this application is defined.

```
session-tutorial-init-env
    └── src
        └── main
            └── resources
```

```
                └── database  ... (1)
                    ├── H2-dataload.sql
                    └── H2-schema.sql
```

File name	Description
(1)	Directory to store SQL in order to setup in-memory database (H2 Database) in this application.

```
session-tutorial-init-web
    └── src
        └── main
             ├── java
             |    └── com
             |        └── example
             |            └── session
             |                └── app  ... (1)
             |                    ├── account
             |                    |   ├── AccountCreateController.java
             |                    |   ├── AccountCreateForm.java
             |                    |   ├── IlleagalOperationException.java
             |                    |   └── IlleagalOperationExceptionHandler.java
             |                    ├── goods
             |                    |   ├── GoodsController.java
             |                    |   └── GoodsViewForm.java
             |                    ├── login
             |                    |   └── LoginController.java
             |                    └── validation
             |                        ├── Confirm.java
             |                        └── ConfirmValidator.java
             ├── resources
             |    ├── i18n
             |    |   └── application-messages.properties  ... (2)
             |    ├── META-INF
             |    |    └── spring  ... (3)
             |    |        ├── spring-mvc.xml
             |    |        └── spring-security.xml
             |    └── ValidationMessages.properties  ... (2)
             └── webapp
                  ├── resources  ... (4)
                  |   ├── app
                  |   |   └── css
                  |   |       └── styles.css
                  |   └── vendor
                  |       └── bootstrap-3.0.0
                  |           └── css
                  |               └── bootstrap.css
                  └── WEB-INF
```

```
                    └── views  ... (5)
                        ├── account
                        |   ├── createConfirm.jsp
                        |   ├── createFinish.jsp
                        |   └── createForm.jsp
                        ├── common
                        |   ├── error
                        |   |   └── illegalOperationError.jsp
                        |   └── include.jsp
                        ├── goods
                        |   ├── showGoods.jsp
                        |   └── showGoodsDetails.jsp
                        └── login
                            └── loginForm.jsp
```

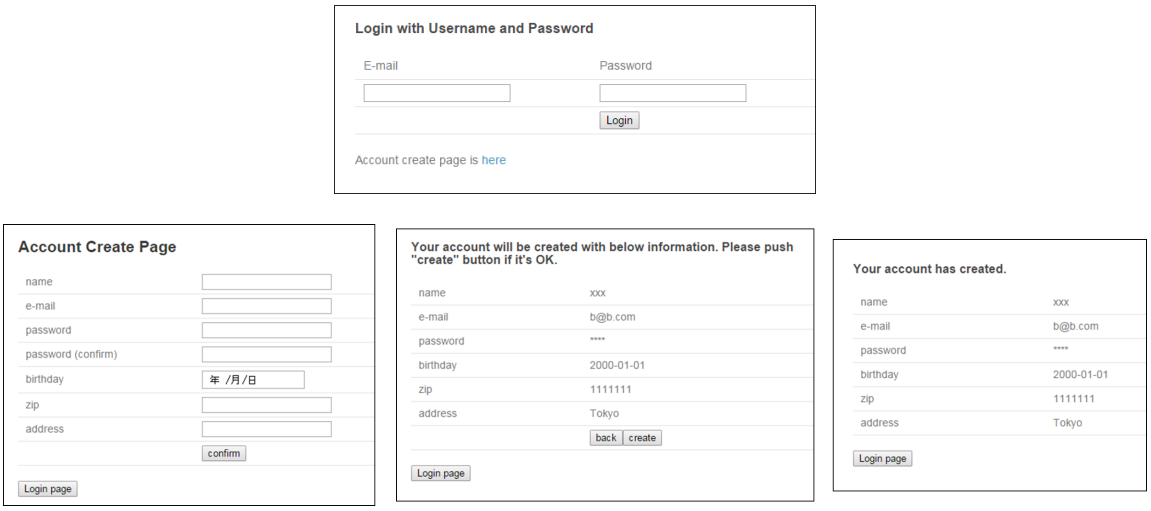
Sr. No.	Description
(1)	Package to store the class in application layer used in this application.
(2)	Property file in which message used in this application is defined
(3)	Bean definition file in which component used in this application is defined
(4)	Static resource file used in this application
(5)	Directory in which jsp used in this application is stored

Operation verification

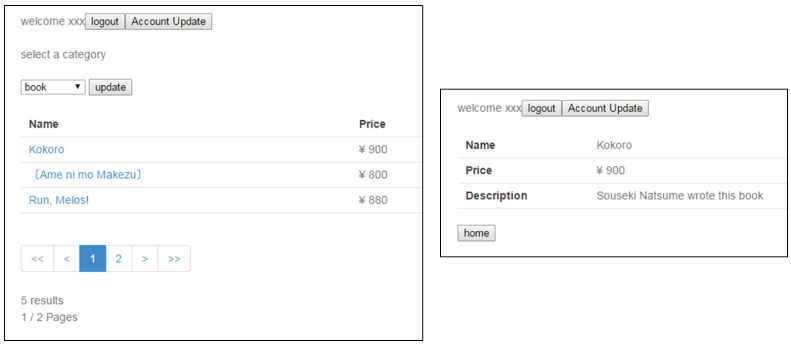
Check the operation of project fetched before the application development. Start the application server with the project imported in STS as the target The method to start the application server is omitted in this tutorial since it is described in *Tutorial (Todo Application)*.

The following screen is displayed when http://localhost:8080/session-tutorial-init-web/loginForm is accessed after the application server is started.

Account can be created when “here” link on the login screen is selected.



When (E-mail="a@b.com", Password="demo") is input in the form on the login screen, login can be done. Product list is displayed after the login. The product details can be displayed when the product name is selected.



10.3.5 Creating simple EC site application

Create account information change function

Create a function that allows the user to input the information and updates the account information.

Manage the Change account information using "@SessionAttributes annotation" as described in *Application design*.

The information of the screen implemented in the Change account information function is shown below.

Process name	HTTP method	Path	Screen
Account information change screen 1 display process	GET	/account/update?form1	/account/updateForm1
Account information change screen 2 display process	GET	/account/update?form2	/account/updateForm2
Account information change confirmation screen display process	GET	/account/update?confirm	/account/updateConfirm
Account information change process	POST	/account/update	Redirect to Account information change completion screen display process
Account information change completion screen display process	GET	/account/update?finish	/account/updateFinish
Process to return to Account information change screen 1	GET	/account/update?redoform1	/account/updateForm1
Process to return to Account information change screen 2	GET	/account/update?redoform2	/account/updateForm2
Process to return to Home	GET	/account/update?home	Redirect to Home screen display process

Creating form object

Create a class to retain the account change information.

"/session-tutorial-init-web/src/main/java/com/example/session/app/account/AccountUpdateForm.java"

```
package com.example.session.app.account;

import java.io.Serializable;
import java.util.Date;

import javax.validation.constraints.NotNull;
import javax.validation.constraints.Size;

import org.hibernate.validator.constraints.Email;
import org.springframework.format.annotation.DateTimeFormat;

public class AccountUpdateForm implements Serializable {  // (1)

    /**
     *
     */
    private static final long serialVersionUID = 1L;

    private String id;

    // (2)
    @NotNull(groups = { Wizard1.class })
    @Size(min = 1, max = 255, groups = { Wizard1.class })
    private String name;

    @NotNull(groups = { Wizard1.class })
    @Size(min = 1, max = 255, groups = { Wizard1.class })
    @Email(groups = { Wizard1.class })
    private String email;

    @NotNull(groups = { Wizard1.class })
    @DateTimeFormat(iso = DateTimeFormat.ISO.DATE)
    private Date birthday;

    @NotNull(groups = { Wizard1.class })
    @Size(min = 7, max = 7, groups = { Wizard1.class })
    private String zip;

    @NotNull(groups = { Wizard1.class })
    @Size(min = 1, max = 255, groups = { Wizard1.class })
    private String address;

    @Size(min = 16, max = 16, groups = { Wizard2.class })
    private String cardNumber;
```

```
    @DateTimeFormat(pattern = "yyyy-MM")
    private Date cardExpirationDate;

    @Size(min = 1, max = 255, groups = { Wizard2.class })
    private String cardSecurityCode;

    public String getId() {
        return id;
    }

    public void setId(String id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public Date getBirthday() {
        return birthday;
    }

    public void setBirthday(Date birthday) {
        this.birthday = birthday;
    }

    public String getZip() {
        return zip;
    }

    public void setZip(String zip) {
        this.zip = zip;
    }

    public String getAddress() {
        return address;
    }
```

```
    public void setAddress(String address) {
        this.address = address;
    }

    public String getCardNumber() {
        return cardNumber;
    }

    public void setCardNumber(String cardNumber) {
        this.cardNumber = cardNumber;
    }

    public Date getCardExpirationDate() {
        return cardExpirationDate;
    }

    public void setCardExpirationDate(Date cardExpirationDate) {
        this.cardExpirationDate = cardExpirationDate;
    }

    public String getCardSecurityCode() {
        return cardSecurityCode;
    }

    public void setCardSecurityCode(String cardSecurityCode) {
        this.cardSecurityCode = cardSecurityCode;
    }

    public String getLastFourOfCardNumber() {
        if (cardNumber == null) {
            return "";
        }
        return cardNumber.substring(cardNumber.length() - 4);
    }

    public static interface Wizard1 {

    }

    public static interface Wizard2 {

    }
}
```

Sr. No.	Description
(1)	Implement Serializable in advance to store the instance of this class in the session.
(2)	Make validation groups to specify the target of the input check for each screen transition. In the example above, 2 groups are created implement the input check corresponding to the input item on the 1st page and the input item on the 2nd page respectively.

Creating Controller

Create the Controller. The description in which the form that receives the input information is managed using "@SessionAttributes" annotation is required in the Controller.

"/session-tutorial-init-web/src/main/java/com/example/session/app/account/AccountUpdateController.java"

```
package com.example.session.app.account;

import javax.inject.Inject;

import org.dozer.Mapper;
import org.springframework.beans.propertyeditors.StringTrimmerEditor;
import org.springframework.security.core.annotation.AuthenticationPrincipal;
import org.springframework.stereotype.Controller;
import org.springframework.validation.BindingResult;
import org.springframework.validation.annotation.Validated;
import org.springframework.web.bind.WebDataBinder;
import org.springframework.web.bind.annotation.InitBinder;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.SessionAttributes;
import org.springframework.web.bind.support.SessionStatus;
import org.springframework.web.servlet.mvc.support.RedirectAttributes;
import org.terasoluna.gfw.common.message.ResultMessages;

import com.example.session.app.account.AccountUpdateForm.Wizard1;
import com.example.session.app.account.AccountUpdateForm.Wizard2;
import com.example.session.domain.model.Account;
import com.example.session.domain.service.account.AccountService;
import com.example.session.domain.service.userdetails.AccountDetails;

@Controller
@SessionAttributes(value = { "accountUpdateForm" }) // (1)
@RequestMapping("account")
public class AccountUpdateController {
```

```
    @Inject
    AccountService accountService;

    @Inject
    Mapper beanMapper;

    @InitBinder
    public void initBinder(WebDataBinder binder) {
        binder.registerCustomEditor(String.class, new StringTrimmerEditor(true));
    }

    @ModelAttribute(value = "accountUpdateForm") // (2)
    public AccountUpdateForm setUpAccountForm() {
        return new AccountUpdateForm();
    }

    @RequestMapping(value = "update", params = "form1")
    public String showUpdateForm1(
            @AuthenticationPrincipal AccountDetails userDetails,
            AccountUpdateForm form) { // (3)

        Account account = accountService.findOne(userDetails.getAccount()
                .getEmail());
        beanMapper.map(account, form);

        return "account/updateForm1";
    }

    @RequestMapping(value = "update", params = "form2")
    public String showUpdateForm2(
            @Validated((Wizard1.class)) AccountUpdateForm form,
            BindingResult result) {

        if (result.hasErrors()) {
            return "account/updateForm1";
        }

        return "account/updateForm2";
    }

    @RequestMapping(value = "update", params = "redoForm1")
    public String redoUpdateForm1() {
        return "account/updateForm1";
    }

    @RequestMapping(value = "update", params = "confirm")
    public String confirmUpdate(
            @Validated(Wizard2.class) AccountUpdateForm form,
            BindingResult result) {
```

```
        if (result.hasErrors()) {
            return "account/updateForm2";
        }

        return "account/updateConfirm";
    }

    @RequestMapping(value = "update", params = "redoForm2")
    public String redoUpdateForm2() {
        return "account/updateForm2";
    }

    @RequestMapping(value = "update", method = RequestMethod.POST)
    public String update(
            @AuthenticationPrincipal AccountDetails userDetails,
            @Validated({ Wizard1.class, Wizard2.class }) AccountUpdateForm form,
            BindingResult result, RedirectAttributes attributes, SessionStatus␣
↪sessionStatus) {

        if (result.hasErrors()) {
            ResultMessages messages = ResultMessages.error();
            messages.add("e.st.ac.5001");
            throw new IllegalOperationException(messages);
        }

        Account account = beanMapper.map(form, Account.class);
        accountService.update(account);
        userDetails.setAccount(account);
        attributes.addFlashAttribute("account", account);
        sessionStatus.setComplete();  // (4)

        return "redirect:/account/update?finish";
    }

    @RequestMapping(value = "update", method = RequestMethod.GET, params = "finish
↪")
    public String finishUpdate() {
        return "account/updateFinish";
    }

    @RequestMapping(value = "update", method = RequestMethod.GET, params = "home")
    public String home(SessionStatus sessionStatus) {
        sessionStatus.setComplete();
        return "redirect:/goods";
    }

}
```

Sr. No.	Description
(1)	Specify the attribute name of the object to be stored in the session, in the value attribute of "@SessionAttributes" annotation. The object having the attribute name `"accountUpdateForm"` is stored in the session in the above example.
(2)	Specify the attribute name to be stored in the Model object in the value attribute. The returned object is stored in the session by the attribute name as `"accountUpdateForm"` in the above example. Since the "@ModelAttribute" annotated method can no longer be called by the request after the object is stored in the session when the value attribute is specified, it has a merit that unnecessary object is not generated.
(3)	In order to use the object managed by "@SessionAttributes" annotation, add argument in the method so that the object can be received. Use "@Validated" annotation when the input check is required. The object that contains `"accountUpdateForm"` that is the default attribute name of "AccountUpdateForm", in the attribute name is passed as an argument in the above example.
(4)	Call "setComplete" method of "SessionStatus" object and delete the object from the session.

Warning: The object managed using "@SessionAttributes" annotation continues to remain in the session unless it is explicitly deleted. Therefore, the data retained even when the transition has taken place outside the screen handled by the Controller and returned again, can be browsed. The data that is no longer required, should always be deleted to avoid the memory exhaustion.

Warning: When the user goes back using the browser button, inputs the URL directly and moves from one screen to another, it is required to note the point that "setComplete" method is not called and the session remains active without being cleared.

Creating JSP

Create a screen to pass data to the form object managed by "@SessionAttributes" annotation.

Input screen on the 1st page

"/session-tutorial-init-web/src/main/webapp/WEB-INF/views/account/updateForm1.jsp"

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8" />
<title>Account Update Page</title>
<link rel="stylesheet"
    href="${pageContext.request.contextPath}/resources/app/css/styles.css">
</head>
<body>

    <div class="container">
        <%-- (1) --%>
        <form:form action="${pageContext.request.contextPath}/account/update"
            method="post" modelAttribute="accountUpdateForm">

            <h2>Account Update Page 1/2</h2>
            <table>
                <tr>
                    <td><form:label path="name" cssErrorClass="error-label">name</
↪form:label></td>
                    <%-- (2) --%>
                    <td><form:input path="name" cssErrorClass="error-input" />
↪<form:errors
                            path="name" cssClass="error-messages" /></td>
                </tr>
                <tr>
                    <td><form:label path="email" cssErrorClass="error-label">e-mail
↪</form:label></td>
                    <td><form:input path="email" cssErrorClass="error-input" />
↪<form:errors
                            path="email" cssClass="error-messages" /></td>
                </tr>
                <tr>
                    <td><form:label path="birthday" cssErrorClass="error-label">
↪birthday</form:label></td>
                    <td><fmt:formatDate value="${accountUpdateForm.birthday}"
                            pattern="yyyy-MM-dd" var="formattedBirthday" /> <input
                        type="date" id="birthday" name="birthday"
                        value="${formattedBirthday}"> <form:errors path="birthday"
                            cssClass="error-messages" /></td>
                </tr>
                <tr>
                    <td><form:label path="zip" cssErrorClass="error-label">zip</
↪form:label></td>
```

```
                    <td><form:input path="zip" cssErrorClass="error-input" />
↪<form:errors
                                path="zip" cssClass="error-messages" /></td>
                </tr>
                <tr>
                    <td><form:label path="address" cssErrorClass="error-label">
↪address</form:label></td>
                    <td><form:input path="address" cssErrorClass="error-input" />
                        <form:errors path="address" cssClass="error-messages" /></
↪td>
                </tr>
                <tr>
                    <td> </td>
                    <td><input type="submit" name="form2" id="next" value="next" />
↪</td>
                </tr>
            </table>
        </form:form>

        <form method="get"
            action="${pageContext.request.contextPath}/account/update">
            <input type="submit" name="home" id="home" value="home" />
        </form>
    </div>
</body>
</html>
```

Sr. No.	Description
(1)	Specify the attribute name of the form object that receives the input data, in modelAttribute. The object with the attribute name "`accountUpdateForm`" receives the input data in the above example.
(2)	Specify the element name of the object that stores the input data in path attribute of form:input tag. If this method is used, when data already exists in the element name of the specified object, that value becomes as the default value of the input form.

Input screen on 2nd page

"/session-tutorial-init-web/src/main/webapp/WEB-INF/views/account/updateForm2.jsp"

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8" />
```

```
<title>Account Update Page</title>
<link rel="stylesheet"
    href="${pageContext.request.contextPath}/resources/app/css/styles.css">
</head>
<body>

    <div class="container">

        <form:form action="${pageContext.request.contextPath}/account/update"
            method="post" modelAttribute="accountUpdateForm">

            <h2>Account Update Page 2/2</h2>
            <table>
                <tr>
                    <td><form:label path="cardNumber" cssErrorClass="error-label">
↪your card number</form:label></td>
                    <td><form:input path="cardNumber" cssErrorClass="error-input" /
↪>
                        <form:errors path="cardNumber" cssClass="error-messages" />
↪</td>
                </tr>
                <tr>
                    <td><form:label path="cardExpirationDate"
                            cssErrorClass="error-label">expiration date of
                            your card</form:label></td>
                    <td><fmt:formatDate
                            value="${accountUpdateForm.cardExpirationDate}"␣
↪pattern="yyyy-MM"
                            var="formattedCardExpirationDate" /><input type="month"
                        name="cardExpirationDate" id="cardExpirationDate"
                        value="${formattedCardExpirationDate}"> <form:errors
                            path="cardExpirationDate" cssClass="error-messages" />
↪</td>
                </tr>
                <tr>
                    <td><form:label path="cardSecurityCode"
                            cssErrorClass="error-label">security code of
                            your card</form:label></td>
                    <td><form:input path="cardSecurityCode"
                            cssErrorClass="error-input" /> <form:errors
                            path="cardSecurityCode" cssClass="error-messages" /></
↪td>
                </tr>
                <tr>
                    <td> </td>
                    <td><input type="submit" name="redoForm1" id="back"
                        value="back" /><input type="submit" name="confirm" id=
↪"confirm"
                            value="confirm" /></td>
                </tr>
```

```
            </table>
        </form:form>

        <form method="get"
            action="${pageContext.request.contextPath}/account/update">
            <input type="submit" name="home" id="home" value="home" />
        </form>
    </div>
</body>
</html>
```

Confirmation screen

"/session-tutorial-init-web/src/main/webapp/WEB-INF/views/account/updateConfirm.jsp"

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8" />
<title>Account Update Page</title>
<link rel="stylesheet"
    href="${pageContext.request.contextPath}/resources/app/css/styles.css">
</head>
<body>
    <div class="container">

        <form:form action="${pageContext.request.contextPath}/account/update"
            method="post">

            <h3>Your account will be updated with below information. Please
                push "update" button if it's OK.</h3>
            <table>
                <tr>
                    <td><label for="name">name</label></td>
                    <td><span id="name">${f:h(accountUpdateForm.name)}</span></td>
                </tr>
                <tr>
                    <td><label for="email">e-mail</label></td>
                    <td><span id="email">${f:h(accountUpdateForm.email)}</span></
↪td>
                </tr>
                <tr>
                    <td><label for="birthday">birthday</label></td>
                    <td><span id="birthday"><fmt:formatDate
                                value="${accountUpdateForm.birthday}" pattern=
↪"yyyy-MM-dd" /></span></td>
                </tr>
                <tr>
                    <td><label for="zip">zip</label></td>
                    <td><span id="zip">${f:h(accountUpdateForm.zip)}</span></td>
```

```
                </tr>
                <tr>
                    <td><label for="address">address</label></td>
                    <td><span id="address">${f:h(accountUpdateForm.address)}</span>
↪</td>
                </tr>
                <tr>
                    <td><label for="cardNumber">your card number</label></td>
                    <td><span id="cardNumber">****-****-****-${f:
↪h(accountUpdateForm.lastFourOfCardNumber)}</span></td>
                </tr>
                <tr>
                    <td><label for="cardExpirationDate">expiration date of
                            your card</label></td>
                    <td><span id="cardExpirationDate"><fmt:formatDate
                                value="${accountUpdateForm.cardExpirationDate}"
                                pattern="yyyy-MM" /></span></td>
                </tr>
                <tr>
                    <td><label for="cardSecurityCode">security code of
                            your card</label></td>
                    <td><span id="cardSecurityCode">${f:h(accountUpdateForm.
↪cardSecurityCode)}</span></td>
                </tr>
                <tr>
                    <td> </td>
                    <td><input type="submit" name="redoForm2" id="back"
                        value="back" /><input type="submit" id="update" value=
↪"update" /></td>
                </tr>
            </table>
        </form:form>


        <form method="get"
            action="${pageContext.request.contextPath}/account/update">
            <input type="submit" name="home" id="home" value="home" />
        </form>
    </div>
</body>
</html>
```

Completion screen

“/session-tutorial-init-web/src/main/webapp/WEB-INF/views/account/updateFinish.jsp”

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8" />
```

```
<title>Account Update Page</title>
<link rel="stylesheet"
    href="${pageContext.request.contextPath}/resources/app/css/styles.css">
</head>
<body>
    <div class="container">

        <h3>Your account has updated.</h3>
        <table>
            <tr>
                <td><label for="name">name</label></td>
                <td><span id="name">${f:h(account.name)}</span></td>
            </tr>
            <tr>
                <td><label for="email">e-mail</label></td>
                <td><span id="email">${f:h(account.email)}</span></td>
            </tr>
            <tr>
                <td><label for="birthday">birthday</label></td>
                <td><span id="birthday"><fmt:formatDate
                            value="${account.birthday}" pattern="yyyy-MM-dd" /></
↪span></td>
            </tr>
            <tr>
                <td><label for="zip">zip</label></td>
                <td><span id="zip">${f:h(account.zip)}</span></td>
            </tr>
            <tr>
                <td><label for="address">address</label></td>
                <td><span id="address">${f:h(account.address)}</span></td>
            </tr>
            <tr>
                <td><label for="cardNumber">your card number</label></td>
                <td><span id="cardNumber">****-****-****-${f:h(account.
↪lastFourOfCardNumber)}</span></td>
            </tr>
            <tr>
                <td><label for="cardExpirationDate">expiration date of
                        your card</label></td>
                <td><span id="cardExpirationDate"><fmt:formatDate
                            value="${account.cardExpirationDate}" pattern="yyyy-MM
↪" /></span></td>
            </tr>
            <tr>
                <td><label for="cardSecurityCode">security code of your
                        card</label></td>
                <td><span id="cardSecurityCode">${f:h(account.cardSecurityCode)}</
↪span></td>
            </tr>
        </table>
```

```
        <form method="get"
            action="${pageContext.request.contextPath}/account/update">
            <input type="submit" name="home" id="home" value="home" />
        </form>

    </div>
</body>
</html>
```

Checking operation

The account information can be updated using the implementation so far. The transition takes place to the "Update account information" screen by clicking 'Account Update' button in the upper part of the product list display screen. At present, the information of account to which the user is logged in, is displayed in the form as the initial value. Finally the account information is updated when the user changes the form value and proceeds to the next screen.

Since the form that receives the input value is stored in the session with the implementation so far, passing of data can be easily implemented. Further, the changed information is reset if the transition takes place to the Update account information screen after clicking 'home' button since the session is discarded when 'home' button is clicked.

Create cart item registration function

Create a function to register the products for a spe cified quantity in the cart.

Manage the cart information as a Bean for session scope as explained in *Application design*.

The information of screen implemented in cart item registration function is shown below.

Process name	HTTP method	Path	Screen
Process to add a product to the cart	POST	/addToCart	Redirect to Product list screen display process

Defining the Bean for session scope

Object to retain the cart information is already created as "Cart.java". Therefore, add the settings so as to enable handling this object as the Bean for session scope.

There are 2 types of configuration methods mentioned in *Session Management* as the methods to use the Bean for session scope. Define the Bean using component-scan in this tutorial.

Warning: The target object needs to be 'Serializable' to register as the Bean for session scope

To define the Bean for session scope using component-scan, the following annotation should be added to the class to be registered as Bean.

"/session-tutorial-init-domain/src/main/java/com/example/session/domain/model/Cart.java"

```
package com.example.session.domain.model;

import java.io.Serializable;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.Collection;
import java.util.LinkedHashMap;
import java.util.Map;
import java.util.Set;

import org.springframework.context.annotation.Scope;
import org.springframework.context.annotation.ScopedProxyMode;
import org.springframework.security.crypto.codec.Base64;
import org.springframework.stereotype.Component;
import org.springframework.util.SerializationUtils;

@Component // (1)
@Scope(value = "session", proxyMode = ScopedProxyMode.TARGET_CLASS) // (2)
public class Cart implements Serializable {

    // omitted

}
```

Sr. No.	Description
(1)	Specify "@Component" annotation so that it becomes the target for component-scan.
(2)	Set Bean scope to `"session"`. Further, specify `"ScopedProxyMode.TARGET_CLASS"` using proxyMode attribute and enable scoped-proxy.

Further, the base-package that is a target for component-scan, needs to be specified in the Bean definition file. However, since the following is already mentioned in the Bean definition file created in this tutorial, it is not required to add a new description.

"/session-tutorial-init-domain/src/main/resources/META-INF/spring/session-tutorial-init-domain.xml"

```
<!-- (1) -->
<context:component-scan base-package="com.example.session.domain" />
```

Sr. No.	Description
(1)	Specify the package used as a target for component-scan.

Creating the form object

Create a class to retain the information of ordered products.

"/session-tutorial-init-web/src/main/java/com/example/session/app/goods/GoodAddForm.java"

```
package com.example.session.app.goods;

import java.io.Serializable;

import javax.validation.constraints.Min;
import javax.validation.constraints.NotNull;

public class GoodAddForm implements Serializable {

    /**
     *
     */
    private static final long serialVersionUID = 1L;

    @NotNull
    private String goodsId;

    @NotNull
    @Min(1)
    private int quantity;

    public String getGoodsId() {
        return goodsId;
    }

    public void setGoodsId(String goodsId) {
        this.goodsId = goodsId;
    }

    public int getQuantity() {
        return quantity;
    }

    public void setQuantity(int quantity) {
```

```
        this.quantity = quantity;
    }
}
```

Creating Controller

Create the Controller.

Since it is already created to process the partial request, the following code is added.

"/session-tutorial-init-web/src/main/java/com/example/session/app/goods/GoodsController.java"

```
package com.example.session.app.goods;

import javax.inject.Inject;

import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.BindingResult;
import org.springframework.validation.annotation.Validated;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.servlet.mvc.support.RedirectAttributes;
import org.terasoluna.gfw.common.message.ResultMessages;

import com.example.session.domain.model.Cart;
import com.example.session.domain.model.CartItem;
import com.example.session.domain.model.Goods;
import com.example.session.domain.service.goods.GoodsService;

@Controller
@RequestMapping("goods")
public class GoodsController {

    @Inject
    GoodsService goodsService;

    // (1)
    @Inject
    Cart cart;

    @ModelAttribute(value = "goodViewForm")
    public GoodViewForm setUpCategoryId() {
        return new GoodViewForm();
    }
```

```
    @RequestMapping(value = "", method = RequestMethod.GET)
    String showGoods(GoodViewForm form, Pageable pageable, Model model) {

        Page<Goods> page = goodsService.findByCategoryId(form.getCategoryId(),
                pageable);
        model.addAttribute("page", page);
        return "goods/showGoods";
    }

    @RequestMapping(value = "/{goodsId}", method = RequestMethod.GET)
    public String showGoodsDetail(@PathVariable String goodsId, Model model) {

        Goods goods = goodsService.findOne(goodsId);
        model.addAttribute(goods);

        return "/goods/showGoodsDetail";
    }

    @RequestMapping(value = "/addToCart", method = RequestMethod.POST)
    public String addToCart(@Validated GoodAddForm form, BindingResult result,
            RedirectAttributes attributes) {

        if (result.hasErrors()) {
            ResultMessages messages = ResultMessages.error()
                    .add("e.st.go.5001");
            attributes.addFlashAttribute(messages);
            return "redirect:/goods";
        }

        Goods goods = goodsService.findOne(form.getGoodsId());
        CartItem cartItem = new CartItem();
        cartItem.setGoods(goods);
        cartItem.setQuantity(form.getQuantity());
        cart.add(cartItem); // (2)

        return "redirect:/goods";
    }
}
```

Sr. No.	Description
(1)	Fetch the Bean for session scope from DI container.
(2)	Add the data in the Bean for session scope. It is not necessary to add a object to Model to display the information in the screen.

Creating JSP

Create JSP to display the contents of the cart.

Since JSP is also already created, the code shown below is added at the end of body tag.

"/session-tutorial-init-web/src/main/webapp/WEB-INF/views/goods/showGoods.jsp"

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8" />
<title>Item List Page</title>
<link rel="stylesheet"
    href="${pageContext.request.contextPath}/resources/app/css/styles.css">
<link rel="stylesheet"
    href="${pageContext.request.contextPath}/resources/vendor/bootstrap-3.0.0/css/
↪bootstrap.css"
    type="text/css" media="screen, projection">
</head>
<body>

    <sec:authentication property="principal" var="userDetails" />
    <div style="display: inline-flex">
        welcome   <span id="userName">${f:h(userDetails.account.name)}</
↪span>
        <form method="post" action="${pageContext.request.contextPath}/logout">
            <input type="submit" id="logout" value="logout" />
            <sec:csrfInput />
        </form>
        <form method="get"
            action="${pageContext.request.contextPath}/account/update">
            <input type="submit" name="form1" id="updateAccount"
                value="Account Update" />
        </form>
    </div>
    <br>
    <br>
```

```
    <div class="container">
        <p>select a category</p>

        <form:form method="get"
            action="${pageContext.request.contextPath}/goods/"
            modelAttribute="goodViewForm">
            <form:select path="categoryId" items="${CL_CATEGORIES}" />
            <input type="submit" id="update" value="update" />
        </form:form>
        <br />
        <t:messagesPanel />
        <table>
            <tr>
                <th>Name</th>
                <th>Price</th>
                <th>Quantity</th>
            </tr>
            <c:forEach items="${page.content}" var="goods" varStatus="status">
                <tr>
                    <td><a id="${f:h(goods.name)}"
                        href="${pageContext.request.contextPath}/goods/${f:h(goods.
↪id)}">${f:h(goods.name)}</a></td>
                    <td><fmt:formatNumber value="${f:h(goods.price)}"
                            type="CURRENCY" currencySymbol="&yen;"␣
↪maxFractionDigits="0" /></td>
                    <td><form:form method="post"
                            action="${pageContext.request.contextPath}/goods/
↪addToCart"
                            modelAttribute="goodAddForm">
                            <input type="text" name="quantity" id="quantity$
↪{status.index}" value="1" />
                            <input type="hidden" name="goodsId" value="${f:h(goods.
↪id)}" />
                            <input type="submit" id="add${status.index}" value="add
↪" />
                        </form:form></td>
                </tr>
            </c:forEach>
        </table>
        <t:pagination page="${page}" outerElementClass="pagination" />
    </div>
    <div>
        <p>
            <fmt:formatNumber value="${page.totalElements}" />
            results <br> ${f:h(page.number + 1) } / ${f:h(page.totalPages)}
            Pages
        </p>
    </div>
    <div>
```

```
        <%-- (1) --%>
        <spring:eval var="cart" expression="@cart" />
        <form method="get" action="${pageContext.request.contextPath}/cart">
            <input type="submit" id="viewCart" value="view cart" />
        </form>
        <table>
            <%-- (2) --%>
            <c:forEach items="${cart.cartItems}" var="cartItem" varStatus="status">
                <tr>
                    <td><span id="itemName${status.index}">${f:h(cartItem.goods.
↪name)}</span></td>
                    <td><span id="itemPrice${status.index}"><fmt:formatNumber
                                value="${cartItem.goods.price}" type="CURRENCY"
                                currencySymbol="&yen;" maxFractionDigits="0" /></
↪span></td>
                    <td><span id="itemQuantity${status.index}">${f:h(cartItem.
↪quantity)}</span></td>
                </tr>
            </c:forEach>
            <tr>
                <td>Total</td>
                <td><span id="totalPrice"><fmt:formatNumber
                            value="${f:h(cart.totalAmount)}" type="CURRENCY"
                            currencySymbol="&yen;" maxFractionDigits="0" /></span>
↪</td>
                <td></td>
            </tr>
        </table>
    </div>

</body>
</html>
```

Sr. No.	Description
(1)	Store the Bean in a variable in order to display the contents of the Bean for session scope in a screen. Cart object in session scope is stored in variable “cart” in the above example.
(2)	Browse the contents of the Bean for session scope through the variable created in (1). The contents of the Bean for session scope are browsed through variable “var” in the above example.

Note: var attribute is not required if simply the contents of Bean are to be displayed alone without storing them in a variable. It can be displayed using “<spring:eval expression=”@cart” />” in the above example.

“/session-tutorial-init-web/src/main/webapp/WEB-INF/views/goods/showGoodsDetail.jsp”

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8" />
<title>Item List Page</title>
<link rel="stylesheet"
    href="${pageContext.request.contextPath}/resources/app/css/styles.css">
</head>
<body>

    <sec:authentication property="principal" var="userDetails" />
    <div style="display: inline-flex">
        welcome   <span id="userName">${f:h(userDetails.account.name)}</
↪span>
        <form:form method="post"
            action="${pageContext.request.contextPath}/logout">
            <input type="submit" id="logout" value="logout" />
        </form:form>
        <form method="get"
            action="${pageContext.request.contextPath}/account/update">
            <input type="submit" name="form1" id="updateAccount"
                value="Account Update" />
        </form>
    </div>
    <br>
    <br>

    <div class="container">

        <table>
            <tr>
                <th>Name</th>
                <td>${f:h(goods.name)}</td>
            </tr>
            <tr>
                <th>Price</th>
                <td><fmt:formatNumber value="${f:h(goods.price)}"
                        type="CURRENCY" currencySymbol="&yen;" maxFractionDigits="0
↪" /></td>
            </tr>
            <tr>
                <th>Description</th>
                <td>${f:h(goods.description)}</td>
            </tr>
        </table>
        <form:form method="post"
            action="${pageContext.request.contextPath}/goods/addToCart"
            modelAttribute="AddToCartForm">
            Quantity<input type="text" id="quantity" name="quantity"
```

```
                value="1" />
            <input type="hidden" name="goodsId" value="${f:h(goods.id)}" />
            <input type="submit" id="add" value="add" />
        </form:form>

        <form method="get" action="${pageContext.request.contextPath}/goods">
            <input type="submit" id="home" value="home" />
        </form>
    </div>
    <div>
        <spring:eval var="cart" expression="@cart" />
        <form method="get" action="${pageContext.request.contextPath}/cart">
            <input type="submit" value="view cart" />
        </form>
        <table>
            <c:forEach items="${cart.cartItems}" var="cartItem"
                varStatus="status">
                <tr>
                    <td><span id="itemName${status.index}">${f:h(cartItem.goods.
↪name)}</span></td>
                    <td><span id="itemPrice${status.index}"><fmt:formatNumber
                                value="${cartItem.goods.price}" type="CURRENCY"
                                currencySymbol="&yen;" maxFractionDigits="0" /></
↪span></td>
                    <td><span id="itemQuantity${status.index}">${f:h(cartItem.
↪quantity)}</span></td>
                </tr>
            </c:forEach>
            <tr>
                <td>Total</td>
                <td><span id="totalPrice"><fmt:formatNumber
                            value="${f:h(cart.totalAmount)}" type="CURRENCY"
                            currencySymbol="&yen;" maxFractionDigits="0" /></span>
↪</td>
                <td></td>
            </tr>
        </table>
    </div>
</body>
</html>
```

Checking operation

Products can be registered to the cart with the implementation so far. Contents of the cart on the same page are displayed by clicking 'add' button for a product on the product list display screen.

Since the cart object is stored in the session with the implementation so far, cart information is saved even though moved to the account information update screen and returned.

Create a mechanism to retain the product search information

Products can be added to the cart with the implementation so far. However, screen where the transition after adding products takes place, is usually the 1st page of 'book' category.

This tutorial has a specification to retain the product search information including the selection category and page number till the order is completed. Therefore, modify the implementation so as to transit to the previous status after adding products or when returned from the account update screen.

Manage the product search information as the Bean for session scope as explained in *Application design*.

Information of screen to be modified is shown below.

Process name	HTTP method	Path	Screen
Product list screen display process (default)	GET	/goods **(created)**	/goods/showGoods
Product list screen display process (while selecting category)	GET	/goods?categoryId **(created)**	/goods/showGoods
Product list screen display process (while selecting page)	GET	/goods?page **(created)**	/goods/showGoods

Create session scope Bean

Create the session scope Bean to retain the product search information. Define a bean using component-scan similar to the cart information.

"/session-tutorial-init-web/src/main/java/com/example/session/app/goods/GoodsSearchCriteria.java"

```
package com.example.session.app.goods;

import java.io.Serializable;

import org.springframework.context.annotation.Scope;
import org.springframework.context.annotation.ScopedProxyMode;
import org.springframework.stereotype.Component;

@Component // (1)
@Scope(value = "session", proxyMode = ScopedProxyMode.TARGET_CLASS) // (2)
```

```
public class GoodsSearchCriteria implements Serializable {

    /**
     *
     */
    private static final long serialVersionUID = 1L;

    private int categoryId = 1;

    private int page = 0;

    public int getCategoryId() {
        return categoryId;
    }

    public void setCategoryId(int categoryId) {
        this.categoryId = categoryId;
    }

    public int getPage() {
        return page;
    }

    public void setPage(int page) {
        this.page = page;
    }

    public void clear() {
        categoryId = 1;
        page = 0;
    }

}
```

Sr. No.	Description
(1)	Specify “@Component” annotation so that it becomes the target for component-scan
(2)	Set scope of Bean as `"session"`. Further, specify `"ScopedProxyMode.TARGET_CLASS"` in proxyMode attribute and enable scoped-proxy.

Further, it is required to specify base-package that is a target for component-scan in Bean definition file. However, since the following is already mentioned in the Bean definition file created in this tutorial, it is not required to add a new description.

“/session-tutorial-init-web/src/main/resources/META-INF/spring/spring-mvc.xml”

```
<!-- (1) -->
<context:component-scan base-package="com.example.session.app" />
```

Sr. No.	Description
(1)	Specify the package that is a target for component-scan.

Modifying Controller

Modify the Controller so as to retain the product search information in the session and to use the product search information retained in the session.

"/session-tutorial-init-web/src/main/java/com/example/session/app/goods/GoodsController.java"

```
package com.example.session.app.goods;

import javax.inject.Inject;

import org.springframework.data.domain.Page;
import org.springframework.data.domain.PageRequest;
import org.springframework.data.domain.Pageable;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.BindingResult;
import org.springframework.validation.annotation.Validated;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.servlet.mvc.support.RedirectAttributes;
import org.terasoluna.gfw.common.message.ResultMessages;

import com.example.session.domain.model.Cart;
import com.example.session.domain.model.CartItem;
import com.example.session.domain.model.Goods;
import com.example.session.domain.service.goods.GoodsService;

@Controller
@RequestMapping("goods")
public class GoodsController {

    @Inject
    GoodsService goodsService;

    @Inject
    Cart cart;
```

```
// (1)
@Inject
GoodsSearchCriteria criteria;

@ModelAttribute(value = "goodViewForm")
public GoodViewForm setUpCategoryId() {
    return new GoodViewForm();
}

// (2)
@RequestMapping(value = "", method = RequestMethod.GET)
String showGoods(GoodViewForm form, Model model) {
    Pageable pageable = new PageRequest(criteria.getPage(), 3);
    form.setCategoryId(criteria.getCategoryId());
    return showGoods(pageable, model);
}

// (3)
@RequestMapping(value = "", method = RequestMethod.GET, params = "categoryId")
String changeCategoryId(GoodViewForm form, Pageable pageable, Model model) {
    criteria.setPage(pageable.getPageNumber());
    criteria.setCategoryId(form.getCategoryId());
    return showGoods(pageable, model);
}

// (4)
@RequestMapping(value = "", method = RequestMethod.GET, params = "page")
String changePage(GoodViewForm form, Pageable pageable, Model model) {
    criteria.setPage(pageable.getPageNumber());
    form.setCategoryId(criteria.getCategoryId());
    return showGoods(pageable, model);
}

// (5)
String showGoods(Pageable pageable, Model model) {
    Page<Goods> page = goodsService.findByCategoryId(
            criteria.getCategoryId(), pageable);
    model.addAttribute("page", page);
    return "goods/showGoods";
}

@RequestMapping(value = "/{goodsId}", method = RequestMethod.GET)
public String showGoodsDetail(@PathVariable String goodsId, Model model) {

    Goods goods = goodsService.findOne(goodsId);
    model.addAttribute(goods);

    return "/goods/showGoodsDetail";
}
```

```
    @RequestMapping(value = "/addToCart", method = RequestMethod.POST)
    public String addToCart(@Validated GoodAddForm form, BindingResult result,
            RedirectAttributes attributes) {

        if (result.hasErrors()) {
            ResultMessages messages = ResultMessages.error()
                    .add("e.st.go.5001");
            attributes.addFlashAttribute(messages);
            return "redirect:/goods";
        }

        Goods goods = goodsService.findOne(form.getGoodsId());
        CartItem cartItem = new CartItem();
        cartItem.setGoods(goods);
        cartItem.setQuantity(form.getQuantity());
        cart.add(cartItem);

        return "redirect:/goods";
    }
}
```

Sr. No.	Description
(1)	Fetch the Bean for session scope from DI container.
(2)	Perform the pre-processing of the usual product list screen display process. Set the product category stored in the session to the form and the page number to “pageable”. The product category is set to the form in order to specify the product category displayed using the select box.
(3)	Perform the pre-processing of the product list screen display process when the category is changed. Store the product category entered in the session. Specify the default 1st page of the page numbers in “pageable”.
(4)	Perform the pre-processing of the product list screen display process when the page is changed. Store the page number entered in the session. Set the product category stored in the session in the form.
(5)	Handle the common portion. Search a product based on product category managed in session and “pageable” fetched in the preprocessing.

Checking operation

Product search information can be retained with the implementation so far. For example, when a product is added to the cart on the 2nd page of 'music' category, the destination for transition remains the original 2nd page of 'music' category. Further, when 'Account Update' button is clicked on the same screen, moved to the account update screen, 'home' button on the account update screen is clicked and returned, the destination for transition is just the original 2nd page of 'music' category.

Create cart item deletion function

Create a function to delete the specified product from the cart.

Use checkbox to specify the product to be deleted.

The information of screen implemented using the cart item deletion function is shown below.

Process name	HTTP method	Path	Screen
Cart screen display process	GET	/cart	cart/viewCart
Process to delete a product from the cart	POST	/cart	Redirect to cart screen display process

Creating form object

Create a class to retain ID of the product to be deleted.

"/session-tutorial-init-web/src/main/java/com/example/session/app/cart/CartForm.java"

```
package com.example.session.app.cart;

import java.util.Set;

import org.hibernate.validator.constraints.NotEmpty;

public class CartForm {

    @NotEmpty
    private Set<String> removedItemsIds;

    public Set<String> getRemovedItemsIds() {
        return removedItemsIds;
    }
```

```
    public void setRemovedItemsIds(Set<String> removedItemsIds) {
        this.removedItemsIds = removedItemsIds;
    }
}
```

Creating Controller

Create Controller.

"/session-tutorial-init-web/src/main/java/com/example/session/app/cart/CartController.java"

```
package com.example.session.app.cart;

import javax.inject.Inject;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.BindingResult;
import org.springframework.validation.annotation.Validated;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.terasoluna.gfw.common.message.ResultMessages;

import com.example.session.domain.model.Cart;

@Controller
@RequestMapping("cart")
public class CartController {

    // (1)
    @Inject
    Cart cart;

    @ModelAttribute
    CartForm setUpForm() {
        return new CartForm();
    }

    @RequestMapping(method = RequestMethod.GET)
    String viewCart(Model model) {
        return "cart/viewCart";
    }

    @RequestMapping(method = RequestMethod.POST)
    String removeFromCart(@Validated CartForm cartForm,
            BindingResult bindingResult, Model model) {
        if (bindingResult.hasErrors()) {
```

```
            ResultMessages messages = ResultMessages.error()
                    .add("e.st.ca.5001");
            model.addAttribute(messages);
            return viewCart(model);
        }
        cart.remove(cartForm.getRemovedItemsIds()); // (2)
        return "redirect:/cart";
    }
}
```

Sr. No.	Description
(1)	Fetch the Bean for session scope from DI container.
(2)	Delete the data in the Bean for session scope.

Creating JSP

Display the cart list and create JSP to select the product to be deleted. Products can be ordered from this screen.

"/session-tutorial-init-web/src/main/webapp/WEB-INF/views/cart/viewCart.jsp"

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8" />
<title>View Cart Page</title>
<link rel="stylesheet"
    href="${pageContext.request.contextPath}/resources/app/css/styles.css">
</head>
<body>

    <sec:authentication property="principal" var="userDetails" />

    <div style="display: inline-flex">
        welcome ${f:h(userDetails.account.name)}
        <form:form method="post"
            action="${pageContext.request.contextPath}/logout">
            <input type="submit" id="logout" value="logout" />
        </form:form>
        <form method="get"
            action="${pageContext.request.contextPath}/account/update">
            <input type="submit" name="form1" id="updateAccount"
                value="Account Update" />
        </form>
```

```
    </div>
    <br>
    <br>

    <div>
        <spring:eval var="cart" expression="@cart" />
        <form:form method="post"
            action="${pageContext.request.contextPath}/cart"
            modelAttribute="cartForm">
            <form:errors path="removedItemsIds" cssClass="error-messages" />
            <t:messagesPanel />
            <table>
                <tr>
                    <th>Name</th>
                    <th>Price</th>
                    <th>Quantity</th>
                    <th>Remove</th>
                </tr>
                <c:forEach items="${cart.cartItems}" var="cartItem"
                    varStatus="status">
                    <tr>
                        <td><span id="itemName${status.index}">${f:h(cartItem.
↪goods.name)}</span></td>
                        <td><span id="itemPrice${status.index}"><fmt:formatNumber
                                    value="${cartItem.goods.price}" type="CURRENCY"
                                    currencySymbol="&yen;" maxFractionDigits="0" />
↪</span></td>
                        <td><span id="itemQuantity${status.index}">${f:h(cartItem.
↪quantity)}</span></td>
                        <%-- (1) --%>
                        <td><input type="checkbox" name="removedItemsIds"
                            id="removedItemsIds${status.index}"
                            value="${f:h(cartItem.goods.id)}" /></td>
                    </tr>
                </c:forEach>
                <tr>
                    <td>Total</td>
                    <td><span id="totalPrice"><fmt:formatNumber
                                value="${f:h(cart.totalAmount)}" type="CURRENCY"
                                currencySymbol="&yen;" maxFractionDigits="0" /></
↪span></td>
                    <td></td>
                    <td></td>
                </tr>
            </table>
            <input type="submit" id="remove" value="remove" />
        </form:form>
    </div>

    <div style="display: inline-flex">
```

```
        <form method="get" action="${pageContext.request.contextPath}/order">
            <input type="submit" id="confirm" name="confirm"
                value="confirm your order" />
        </form>
        <form method="get" action="${pageContext.request.contextPath}/goods">
            <input type="submit" id="home" value="home" />
        </form>
    </div>
</body>
</html>
```

Sr. No.	Description
(1)	Specify the product to be deleted using a checkbox. When Delete button is clicked with the checkbox selected, ID of the corresponding product is sent to the server.

Checking operation

Products registered in the cart can be deleted with the implementation so far. Transition to the cart display screen takes place by clicking 'viewCart' button on the product list display screen. Product can be deleted from the cart by checking the product to be deleted on the cart display screen and clicking 'remove' button.

Create the product order function

Create a function to order the products registered in the cart.

Contents of the cart become blank after the order completion.

Information of the screen implemented in the product order function is shown below.

Process name	HTTP method	Path	Screen
Order confirmation screen display process	GET	/order?confirm	order/confirm
Order processing	POST	/order	Redirect to order completion screen display process
Order completion screen display process	GET	/order?finish	order/finish

Creating Controller

Create the Controller.

"/session-tutorial-init-web/src/main/java/com/example/session/app/order/OrderController.java"

```
package com.example.session.app.order;

import javax.inject.Inject;

import org.springframework.http.HttpStatus;
import org.springframework.security.core.annotation.AuthenticationPrincipal;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.ResponseStatus;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.mvc.support.RedirectAttributes;
import org.terasoluna.gfw.common.exception.BusinessException;
import org.terasoluna.gfw.common.message.ResultMessages;

import com.example.session.app.goods.GoodsSearchCriteria;
import com.example.session.domain.model.Cart;
import com.example.session.domain.model.Order;
import com.example.session.domain.service.order.EmptyCartOrderException;
import com.example.session.domain.service.order.InvalidCartOrderException;
import com.example.session.domain.service.order.OrderService;
import com.example.session.domain.service.userdetails.AccountDetails;
```

```
@Controller
@RequestMapping("order")
public class OrderController {

    @Inject
    OrderService orderService;

    // (1)
    @Inject
    Cart cart;

    @Inject
    GoodsSearchCriteria criteria;

    @RequestMapping(method = RequestMethod.GET, params = "confirm")
    String confirm(@AuthenticationPrincipal AccountDetails userDetails,
            Model model) {
        if (cart.isEmpty()) {
            ResultMessages messages = ResultMessages.error()
                    .add("e.st.od.5001");
            model.addAttribute(messages);
            return "cart/viewCart";
        }
        model.addAttribute("account", userDetails.getAccount());
        model.addAttribute("signature", cart.calcSignature());
        return "order/confirm";
    }

    @RequestMapping(method = RequestMethod.POST)
    String order(@AuthenticationPrincipal AccountDetails userDetails,
            @RequestParam String signature, RedirectAttributes attributes) {
        Order order = orderService.purchase(userDetails.getAccount(), cart,
                signature); // (2)
        attributes.addFlashAttribute(order);
        criteria.clear(); // (3)
        return "redirect:/order?finish";
    }

    @RequestMapping(method = RequestMethod.GET, params = "finish")
    String finish() {
        return "order/finish";
    }

    // (4)
    @ExceptionHandler({ EmptyCartOrderException.class,
            InvalidCartOrderException.class })
    @ResponseStatus(HttpStatus.CONFLICT)
    ModelAndView handleOrderException(BusinessException e) {
        return new ModelAndView("common/error/businessError").addObject(e
```

```
                .getResultMessages());
    }
}
```

Sr. No.	Description
(1)	Fetch the Bean for session scope from DI container.
(2)	The contents of the Bean for session scope are made empty using the method of Service in the domain layer. Accordingly, Bean for session scope is discarded. Further, information in the Bean for session scope is used by the screen where the user transits after discarding Bean in this application. Therefore, information that existed in the Bean for session scope is re-entered in a different object and added to the Flash scope.
(3)	Product search information is returned to the default status.
(4)	Since a Business exception may occur in the Service method, error handling is performed in this method. Because of this, user transits to the specified error screen when a Business exception occurs.

Warning: Method to discard the Bean for session scope is different from the method to discard the object managed by @SessionAttributes. Discarding the Bean for session scope should be assigned to DI container and should not be discarded by the application. Therefore, fields in the Bean for session scope just need to be reset to discard the Bean for session scope. Bean itself is discarded at the time of session timeout or logout.

Creating JSP

Create JSP to display order details and payment information.

"/session-tutorial-init-web/src/main/webapp/WEB-INF/views/order/confirm.jsp"

```
<!DOCTYPE html>
<html>
```

```
<head>
<meta charset="UTF-8" />
<title>Order Page</title>
<link rel="stylesheet"
    href="${pageContext.request.contextPath}/resources/app/css/styles.css">
</head>
<body>

    <sec:authentication property="principal" var="userDetails" />

    <div style="display: inline-flex">
        welcome ${f:h(userDetails.account.name)}
        <form:form method="post"
            action="${pageContext.request.contextPath}/logout">
            <input type="submit" id="logout" value="logout" />
        </form:form>
        <form method="get"
            action="${pageContext.request.contextPath}/account/update">
            <input type="submit" name="form1" id="updateAccount"
                value="Account Update" />
        </form>
    </div>
    <br>
    <br>

    <div>
        <spring:eval var="cart" expression="@cart" />

        <h3>Below items will be ordered. Please push "order" button if
            it's OK.</h3>
        <table>
            <tr>
                <th>Name</th>
                <th>Price</th>
                <th>Quantity</th>
            </tr>
            <c:forEach items="${cart.cartItems}" var="cartItem"
                varStatus="status">
                <tr>
                    <td><span id="itemName${status.index}">${f:h(cartItem.goods.
↪name)}</span></td>
                    <td><span id="itemPrice${status.index}"><fmt:formatNumber
                                value="${cartItem.goods.price}" type="CURRENCY"
                                currencySymbol="&yen;" maxFractionDigits="0" /></
↪span></td>
                    <td><span id="itemQuantity${status.index}">${f:h(cartItem.
↪quantity)}</span></td>
                </tr>
            </c:forEach>
            <tr>
```

```
                <td>Total</td>
                <td><span id="totalPrice"><fmt:formatNumber
                            value="${f:h(cart.totalAmount)}" type="CURRENCY"
                            currencySymbol="&yen;" maxFractionDigits="0" /></span>
↪</td>
                <td></td>
            </tr>
        </table>

        <table>
            <tr>
                <td><label for="name">name</label></td>
                <td><span id="name">${f:h(account.name)}</span></td>
            </tr>
            <tr>
                <td><label for="email">e-mail</label></td>
                <td><span id="email">${f:h(account.email)}</span></td>
            </tr>
            <tr>
                <td><label for="zip">zip</label></td>
                <td><span id="zip">${f:h(account.zip)}</span></td>
            </tr>
            <tr>
                <td><label for="address">address</label></td>
                <td><span id="address">${f:h(account.address)}</span></td>
            </tr>
            <tr>
                <%-- (1) --%>
                <td>payment</td>
                <td><span id="payment"><c:choose>
                            <c:when test="${empty account.cardNumber}">
                            cash
                        </c:when>
                            <c:otherwise>
                            card (card number : ****-****-****-${f:h(account.
↪lastFourOfCardNumber)})
                        </c:otherwise>
                        </c:choose></span></td>
            </tr>
        </table>
    </div>
    <div style="display: inline-flex">
        <form:form method="post"
            action="${pageContext.request.contextPath}/order">
            <input type="hidden" name="signature" value="${f:h(signature)}" />
            <input type="submit" id="order" value="order" />
        </form:form>
        <form method="get" action="${pageContext.request.contextPath}/cart">
            <input type="submit" id="back" value="back" />
        </form>
```

```
    </div>
    <div>
        <form method="get" action="${pageContext.request.contextPath}/goods">
            <input type="submit" id="home" value="home" />
        </form>
    </div>
</body>
</html>
```

Sr No.	Description
(1)	Method of payment is card payment when card number is registered as the account information. It is considered as cash payment when card number is not registered.

Create JSP to display the information after the confirmation of order.

"/session-tutorial-init-web/src/main/webapp/WEB-INF/views/order/finish.jsp"

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8" />
<title>Order Page</title>
<link rel="stylesheet"
    href="${pageContext.request.contextPath}/resources/app/css/styles.css">
</head>
<body>

    <sec:authentication property="principal" var="userDetails" />

    <div style="display: inline-flex">
        welcome ${f:h(userDetails.account.name)}
        <form:form method="post"
            action="${pageContext.request.contextPath}/logout">
            <input type="submit" id="logout" value="logout" />
        </form:form>
        <form method="get"
            action="${pageContext.request.contextPath}/account/update">
            <input type="submit" name="form1" id="updateAccount"
                value="Account Update" />
        </form>
    </div>
    <br>
    <br>

    <div>

        <h3>Your order has been accepted</h3>
```

```
        <table>
            <tr>
                <td><label for="orderNumber">order number</label></td>
                <td><span id="orderNumber">${f:h(order.id)}</span></td>
            </tr>
            <tr>
                <td><label for="orderDate">order date</label></td>
                <td><span id="orderDate"><fmt:formatDate
                            value="${order.orderDate}" pattern="yyyy-MM-dd  hh:mm:
↪ss" /></span></td>
            </tr>
        </table>
        <table>
            <tr>
                <th>Name</th>
                <th>Price</th>
                <th>Quantity</th>
            </tr>
            <c:forEach items="${order.orderLines}" var="orderLine" varStatus=
↪"status">
                <tr>
                    <td><span id="itemName${status.index}">${f:h(orderLine.goods.
↪name)}</span></td>
                    <td><span id="itemPrice${status.index}"><fmt:formatNumber
                                value="${orderLine.goods.price}" type="CURRENCY"
                                currencySymbol="&yen;" maxFractionDigits="0" /></
↪span></td>
                    <td><span id="itemQuantity${status.index}">${f:h(orderLine.
↪quantity)}</span></td>
                </tr>
            </c:forEach>
            <tr>
                <td>Total</td>
                <td><span id="totalPrice"><fmt:formatNumber
                            value="${f:h(order.totalAmount)}" type="CURRENCY"
                            currencySymbol="&yen;" maxFractionDigits="0" /></span>
↪</td>
                <td></td>
            </tr>
        </table>
    </div>
    <div>
        <form method="get" action="${pageContext.request.contextPath}/goods">
            <input type="submit" id="home" value="home" />
        </form>
    </div>
</body>
</html>
```

Checking operation

Products registered in the cart can be ordered with the implementation so far. Transition to the order confirmation screen takes place by clicking 'confirm your order' button on the cart display screen. Order is completed by clicking 'order' button on the order confirmation screen.

Cart object in the session when the order is completed, is deleted with the implementation so far. Therefore, contents of the cart are cleared when the user returns to the product list screen after the order completion.

Settings for session synchronization and timeout

Finally perform the settings for session synchronization and timeout.

Session synchronization is implemented using BeanProcessor.

"/session-tutorial-init-web/src/main/java/com/example/session/app/config/EnableSynchronizeOnSessionPostProcessor.java"

```
package com.example.session.app.config;

import org.springframework.beans.BeansException;
import org.springframework.beans.factory.config.BeanPostProcessor;
import org.springframework.web.servlet.mvc.method.annotation.
↪RequestMappingHandlerAdapter;

public class EnableSynchronizeOnSessionPostProcessor implements
        BeanPostProcessor {

    @Override
    public Object postProcessBeforeInitialization(Object bean, String beanName)
            throws BeansException {
        return bean;
    }

    @Override
    public Object postProcessAfterInitialization(Object bean, String beanName)
            throws BeansException {
        if (bean instanceof RequestMappingHandlerAdapter) {
            RequestMappingHandlerAdapter adapter = (RequestMappingHandlerAdapter)␣
↪bean;
            adapter.setSynchronizeOnSession(true); // (1)
        }
        return bean;
    }
}
```

Sr. No.	Description
(1)	Requests within the same session are synchronized by specifying true in the argument of setSynchronizeOnSession method.

"/session-tutorial-init-web/src/main/resources/META-INF/spring/spring-mvc.xml"

```
<!-- Bean Processor -->
<bean class="com.example.session.app.config.EnableSynchronizeOnSessionPostProcessor
↪" />
```

Set the timeout time in web.xml. Set the default value as 30 minutes.

"/session-tutorial-init-web/src/main/webapp/WEB-INF/web.xml" (set by default)

```
<session-config>
    <!-- 30min -->
    <session-timeout>30</session-timeout>
    <cookie-config>
        <http-only>true</http-only>
        <!-- <secure>true</secure> -->
    </cookie-config>
    <tracking-mode>COOKIE</tracking-mode>
</session-config>
```

Use the Spring Security function for request detection after timeout.

"/session-tutorial-init-web/src/main/resources/META-INF/spring/spring-security.xml"

```
<!-- (1) -->
<sec:session-management invalid-session-url="/loginForm" />
```

Sr. No.	Description
(1)	Mention the destination for transition when request after the timeout is detected in invalid-session-url attribute of sec:session-management tag.

10.3.6 Conclusion

We have learnt the following contents in this tutorial.

- How to design the data for session management
  - Selecting data stored in the session

- Example of the flow to determine whether to use the session
- Discarding data during the session

•Specific method of using session in this FW

- Method of using @SessionAttributes
- Method of using the Bean for session scope
- How to refer the data in the session in each method of usage
- How to discard session in each method of usage

10.4 Spring Security Tutorial

Table of Contents

- *Introduction*
  - *Topics covered in this tutorial*
  - *Target Audience*
  - *Verification environment*
- *Overview of application to be created*
- *Creating environment*
  - *Creating a project*
- *Creating an application*
  - *Implementing domain layer*
    - *Creating a Domain Object*
    - *Creating AccountRepository*
    - *Creating AccountSharedService*
    - *Creating Authentication Service*
    - *Setting database initialization script*
    - *Package Explorer after creating Domain Layer*
  - *Implementing Application Layer*

        * *Spring Security settings*
        * *Creating login page*
        * *Accessing account information of login user from JSP*
        * *Adding logout button*
        * *Accessing account information of login user from Controller*
        * *Package explorer after creating application layer*
    - *Summary*
    - *Appendix*
        - *Description of configuration file*
            * *spring-security.xml*
            * *spring-mvc.xml*

10.4.1 Introduction

Topics covered in this tutorial

- Basic authentication/authorization using Spring Security
- Login using the account information in the database
- How to fetch authenticated account object

Target Audience

- Completed implementation of *Tutorial (Todo Application)*(MyBatis3 should be used for implementing infrastructure layer)
- Understands the basic operations of Maven

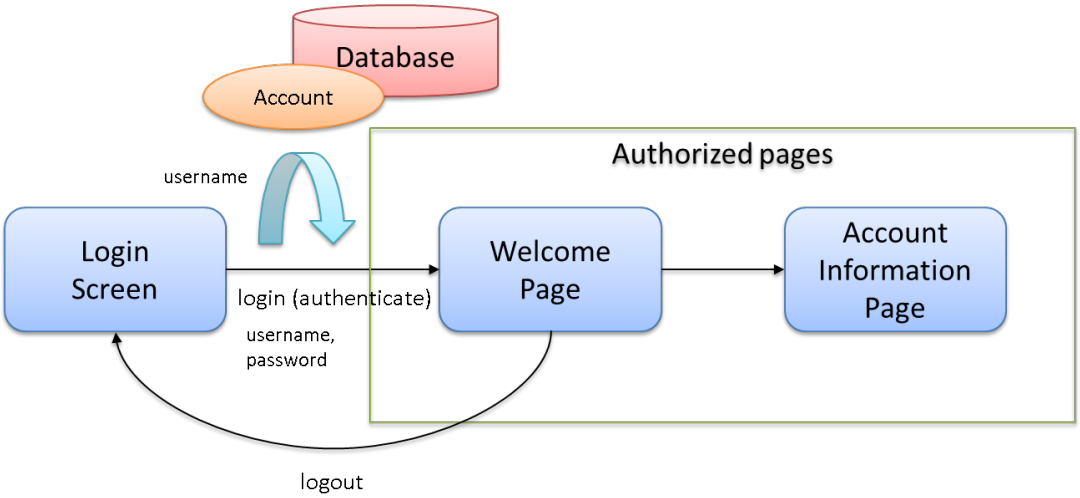
Verification environment

- Similar to *Tutorial (Todo Application)*.

10.4.2 Overview of application to be created

- Login to application is possible by specifying ID and password on login page.
- Store account information required for login in the database.
- There is a Welcome page and Account information display page which can be viewed only by the logged in users.
- Logout from application is possible.

Overview of application is shown in the following figure.



URL list is shown below.

Sr. No.	Process name	HTTP method	URL	Description
1	Login form display	GET	/login.jsp	Displays login form
2	Login	POST	/authentication	Authenticates using username and password entered from login form (performed by Spring Security)
3	Welcome page display	GET	/	Displays Welcome page.
4	Account information display	GET	/account	Displays account information of logged-in user.
5	Logout	POST	/logout	Performs logout (performed by Spring Security)

10.4.3 Creating environment

Creating a project

Create A blank project of TERASOLUNA Server Framework for Java (5.x) using Maven archetype.

In this tutorial, a blank project is created for MyBatis3.

Basic knowledge such as how to import to Spring Tool Suite(STS), how to start an application server, etc. is omitted in this tutorial, since it is already described in *Tutorial (Todo Application)*.

```
mvn archetype:generate -B^
 -DarchetypeGroupId=org.terasoluna.gfw.blank^
 -DarchetypeArtifactId=terasoluna-gfw-web-blank-mybatis3-archetype^
 -DarchetypeVersion=5.3.0.RELEASE^
 -DgroupId=com.example.security^
 -DartifactId=first-springsecurity^
 -Dversion=1.0.0-SNAPSHOT
```

Most of the settings which are required for executing this tutorial are already performed in blank project. It is not mandatory to understand these settings just for executing the tutorial; however, it is recommended that you understand the settings which are required to run the application.

For description about settings required to run the application (configuration file), refer to "*Description of configuration file*".

10.4.4 Creating an application

Implementing domain layer

The flow of authentication process of Spring Security is as follows:

1. Search user information from the entered `username`.
2. When user information exists, compare the password stored in the corresponding user information with the hashed password that has been entered.
3. When passwords match, authentication is considered to be successful.

If user information is not found or if the passwords do not match, authentication fails.

In domain layer, process to fetch Account object from user name is essential. The process is implemented in the following order.

1. Creation of Domain Object(`Account`)
2. Creation of `AccountRepository`
3. Creation of `AccountSharedService`

Creating a Domain Object

Create `Account` class that stores authentication information (user name and password).

`src/main/java/com/example/security/domain/model/Account.java`

```
package com.example.security.domain.model;

import java.io.Serializable;

public class Account implements Serializable {
    private static final long serialVersionUID = 1L;

    private String username;

    private String password;

    private String firstName;

    private String lastName;

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getPassword() {
        return password;
    }
```

```
    public void setPassword(String password) {
        this.password = password;
    }

    public String getFirstName() {
        return firstName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

    @Override
    public String toString() {
        return "Account [username=" + username + ", password=" + password
                + ", firstName=" + firstName + ", lastName=" + lastName + "]";
    }
}
```

Creating AccountRepository

Implement a process to fetch `Account` object from the database.

Create `AccountRepository` interface.

`src/main/java/com/example/security/domain/repository/account/AccountRepository.java`

```
package com.example.security.domain.repository.account;

import com.example.security.domain.model.Account;

public interface AccountRepository {
```

```
    Account findOne(String username);
}
```

Define SQL for fetching single `Account` record, in Mapper file.

`src/main/resources/com/example/security/domain/repository/account/AccountRepository.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
    "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.example.security.domain.repository.account.AccountRepository
↪">

    <resultMap id="accountResultMap" type="Account">
        <id property="username" column="username" />
        <result property="password" column="password" />
        <result property="firstName" column="first_name" />
        <result property="lastName" column="last_name" />
    </resultMap>

    <select id="findOne" parameterType="String" resultMap="accountResultMap">
        SELECT
            username,
            password,
            first_name,
            last_name
        FROM
            account
        WHERE
            username = #{username}
    </select>
</mapper>
```

Creating AccountSharedService

Implement a business process to fetch `Account` object from user name.

Since this process is to be used from Spring Security's Authentication service, interface name would be `AccountSharedService` and class name would be `AccountSharedServiceImpl`.

Note: This guideline does not recommend calling a Service from another Service.

To have common domain layer process (service), it is recommended to name it as `XxxSharedService` instead of `XxxService` to indicate that it is a service common across various service processes.

The application created in this tutorial does not require common services. However, in general application, it is assumed to have common services for processing the account information. Therefore, in this tutorial, process to fetch the account information is implemented as SharedService.

Create `AccountSharedService` interface.

`src/main/java/com/example/security/domain/service/account/AccountSharedService.java`

```
package com.example.security.domain.service.account;

import com.example.security.domain.model.Account;

public interface AccountSharedService {
    Account findOne(String username);
}
```

Create `AccountSharedServiceImpl` class.

`src/main/java/com/example/security/domain/service/account/AccountSharedServiceImpl.java`

```
package com.example.security.domain.service.account;

import javax.inject.Inject;
```

```
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import org.terasoluna.gfw.common.exception.ResourceNotFoundException;

import com.example.security.domain.model.Account;
import com.example.security.domain.repository.account.AccountRepository;

@Service
public class AccountSharedServiceImpl implements AccountSharedService {
    @Inject
    AccountRepository accountRepository;

    @Transactional(readOnly=true)
    @Override
    public Account findOne(String username) {
        // (1)
        Account account = accountRepository.findOne(username);
        // (2)
        if (account == null) {
            throw new ResourceNotFoundException("The given account is not found!␣
↪username="
                    + username);
        }
        return account;
    }

}
```

Sr. No.	Description
(1)	Fetch single `Account` object that matches with the user name.
(2)	If `Account` that matches with the user name does not exist, throw `ResourceNotFoundException` provided by common library.

Creating Authentication Service

Create a class to store authenticated user information which is used in Spring Security.

`src/main/java/com/example/security/domain/service/userdetails/SampleUserDetails.java`

```
package com.example.security.domain.service.userdetails;

import org.springframework.security.core.authority.AuthorityUtils;
import org.springframework.security.core.userdetails.User;

import com.example.security.domain.model.Account;

public class SampleUserDetails extends User { // (1)
    private static final long serialVersionUID = 1L;

    private final Account account; // (2)

    public SampleUserDetails(Account account) {
        // (3)
        super(account.getUsername(), account.getPassword(), AuthorityUtils
                .createAuthorityList("ROLE_USER")); // (4)
        this.account = account;
    }

    public Account getAccount() { // (5)
        return account;
    }

}
```

Sr. No.	Description
(1)	Implement `org.springframework.security.core.userdetails.UserDetails` interface. Here, implement project specific `UserDetails` class, by inheriting `org.springframework.security.core.userdetails.User` class that implements `UserDetails`.
(2)	Maintain account information of this project in Spring's authentication user class.
(3)	Call constructor of `User` class. The first argument is user name, the second is password and the third is authority list.
(4)	As a simple implementation, create an authority having only a role named as `"ROLE_USER"`.
(5)	Create getter of account information. This enables fetching of `Account` object of login user.

Create a service to fetch authentication user information which is used in Spring Security.

`src/main/java/com/example/security/domain/service/userdetails/SampleUserDetailsService.j`

```
package com.example.security.domain.service.userdetails;

import javax.inject.Inject;

import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import org.terasoluna.gfw.common.exception.ResourceNotFoundException;
```

```
import com.example.security.domain.model.Account;
import com.example.security.domain.service.account.AccountSharedService;

@Service
public class SampleUserDetailsService implements UserDetailsService { // (1)
    @Inject
    AccountSharedService accountSharedService; // (2)

    @Transactional(readOnly=true)
    @Override
    public UserDetails loadUserByUsername(String username) throws␣
↪UsernameNotFoundException {
        try {
            Account account = accountSharedService.findOne(username); // (3)
            return new SampleUserDetails(account); // (4)
        } catch (ResourceNotFoundException e) {
            throw new UsernameNotFoundException("user not found", e); // (5)
        }
    }

}
```

Sr. No.	Description
(1)	Implement `org.springframework.security.core.userdetails.UserDetailsService` interface.
(2)	Inject `AccountSharedService`.
(3)	Delegate the process of fetching `Account` object from `username` to `AccountSharedService`.
(4)	Create project specific `UserDetails` object using the fetched `Account` object, and return as the return value of method.
(5)	Throw `UsernameNotFoundException` when target user is not found.

Setting database initialization script

In this tutorial, H2 database (in memory database) is used as a database to store account information. As a result, database initialization is necessary by executing SQL at the time of starting the application server.

Add the settings for executing SQL script that is used to initialize the database.

`src/main/resources/META-INF/spring/first-springsecurity-env.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:jdbc="http://www.springframework.org/schema/jdbc"
    xsi:schemaLocation="
        http://www.springframework.org/schema/jdbc http://www.springframework.org/
↪schema/jdbc/spring-jdbc.xsd
        http://www.springframework.org/schema/beans http://www.springframework.org/
↪schema/beans/spring-beans.xsd
    ">

    <bean id="dateFactory" class="org.terasoluna.gfw.common.date.jodatime.
↪DefaultJodaTimeDateFactory" />

    <bean id="realDataSource" class="org.apache.commons.dbcp2.BasicDataSource"
        destroy-method="close">
        <property name="driverClassName" value="${database.driverClassName}" />
        <property name="url" value="${database.url}" />
        <property name="username" value="${database.username}" />
        <property name="password" value="${database.password}" />
        <property name="defaultAutoCommit" value="false" />
        <property name="maxTotal" value="${cp.maxActive}" />
        <property name="maxIdle" value="${cp.maxIdle}" />
        <property name="minIdle" value="${cp.minIdle}" />
        <property name="maxWaitMillis" value="${cp.maxWait}" />
    </bean>


    <bean id="dataSource" class="net.sf.log4jdbc.Log4jdbcProxyDataSource">
        <constructor-arg index="0" ref="realDataSource" />
    </bean>

    <!-- (1) -->
    <jdbc:initialize-database data-source="dataSource"
```

```
        ignore-failures="ALL">
        <!-- (2) -->
        <jdbc:script location="classpath:/database/${database}-schema.sql"␣
↪encoding="UTF-8" />
        <!-- (3) -->
        <jdbc:script location="classpath:/database/${database}-dataload.sql"␣
↪encoding="UTF-8" />
    </jdbc:initialize-database>

    <!--  REMOVE THIS LINE IF YOU USE JPA
    <bean id="transactionManager"
        class="org.springframework.orm.jpa.JpaTransactionManager">
        <property name="entityManagerFactory" ref="entityManagerFactory" />
    </bean>
          REMOVE THIS LINE IF YOU USE JPA  -->
    <!--  REMOVE THIS LINE IF YOU USE MyBatis3
    <bean id="transactionManager"
        class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
        <property name="dataSource" ref="dataSource" />
        <property name="rollbackOnCommitFailure" value="true" />
    </bean>
          REMOVE THIS LINE IF YOU USE MyBatis3  -->
</beans>
```

Sr. No.	Description
(1)	Perform settings to execute SQL script that initializes the database in `<jdbc:initialize-database>` tag. Define these settings in `first-springsecurity-env.xml`, since they are normally used only during development (environment dependent settings).
(2)	Specify the SQL file where DDL statement for creating a table that stores account information is mentioned. As per blank project settings, `H2-schema.sql` is executed since `database=H2` is defined in `first-springsecurity-infra.properties`.
(3)	Specify SQL file, where DML statement to register the demo user is mentioned. As per blank project settings, `H2-dataload.sql`is executed since `database=H2` is defined in `first-springsecurity-infra.properties`.

Create DDL statement for creating a table that stores account information.

`src/main/resources/database/H2-schema.sql`

```
CREATE TABLE account(
    username varchar(128),
    password varchar(60),
    first_name varchar(128),
    last_name varchar(128),
    constraint pk_tbl_account primary key (username)
);
```

Create DML statement to register the demo user (username=demo, password=demo).

`src/main/resources/database/H2-dataload.sql`

```
INSERT INTO account(username, password, first_name, last_name) VALUES('demo', '$2a
↪$10$oxSJl.keBwxmsMLkcT9lPeAIxfNTPNQxpeywMrF7A3kVszwUTqfTK', 'Taro', 'Yamada'); --
 (1)
COMMIT;
```

Sr. No.	Description
(1)	As per blank project settings, `org.springframework.security.crypto.bcrypt.BCryptPasswordE` is set as a class to hash the password in `applicationContext.xml`. In this tutorial, a string called `"demo"` that is hashed using BCrypt algorithm is inserted in the password in order to perform password hashing using `BCryptPasswordEncoder`.

Package Explorer after creating Domain Layer

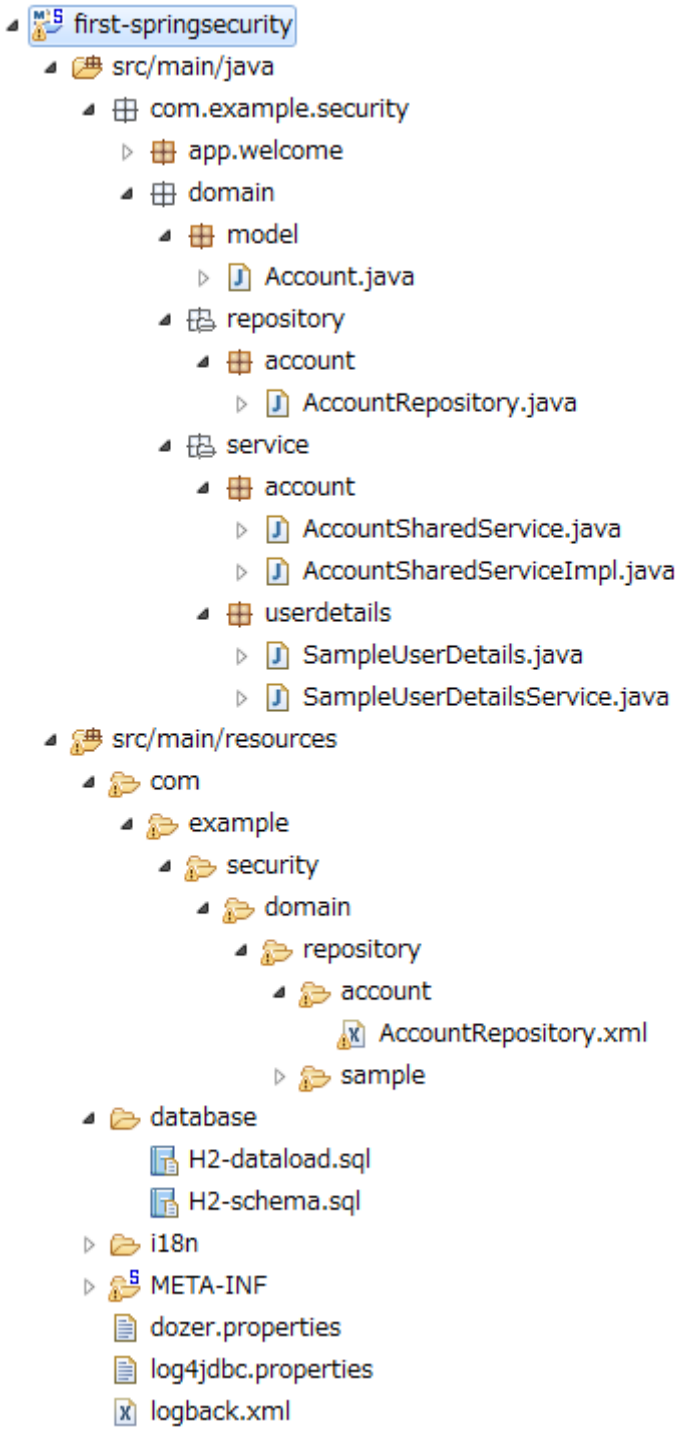
Confirm the file created in domain layer.

"Hierarchical" is being used for Package Presentation of Package Explorer.

Implementing Application Layer

Spring Security settings

Perform authentication/authorization settings using Spring Security in `spring-security.xml`.


first-springsecurity
src/main/java
com.example.security
app.welcome
domain
model
Account.java
repository
account
AccountRepository.java
service
account
AccountSharedService.java
AccountSharedServiceImpl.java
userdetails
SampleUserDetails.java
SampleUserDetailsService.java
src/main/resources
com
example
security
domain
repository
account
AccountRepository.xml
sample
database
H2-dataload.sql
H2-schema.sql
i18n
META-INF
dozer.properties
log4jdbc.properties
logback.xml

Following are URL patterns to be handled by the application created in this tutorial.

URL	Description
/login.jsp	URL to display login form
/login.jsp?error=true	URL to display transition page (login page) in case of authentication error
/login	URL for authentication
/logout	URL for logout
/	URL to display welcome page
/account	URL to display account information of login user.

Add following settings apart from the settings provided by blank project.

`src/main/resources/META-INF/spring/spring-security.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:sec="http://www.springframework.org/schema/security"
    xsi:schemaLocation="
        http://www.springframework.org/schema/security http://www.springframework.
↪org/schema/security/spring-security.xsd
```

```
        http://www.springframework.org/schema/beans http://www.springframework.org/
↪schema/beans/spring-beans.xsd
    ">

    <sec:http pattern="/resources/**" security="none"/>
    <sec:http>
        <!-- (1) -->
        <sec:form-login
            login-page="/login.jsp"
            authentication-failure-url="/login.jsp?error=true" />
        <!-- (2) -->
        <sec:logout
            logout-success-url="/"
            delete-cookies="JSESSIONID" />
        <sec:access-denied-handler ref="accessDeniedHandler"/>
        <sec:custom-filter ref="userIdMDCPutFilter" after="ANONYMOUS_FILTER"/>
        <sec:session-management />
        <!-- (3) -->
        <sec:intercept-url pattern="/login.jsp" access="permitAll" />
        <sec:intercept-url pattern="/**" access="isAuthenticated()" />
    </sec:http>

    <sec:authentication-manager>
        <!-- com.example.security.domain.service.userdetails.
↪SampleUserDetailsService
          is scanned by component scan with @Service -->
        <!-- (4) -->
        <sec:authentication-provider
            user-service-ref="sampleUserDetailsService">
            <!-- (5) -->
            <sec:password-encoder ref="passwordEncoder" />
        </sec:authentication-provider>
    </sec:authentication-manager>

    <!-- CSRF Protection -->
    <bean id="accessDeniedHandler"
        class="org.springframework.security.web.access.
↪DelegatingAccessDeniedHandler">
        <constructor-arg index="0">
            <map>
                <entry
                    key="org.springframework.security.web.csrf.
↪InvalidCsrfTokenException">
                    <bean
                        class="org.springframework.security.web.access.
↪AccessDeniedHandlerImpl">
                        <property name="errorPage"
                            value="/WEB-INF/views/common/error/
↪invalidCsrfTokenError.jsp" />
                    </bean>
```

```
                </entry>
                <entry
                    key="org.springframework.security.web.csrf.
↪MissingCsrfTokenException">
                    <bean
                        class="org.springframework.security.web.access.
↪AccessDeniedHandlerImpl">
                        <property name="errorPage"
                            value="/WEB-INF/views/common/error/
↪missingCsrfTokenError.jsp" />
                    </bean>
                </entry>
            </map>
        </constructor-arg>
        <constructor-arg index="1">
            <bean
                class="org.springframework.security.web.access.
↪AccessDeniedHandlerImpl">
                <property name="errorPage"
                    value="/WEB-INF/views/common/error/accessDeniedError.jsp" />
            </bean>
        </constructor-arg>
    </bean>

    <!-- Put UserID into MDC -->
    <bean id="userIdMDCPutFilter" class="org.terasoluna.gfw.security.web.logging.
↪UserIdMDCPutFilter">
    </bean>

</beans>
```

Sr. No.	Description
(1)	Perform settings related to login form using `<sec:form-login>` tag. Perform following settings in `<sec:form-login>` tag • URL to display login form in `login-page` attribute. • URL to display destination page in case of an authentication error in `authentication-failure-url` attribute • URL to perform authentication in `login-processing-url` attribute
(2)	Perform settings for logout using `<sec:logout>` tag. Perform following settings in `<sec:logout>` tag. • URL to perform logout in `logout-url` attribute • URL to display destination page after performing logout in `logout-success-url` attribute (URL to display welcome page in this tutorial) • Cookie name to be deleted at the time of logout in `delete-cookies` attribute (Cookie name of session ID in this tutorial)
(3)	Perform authorization settings for each URL using `<sec:intercept-url>` tag. Perform following settings in `<sec:intercept-url>` tag. • `permitAll` that allows all the users to access the URL to display login form • `isAuthenticated()` that allows only the authenticated users to access the URLs other than the URLs mentioned above However, all users can access the URL under `/resources/`, since the settings are such that authentication/authorization is not performed by Spring Security (`<sec:http pattern="/resources/**" security="none"/>`).
(4)	Perform settings of `org.springframework.security.authentication.AuthenticationProvid` that performs authentication using `<sec:authentication-provider>` tag. By default, `UserDetails` is fetched using `UserDetailsService`, and a class (`org.springframework.security.authentication.dao.DaoAuthenticationProvider`) that performs user authentication by comparing hashed password in `UserDetails` with the password specified in login form, is used. Specify component bean name where `UserDetailsService` interface is implemented in `user-service-ref` attribute. In this tutorial, `SampleUserDetailsService` class created in domain layer is set.
(5)	Perform settings for a class (`PasswordEncoder`) to hash the password specified in login form using `<sec:password-encoder>` tag. In this tutorial, `org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder` defined in `applicationContext.xml` is used.

Note: Default URL provided by Spring Security is changed for the URLs that perform authentication and logout process.

This is because, a string (`spring_security`) that implies the usage of Spring Security is included in these

URLs. When default URL is used as it is and if security vulnerability is detected in Spring Security, please be careful as it becomes easy to receive attacks from a malicious user.

Creating login page

Create login form on login page.

`src/main/webapp/login.jsp`

```
<!DOCTYPE html>
<html>
<head>
<title>Login Page</title>
<link rel="stylesheet"
    href="${pageContext.request.contextPath}/resources/app/css/styles.css">
</head>
<body>
    <div id="wrapper">
        <h3>Login with Username and Password</h3>

        <!-- (1) -->
        <c:if test="${param.containsKey('error')}">
            <!-- (2) -->
            <t:messagesPanel messagesType="error"
                messagesAttributeName="SPRING_SECURITY_LAST_EXCEPTION" />
        </c:if>

        <!-- (3) -->
        <form:form action="${pageContext.request.contextPath}/login">
            <table>
                <tr>
                    <td><label for="username">User:</label></td>
                    <td><input type="text" id="username"
                        name="username" value='demo'>(demo)</td><!-- (4) -->
                </tr>
                <tr>
                    <td><label for="password">Password:</label></td>
                    <td><input type="password" id="password"
                        name="password" value="demo" />(demo)</td><!-- (5) -->
                </tr>
                <tr>
                    <td> </td>
```

```
                    <td><input name="submit" type="submit" value="Login" /></td>
                </tr>
            </table>
        </form:form>
    </div>
</body>
</html>
```

Sr. No.	Description
(1)	When authentication fails, `"/login.jsp?error=true"` is called and login page is displayed. Therefore, use `<c:if>` tag, so that error message is displayed only at the time after the display of authentication error.
(2)	Display an error message using `<t:messagesPanel>` tag provided by common library. When authentication fails, an exception object of authentication error is stored with attribute name `"SPRING_SECURITY_LAST_EXCEPTION"` in session scope.
(3)	Set URL for authentication (`"/login"`) in `action` attribute of `<form:form>` tag. This URL is default for Spring Security. Send parameters necessary for authentication (user name and password) using POST method.
(4)	Create a text box to specify user name. Spring Security's default parameter name is `username`.
(5)	Create a text box to specify password (text box for password). Spring Security's default parameter name is `password`.

Ensure that exception object of authentication error stored in session scope could be fetched from JSP.

`src/main/webapp/WEB-INF/views/common/include.jsp`

```
<%@ page session="true"%> <!-- (6) -->
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt"%>
<%@ taglib uri="http://www.springframework.org/tags" prefix="spring"%>
<%@ taglib uri="http://www.springframework.org/tags/form" prefix="form"%>
<%@ taglib uri="http://www.springframework.org/security/tags" prefix="sec"%>
<%@ taglib uri="http://terasoluna.org/tags" prefix="t"%>
<%@ taglib uri="http://terasoluna.org/functions" prefix="f"%>
```

Sr. No.	Description
(6)	Set `session` attribute of `page` directive to `true`.

Note: As per default settings of blank project, session scope cannot be accessed from JSP. This is to ensure that the session cannot be easily used; however, in case of fetching an exception object of authentication error from JSP, it is necessary to be accessible from a JSP by session scope.

Try to display the welcome page by entering http://localhost:8080/first-springsecurity/ in browser address bar.

Since the user is not logged in, it is transited to the set value of `login-page` attribute of `<sec:form-login>` tag (http://localhost:8080/first-springsecurity/login.jsp), and the screen below is displayed.



Accessing account information of login user from JSP

Access the account information of login user from JSP and display the name.

`src/main/webapp/WEB-INF/views/welcome/home.jsp`

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>Home</title>
<link rel="stylesheet"
    href="${pageContext.request.contextPath}/resources/app/css/styles.css">
```

```
</head>

<!-- (1) -->
<sec:authentication property="principal.account" var="account" />

<body>
    <div id="wrapper">
        <h1>Hello world!</h1>
        <p>The time on the server is ${serverTime}.</p>
        <!-- (2) -->
        <p>Welcome ${f:h(account.firstName)} ${f:h(account.lastName)} !!</p>
        <ul>
            <li><a href="${pageContext.request.contextPath}/account">view account</
↪a></li>
        </ul>
    </div>
</body>
</html>
```

Sr. No.	Description
(1)	Access `org.springframework.security.core.Authentication` object of login user using `<sec:authentication>` tag. By using `property` attribute, any property retained by `Authentication` object can be accessed , and the property value that has been accessed can be stored in any scope using `var` attribute. Page scope is set by default and referred within this JSP only. In this tutorial, `Account` object of login user is stored in page scope with attribute name `account`.
(2)	Access `Account` object of login user and display `firstName` and `lastName`.

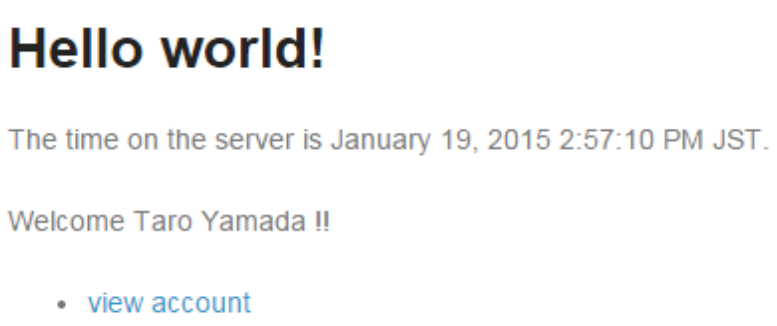
Click Login button on login page to display welcome page.

Adding logout button

Add a button to perform logout.

`src/main/webapp/WEB-INF/views/welcome/home.jsp`

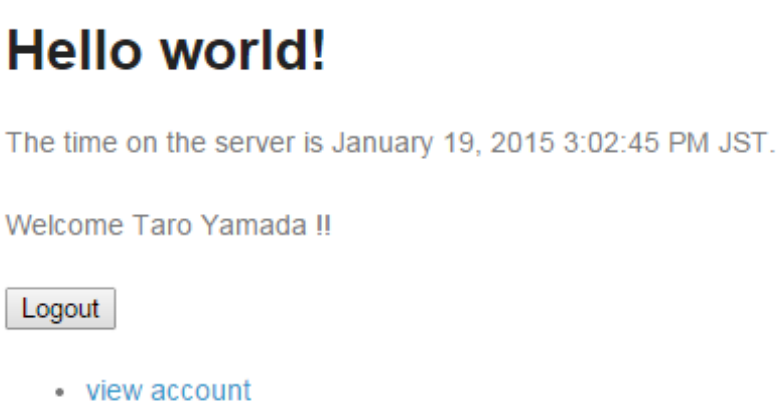
```
<!DOCTYPE html>
<html>
<head>
```




```
<meta charset="utf-8">
<title>Home</title>
<link rel="stylesheet"
    href="${pageContext.request.contextPath}/resources/app/css/styles.css">
</head>

<sec:authentication property="principal.account" var="account" />

<body>
    <div id="wrapper">
        <h1>Hello world!</h1>
        <p>The time on the server is ${serverTime}.</p>
        <p>Welcome ${f:h(account.firstName)} ${f:h(account.lastName)} !!</p>
        <p>
            <!-- (1) -->
            <form:form action="${pageContext.request.contextPath}/logout">
                <button type="submit">Logout</button>
            </form:form>
        </p>
        <ul>
            <li><a href="${pageContext.request.contextPath}/account">view account</
↪a></li>
        </ul>
    </div>
</body>
</html>
```

Sr. No.	Description
(1)	Add a form for logout using `<form:form>` tag. Add Logout button by specifying the URL for logout (`"/logout"`) in `action` attribute. This URL is default for Spring Security.

Click Logout button to log out from the application (login page is displayed).



Accessing account information of login user from Controller

Access account information of login user from Controller and pass it to View.

`src/main/java/com/example/security/app/account/AccountController.java`

```
package com.example.security.app.account;

import org.springframework.security.core.annotation.AuthenticationPrincipal;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;

import com.example.security.domain.model.Account;
import com.example.security.domain.service.userdetails.SampleUserDetails;

@Controller
@RequestMapping("account")
public class AccountController {

    @RequestMapping
    public String view(
            @AuthenticationPrincipal SampleUserDetails userDetails, // (1)
            Model model) {
        // (2)
        Account account = userDetails.getAccount();
        model.addAttribute(account);
        return "account/view";
    }
}
```

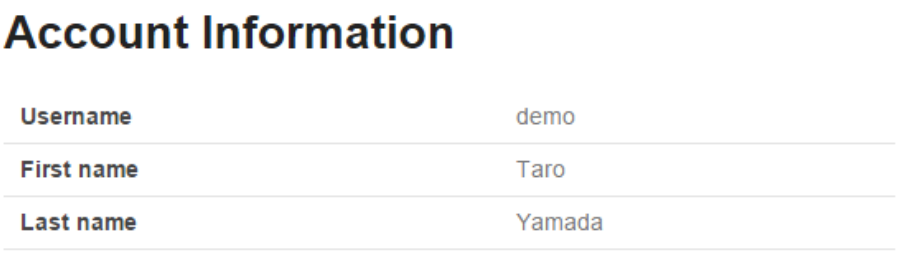
Sr. No.	description
(1)	Receive `UserDetails` object of login user by specifying `@AuthenticationPrincipal` annotation.
(2)	Fetch `Account` object which is retained by `SampleUserDetails` object and store it in `Model` in order to pass it to View.

Access the account information passed from Controller to display the same.

`src/main/webapp/WEB-INF/views/account/view.jsp`

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>Home</title>
<link rel="stylesheet"
    href="${pageContext.request.contextPath}/resources/app/css/styles.css">
</head>
<body>
    <div id="wrapper">
        <h1>Account Information</h1>
        <table>
            <tr>
                <th>Username</th>
                <td>${f:h(account.username)}</td>
            </tr>
            <tr>
                <th>First name</th>
                <td>${f:h(account.firstName)}</td>
            </tr>
            <tr>
                <th>Last name</th>
                <td>${f:h(account.lastName)}</td>
            </tr>
        </table>
    </div>
</body>
</html>
```

Click 'view account' link on welcome page to display "Show account information" page of login user.



Package explorer after creating application layer

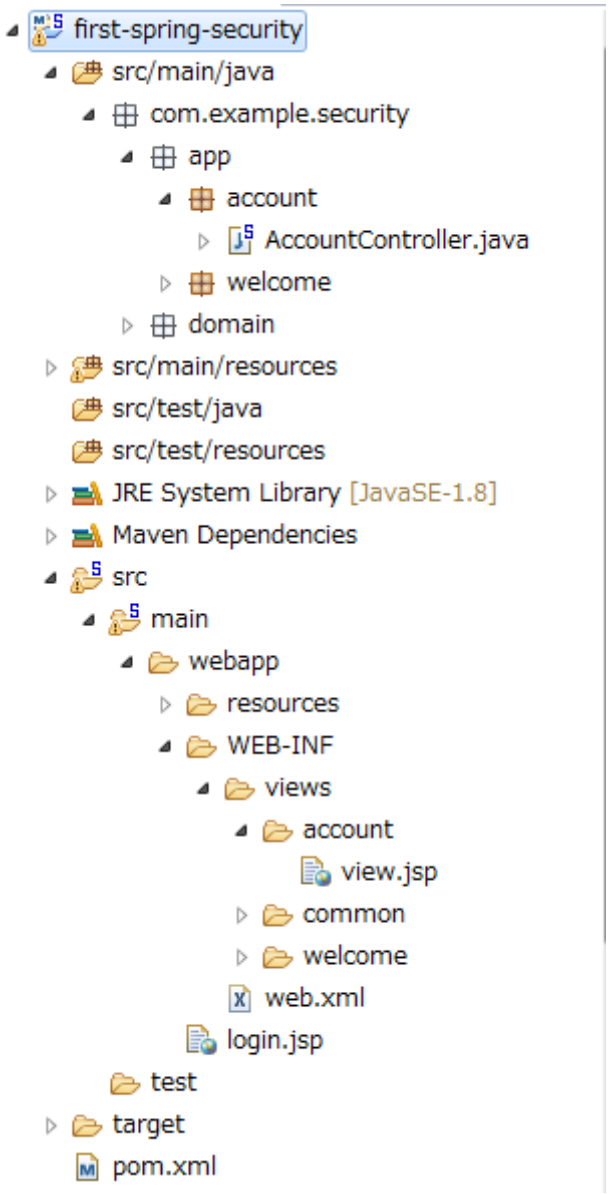
Confirm the file created in application layer.

"Hierarchical" is being used for Package Presentation of Package Explorer.

10.4.5 Summary

We have covered the following topics in this tutorial.

- Basic authentication/authorization using Spring Security
- How to customize authentication user object
- Authentication settings using Repository and Service class
- How to access logged in account information from JSP
- How to access logged in account information from Controller



10.4.6 Appendix

Description of configuration file

Describe configuration file to understand which settings are necessary for using Spring Security.

spring-security.xml

Perform definitions related to Spring Security in `spring-security.xml`.

`src/main/resources/META-INF/spring/spring-security.xml`of the blank project which has been created has following settings.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:sec="http://www.springframework.org/schema/security"
    xsi:schemaLocation="
        http://www.springframework.org/schema/security http://www.springframework.
↪org/schema/security/spring-security.xsd
        http://www.springframework.org/schema/beans http://www.springframework.org/
↪schema/beans/spring-beans.xsd
    ">

    <!-- (1) -->
    <sec:http pattern="/resources/**" security="none"/>
    <sec:http>
        <!-- (2) -->
        <sec:form-login/>
        <!-- (3) -->
        <sec:logout/>
        <!-- (4) -->
        <sec:access-denied-handler ref="accessDeniedHandler"/>
        <!-- (5) -->
        <sec:custom-filter ref="userIdMDCPutFilter" after="ANONYMOUS_FILTER"/>
        <!-- (6) -->
        <sec:session-management />
    </sec:http>

    <!-- (7) -->
    <sec:authentication-manager />

    <!-- (4) -->
    <!-- CSRF Protection -->
    <bean id="accessDeniedHandler"
        class="org.springframework.security.web.access.
↪DelegatingAccessDeniedHandler">
        <constructor-arg index="0">
            <map>
                <entry
                    key="org.springframework.security.web.csrf.
↪InvalidCsrfTokenException">
                    <bean
                        class="org.springframework.security.web.access.
↪AccessDeniedHandlerImpl">
                        <property name="errorPage"
                            value="/WEB-INF/views/common/error/
↪invalidCsrfTokenError.jsp" />
                    </bean>
                </entry>
                <entry
                    key="org.springframework.security.web.csrf.
↪MissingCsrfTokenException">
```

```
                    <bean
                        class="org.springframework.security.web.access.
↪AccessDeniedHandlerImpl">
                        <property name="errorPage"
                            value="/WEB-INF/views/common/error/
↪missingCsrfTokenError.jsp" />
                    </bean>
                </entry>
            </map>
        </constructor-arg>
        <constructor-arg index="1">
            <bean
                class="org.springframework.security.web.access.
↪AccessDeniedHandlerImpl">
                <property name="errorPage"
                    value="/WEB-INF/views/common/error/accessDeniedError.jsp" />
            </bean>
        </constructor-arg>
    </bean>

    <!-- (5) -->
    <!-- Put UserID into MDC -->
    <bean id="userIdMDCPutFilter" class="org.terasoluna.gfw.security.web.logging.
↪UserIdMDCPutFilter">
    </bean>

</beans>
```

Sr. No.	Description
(1)	Control authentication/authorization for HTTP access using `<sec:http>` tag. As per the default settings of blank project , URL to access static resources (js, css, image files, etc.) is out of authentication/authorization scope.
(2)	Control login related operation which use form authentication, by using `<sec:form-login>` tag. For usage method, refer to "*Form authentication*".
(3)	Control logout related operations by using `<sec:logout>` tag. For usage method, refer to [*Logout*].
(4)	Control action after access is denied using `<sec:access-denied-handler>` tag. Following settings are performed as default settings of blank project. • Destination when invalid CSRF token is detected (when `InvalidCsrfTokenException` occurs) • Destination when CSRF token cannot be fetched from token store (when `MissingCsrfTokenException` occurs) • Destination when access is denied in authorization (when `AccessDeniedException` other than above mentioned occurs)
(5)	Enable servlet filter to store authentication user name of Spring Security in logger MDC. Once this setting is enabled, authentication user name is output in log thereby enhancing the traceability.
(6)	Control session management method of Spring Security using `<sec:session-management>` tag. For usage method, refer to "*Applying Session management function*".
(7)	Control authentication using `<sec:authentication-manager>` tag. For usage method, refer to "*Applying DB authentication*".

spring-mvc.xml

Perform settings to link Spring Security and Spring MVC in `spring-mvc.xml`.

`src/main/resources/META-INF/spring/spring-mvc.xml` of the blank project that has been created has following settings. Description of settings not related to Spring Security is omitted.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:mvc="http://www.springframework.org/schema/mvc"
    xmlns:util="http://www.springframework.org/schema/util"
    xmlns:aop="http://www.springframework.org/schema/aop"
    xsi:schemaLocation="http://www.springframework.org/schema/mvc http://www.
↪springframework.org/schema/mvc/spring-mvc.xsd
        http://www.springframework.org/schema/beans http://www.springframework.org/
↪schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/util http://www.springframework.org/
↪schema/util/spring-util.xsd
        http://www.springframework.org/schema/context http://www.springframework.
↪org/schema/context/spring-context.xsd
        http://www.springframework.org/schema/aop http://www.springframework.org/
↪schema/aop/spring-aop.xsd
    ">

    <context:property-placeholder
        location="classpath*:/META-INF/spring/*.properties" />

    <mvc:annotation-driven>
        <mvc:argument-resolvers>
            <bean
                class="org.springframework.data.web.
↪PageableHandlerMethodArgumentResolver" />
            <!-- (1) -->
            <bean
                class="org.springframework.security.web.method.annotation.
↪AuthenticationPrincipalArgumentResolver" />
        </mvc:argument-resolvers>
        <!-- workaround to CVE-2016-5007. -->
        <mvc:path-matching path-matcher="pathMatcher" />
    </mvc:annotation-driven>

    <mvc:default-servlet-handler />

    <context:component-scan base-package="com.example.security.app" />

    <mvc:resources mapping="/resources/**"
        location="/resources/,classpath:META-INF/resources/"
        cache-period="#{60 * 60}" />

    <mvc:interceptors>
        <mvc:interceptor>
            <mvc:mapping path="/**" />
            <mvc:exclude-mapping path="/resources/**" />
            <mvc:exclude-mapping path="/**/*.html" />
            <bean
```

```
                class="org.terasoluna.gfw.web.logging.TraceLoggingInterceptor" />
        </mvc:interceptor>
        <mvc:interceptor>
            <mvc:mapping path="/**" />
            <mvc:exclude-mapping path="/resources/**" />
            <mvc:exclude-mapping path="/**/*.html" />
            <bean
                class="org.terasoluna.gfw.web.token.transaction.
↪TransactionTokenInterceptor" />
        </mvc:interceptor>
        <mvc:interceptor>
            <mvc:mapping path="/**" />
            <mvc:exclude-mapping path="/resources/**" />
            <mvc:exclude-mapping path="/**/*.html" />
            <bean class="org.terasoluna.gfw.web.codelist.CodeListInterceptor">
                <property name="codeListIdPattern" value="CL_.+" />
            </bean>
        </mvc:interceptor>
        <!--  REMOVE THIS LINE IF YOU USE JPA
        <mvc:interceptor>
            <mvc:mapping path="/**" />
            <mvc:exclude-mapping path="/resources/**" />
            <mvc:exclude-mapping path="/**/*.html" />
            <bean
                class="org.springframework.orm.jpa.support.
↪OpenEntityManagerInViewInterceptor" />
        </mvc:interceptor>
            REMOVE THIS LINE IF YOU USE JPA  -->
    </mvc:interceptors>

    <!-- Settings View Resolver. -->
    <mvc:view-resolvers>
        <mvc:bean-name />
        <mvc:tiles />
        <mvc:jsp prefix="/WEB-INF/views/" />
    </mvc:view-resolvers>

    <mvc:tiles-configurer>
        <mvc:definitions location="/WEB-INF/tiles/tiles-definitions.xml" />
    </mvc:tiles-configurer>

    <bean id="requestDataValueProcessor"
        class="org.terasoluna.gfw.web.mvc.support.
↪CompositeRequestDataValueProcessor">
        <constructor-arg>
            <util:list>
                <!-- (2) -->
                <bean
                    class="org.springframework.security.web.servlet.support.csrf.
↪CsrfRequestDataValueProcessor" />
```

```
                <bean
                    class="org.terasoluna.gfw.web.token.transaction.
↪TransactionTokenRequestDataValueProcessor" />
            </util:list>
        </constructor-arg>
    </bean>

    <!-- Setting Exception Handling. -->
    <!-- Exception Resolver. -->
    <bean id="systemExceptionResolver"
        class="org.terasoluna.gfw.web.exception.SystemExceptionResolver">
        <property name="exceptionCodeResolver" ref="exceptionCodeResolver" />
        <!-- Setting and Customization by project. -->
        <property name="order" value="3" />
        <property name="exceptionMappings">
            <map>
                <entry key="ResourceNotFoundException" value="common/error/
↪resourceNotFoundError" />
                <entry key="BusinessException" value="common/error/businessError" /
↪>
                <entry key="InvalidTransactionTokenException" value="common/error/
↪transactionTokenError" />
                <entry key=".DataAccessException" value="common/error/
↪dataAccessError" />
            </map>
        </property>
        <property name="statusCodes">
            <map>
                <entry key="common/error/resourceNotFoundError" value="404" />
                <entry key="common/error/businessError" value="409" />
                <entry key="common/error/transactionTokenError" value="409" />
                <entry key="common/error/dataAccessError" value="500" />
            </map>
        </property>
        <property name="defaultErrorView" value="common/error/systemError" />
        <property name="defaultStatusCode" value="500" />
    </bean>
    <!-- Setting AOP. -->
    <bean id="handlerExceptionResolverLoggingInterceptor"
        class="org.terasoluna.gfw.web.exception.
↪HandlerExceptionResolverLoggingInterceptor">
        <property name="exceptionLogger" ref="exceptionLogger" />
    </bean>
    <aop:config>
        <aop:advisor advice-ref="handlerExceptionResolverLoggingInterceptor"
            pointcut="execution(* org.springframework.web.servlet.
↪HandlerExceptionResolver.resolveException(..))" />
    </aop:config>

    <!-- Setting PathMatcher. -->
```

```
    <bean id="pathMatcher" class="org.springframework.util.AntPathMatcher">
        <property name="trimTokens" value="false" />
    </bean>

</beans>
```

Sr. No.	Description
(1)	Settings to ensure that `UserDetails` object of login user is received as Controller argument, by specifying `@AuthenticationPrincipal` annotation. Specify `AuthenticationPrincipalArgumentResolver` in `<mvc:argument-resolvers>` tag.
(2)	Settings to embed CSRF token value in HTML form using `<form:form>` tag (JSP tag library). Specify `CsrfRequestDataValueProcessor` in `CompositeRequestDataValueProcessor` constructor.

11

Appendix(Know How)

11.1 Maven Repository Management using NEXUS

Sonatype NEXUS is the package repository manager software. OSS version as well as commercial version of NEXUS is available. However its OSS version also has adequate functionalities.

This chapter explains the role and configuration method of OSS version of NEXUS.

11.1.1 Why NEXUS?

When there is only one developer, a central repository on internet and a local repository on developer's machine can be developed using Maven or ant+ivy.

However, when a Java application is to be divided into multiple sub-projects and development is to be carried out in a team, library dependency resolution becomes complex; hence this dependency resolution needs to be automated. For this, availability of package repository server is essential.

The following package repositories are required in Java application development project.

- **Proxy repository** creates proxy to access external repository server including the central repository
- **3rd party repository** for distributing the artifacts provided by others within the organization that are not available in the repository on internet
- **Private repository** for storing the artifacts developed within the project
- **Group repository** for consolidating access to artifacts of different multiple repositories into a single repository URL

In case of NEXUS, operations of such multiple repositories can be easily managed.

11.1.2 Install and Start-up

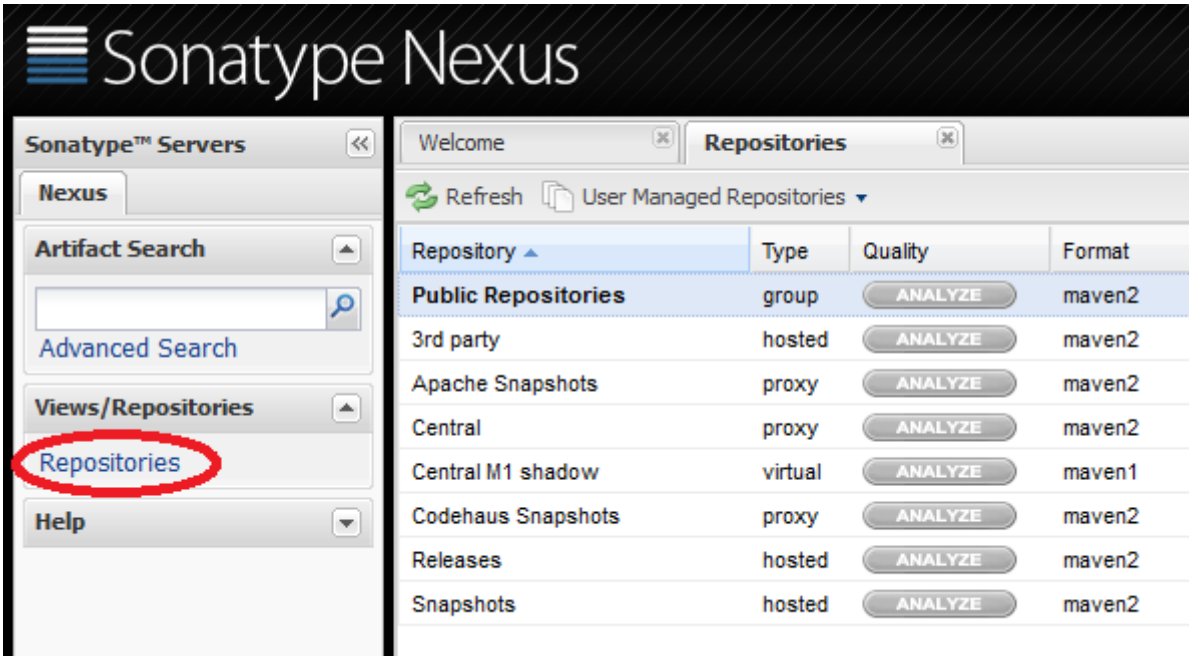
The machine on which NEXUS is to be installed should satisfy the following conditions.

- JRE6 or later should be installed
- Http access to the Central repository (URL starting with http://repo1.maven.org/) is enabled

The installation procedure is as follows:

1. Download NEXUS OSS and deploy archive.
2. Start NEXUS by executing bin/nexus or bin/nexus.bat.
3. Access http://{[}IP or FQDN]:8081/nexus/ and check whether welcome screen of NEXUS is displayed.

Some repositories are provided by default. Except for a few cases, they are used as is for development. Repository list is displayed on clicking Repositories on the menu on the left side of screen.



- **Central** = This repository plays a role of proxy to the central repository on internet (http://repo1.maven.org/maven2/).
- **3rd party** = This repository stores third-party libraries required in development but not available in the repositories on internet.
- **Releases** = This repository stores the work products of release version of the applications developed internally.
- **Snapshots** = This repository stores the work products of SNAPSHOT version of the applications developed internally.

- **Public Repositories** This group repository is used for enabling access to the above repositories through a single URL.

11.1.3 settings.xml

In order to use the created NEXUS using Maven command, settings.xml file needs be created in the home directory of local development environment of the user.

- Windows: C:/Users/[OSaccount]/.m2/settings.xml
- Unix: $HOME/.m2/settings.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<settings>

  <mirrors>
    <mirror>
      <id>myteam-nexus</id>
      <mirrorOf>*</mirrorOf>
      <!-- CHANGE HERE by your team own nexus server -->
      <url>http:// IP or FQDN /nexus/content/groups/public </url>
    </mirror>
  </mirrors>

  <activeProfiles>
    <activeProfile>myteam-nexus</activeProfile>
  </activeProfiles>

  <profiles>
    <profile>
      <id>myteam-nexus</id>
      <repositories>
        <repository>
          <id>central</id>
          <url>http://central</url>
          <releases><enabled>true</enabled></releases>
          <snapshots><enabled>true</enabled></snapshots>
        </repository>
      </repositories>
      <pluginRepositories>
        <pluginRepository>
          <id>central</id>
          <url>http://central</url>
          <releases><enabled>true</enabled></releases>
          <snapshots><enabled>true</enabled></snapshots>
        </pluginRepository>
      </pluginRepositories>
    </profile>
  </profiles>
```

```
</settings>
```

Note: Additionally, also refer to: Configuring Maven to Use a Single Repository Group / Documentation Sonatype.com

11.1.4 mvn deploy how to

Use mvn deploy command to upload jar/war file to the package repository (NEXUS) as an artifact.

A state in which anyone can deploy the application in package repository should be avoided as it causes confusion. Therefore, it is desirable that mvn deployment for package repository is possible only by Jenkins.

In addition to the contents mentioned earlier, add the following to .m2/settings.xml under the home directory of the user executing Jenkins of Jenkins server.

```
<servers>
  <server>
   <id>releases</id>
   <username>deployment</username>
   <password>deployment123</password>
  </server>
  <server>
    <id>snapshots</id>
    <username>deployment</username>
    <password>deployment123</password>
  </server>
</servers>
```

'deployment' is the account (set in NEXUS by default) having deployment permission and its password is deployment123. It is recommended to change the password in advance on NEXUS GUI screen.

Note: To avoid saving the password in plain text in settings.xml, it is advisable to use password encryption function of Maven. Refer to Maven - Password Encryption for details.

Carry out the mvn deployment procedure in the build job of Jenkins as follows:

Todo

Provide screen capture of build job of Jenkins

11.1.5 pom.xml

In case of the project managed in Maven, package repository in which artifact is stored, should be specified using <distributionManagement> tag of pom.xml.

```
<distributionManagement>
  <repository>
    <id>releases</id>
    <!-- CHANGE HERE by your team nexus server -->
    <url>http://192.168.0.1:8081/nexus/content/repositories/releases/</url>
  </repository>
  <snapshotRepository>
    <id>snapshots</id>
    <!-- CHANGE HERE by your team nexus server -->
    <url>http://192.168.0.1:8081/nexus/content/repositories/snapshots/</url>
  </snapshotRepository>
</distributionManagement>
```

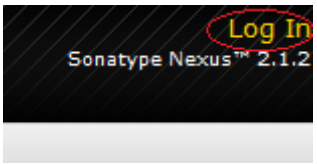
The mvn deploy command uploads the artifact with HTTP PUT for the URL specified using <distributionManagement> tag.

11.1.6 Upload 3rd party artifact (ex. ojdbc6.jar)

In the 3rd party repository, store the artifact which is not disclosed in external remote repository.

Typical example is JDBC driver (ojdbc*.jar) of oracle. Oracle should be used as RDBMS; however, central repository is not stored in the public repository on internet. Therefore, it should be stored in the package repository in the organization.

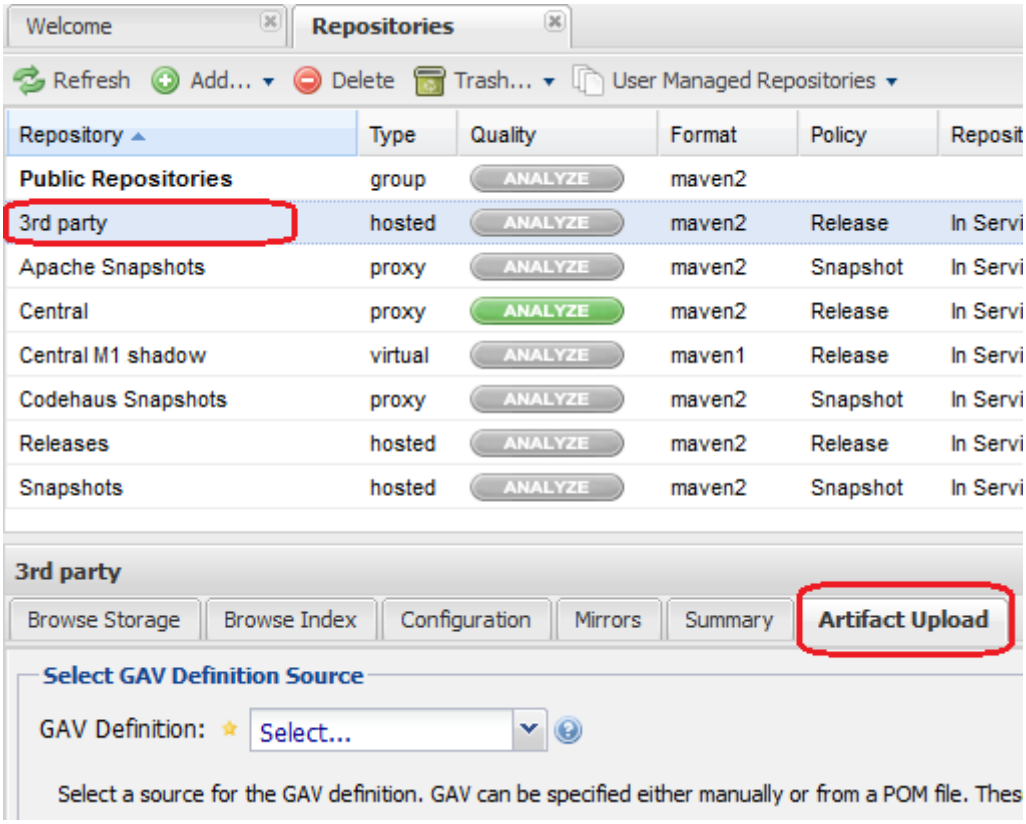
1. Login as admin user. (Default password is admin123)

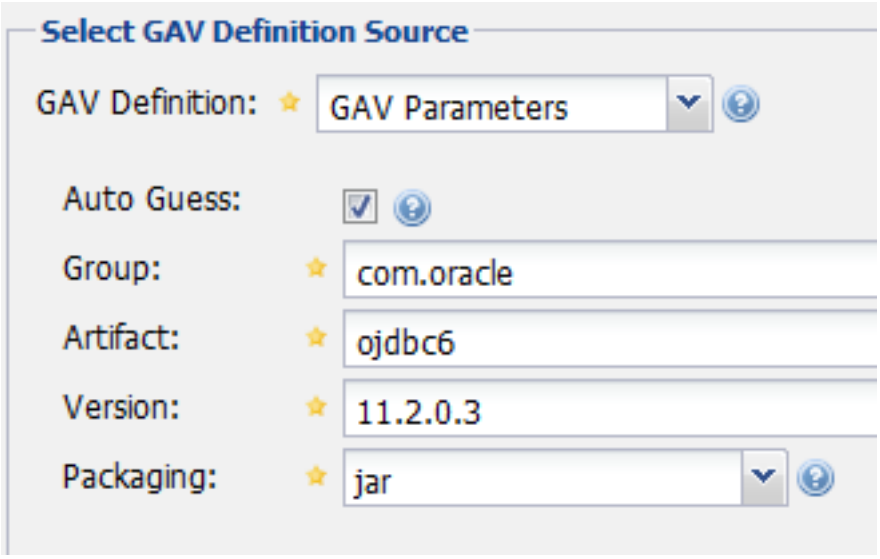


2. Select 3rdParty repository and select **Artifact Upload** tab.

3. Enter GAV information. (GAV = groupId, artifactId, version)

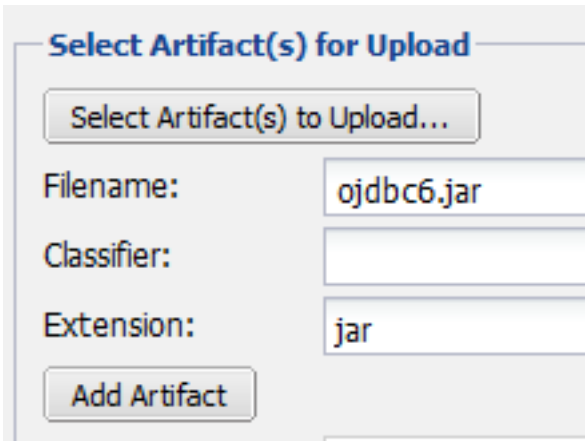
4. Select ojdbc6.jar on local PC and click **Add Artifact** button.

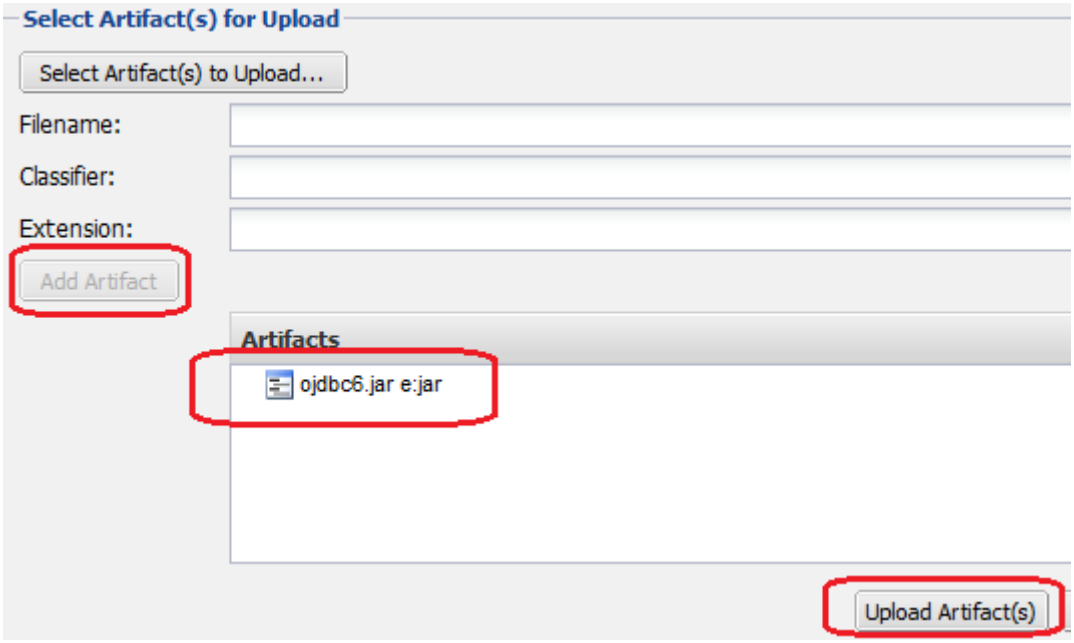




5. At the end, click **Upload Artifact(s)** button to save the jar file in repository.

With this uploading is completed.





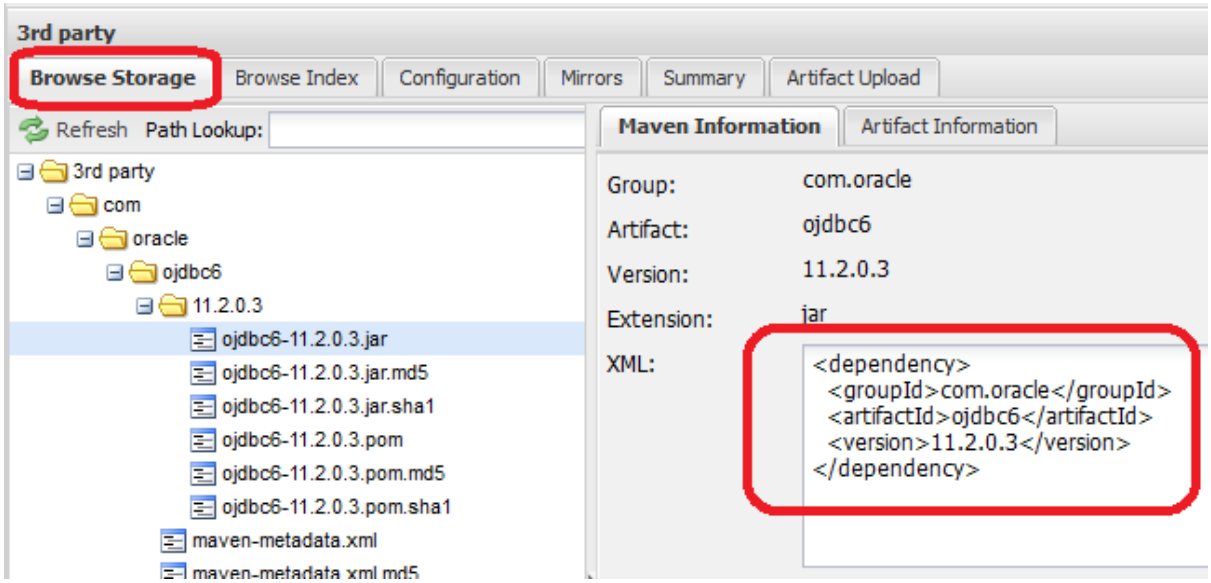
Note: Uploading artifacts using NEXUS GUI screen is a manual task which can easily lead to operational errors. Hence it is not recommended. The method explained here should be used only for simple configurable libraries having 1 or more 3rd party files such as ojdbc6.jar. **mvn deploy** command should be used for other cases.

---

use artifact

In order to add ojdbc6 of 3rd party repository to project dependency management, just add dependency tag to pom.xml of the corresponding project.

If the target artifact is selected from **Browse Storage** tab, sample of dependency tag is displayed on the right side of the screen. It just needs to be copied and pasted in pom.xml.



11.2 Reducing Boilerplate Code (Lombok)

11.2.1 Lombok

Lombok is a library used for reducing the boilerplate code from Java source code.

Boilerplate code is a typical source code that cannot be omitted by language specification. Basically Boilerplate code does not have a specific logic hence it becomes redundant code in implementation.

Following are the typical Boilerplate source code in Java language.

- getter / setter methods for accessing the member variables
- `equals`/`hashCode` methods
- `toString` methods
- Constructors
- Closing process of resources (input and output stream, etc.)
- Generation of logger instance

In Lombok, boilerplate code gets generated at the time of compilation thereby providing a mechanism to remove redundant code from the source code developed by the developer.

Tip: For removing the Boilerplate code, the language specification for closing the resources (input and output stream, etc.) are improved by newly added [try-with-resources] statement in Java SE7.

Java language itself is improving in every version-up for removing the redundant code. Lambda support in Java

SE8 is also called as typical language specification improvement.

11.2.2 Efficiency of Lombok

Below, JavaBean source code is created using Lombok.

```
package com.example.domain.model;

@lombok.Data
public class User {

    private String userId;
    private String password;

}
```

In Lombok, required methods for JavaBean gets created by only assigning `@lombok.Data` annotation at class level.

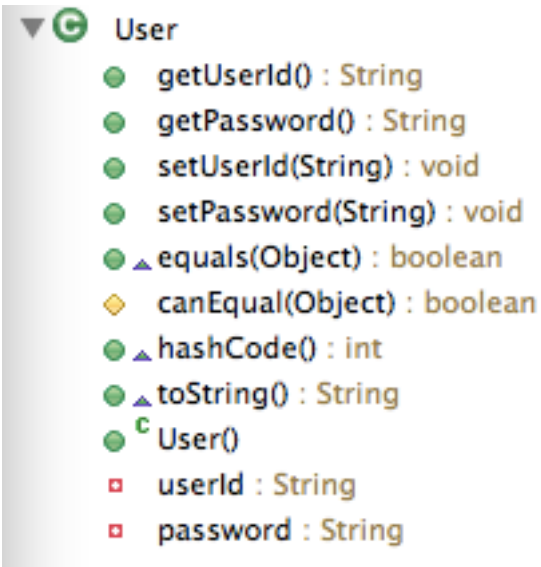


Figure.11.1 **Class structure generated by Lombok**

By only assigning `@Data` annotation of Lombok, it is possible to obtain the same effect as classes generated by (the source code output using the auto-generation function of Eclipse) which has about 60 lines of code given below instead of 10 lines of code.

```
package com.example.domain.model;

public class User {

    private String userId;
    private String password;

    public User() {
    }

    public String getUserId() {
        return userId;
    }

    public void setUserId(String userId) {
        this.userId = userId;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    @Override
    public int hashCode() {
        final int prime = 31;
        int result = 1;
        result = prime * result
                + ((password == null) ? 0 : password.hashCode());
        result = prime * result + ((userId == null) ? 0 : userId.hashCode());
        return result;
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (obj == null)
            return false;
        if (getClass() != obj.getClass())
            return false;
        User other = (User) obj;
        if (password == null) {
            if (other.password != null)
                return false;
        } else if (!password.equals(other.password))
            return false;
```

```
        if (userId == null) {
            if (other.userId != null)
                return false;
        } else if (!userId.equals(other.userId))
            return false;
        return true;
    }

    @Override
    public String toString() {
        return "User [userId=" + userId + ", password=" + password + "]";
    }

}
```

11.2.3 Lombok setup

Inclusion of dependent library

In order to use a class that is offered by Lombok, add Lombok as dependency library.

```
<!-- (1) -->
<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <scope>provided</scope> <!-- (2) -->
</dependency>
```

Sr.No	Description
(1)	Add Lombok dependent library in the Lombok targeted project's `pom.xml` .
(2)	Since Lombok library is not required at the time of application execution, appropriate scope is `provided`.

Note: In the configuration example above, since it is assumed to manage the dependent library version with the parent project terasoluna-gfw-parent, it is not necessary to specify pom.xml version. The above dependency library used by terasoluna-gfw-parent is defined by Spring IO Platform.

IDE Integration

If you want to use Lombok on IDE, it is necessary to install the Lombok to IDE in order to work with compile (build) function provided by the IDE.

In this guideline, introduced how to install Lombok to Spring Tool Suite (Later referred as the "STS"). However installation methods are different depending on IDE henceforth refer this page in case you want to use IDE besides STS.

Download Lombok

Download the jar file of Lombok.

The jar file of Lombok,

- Download page of Lombok
- Local repository of Maven can retrieve from (Normally, `$HOME/.m2/repository/org/projectlombok/lombok/`

Lombok Installation

Launch the installer by running (double-click) the downloaded Lombok jar file.

After selecting the targeted STS, follow installation process by pressing the "Install / Update" button. The installer will automatically detect the location of supported IDE However if cannot auto detected, it is necessary to specify the IDE by pressing the "Specify location ..."

Once Lombok installation completes, it is possible to start development using Lombok on STS after booting (Or re-booting) STS.



Figure.11.2 **Lombok Installer**

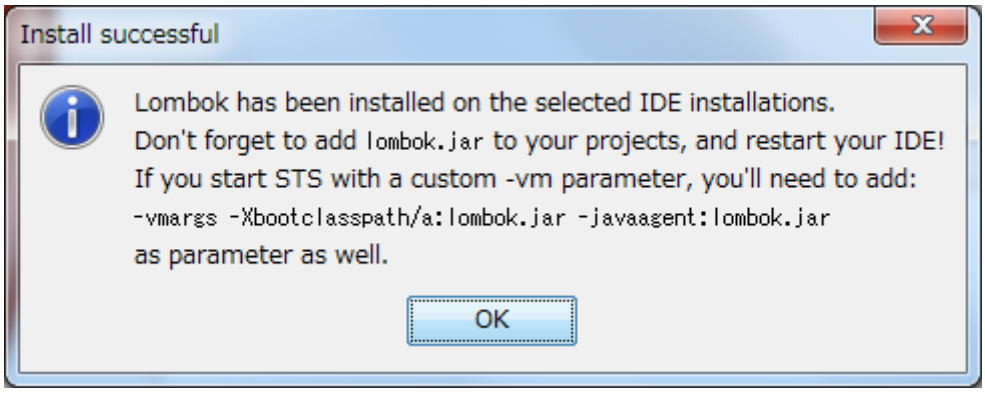


Figure.11.3 **Successful installation dialog**

11.2.4 How to use Lombok

From here, the specific use of Lombok is described.

If Lombok is used first time, it is recommendation to watch Lombok [Demo Video]. The length of Demo Video is less than 4 minutes and described the most basic usage.

Lombok Annotations

Typical annotations provided by Lombok are introduced below.

For Detailed usage of each annotation as well as annotation that have not been explained in this guideline,

please refer,

- Lombok features

Sr.No	Annotation	Description
1.	@lombok.Getter	Annotation for generating getter method. If annotation is specified at class level, it is possible to generate getter methods for all respective fields.
2.	@lombok.Setter	Annotation for generating setter method. If annotation is specified at class level, it is possible to generate setter methods for all non-final respective fields.
3.	@lombok.ToString	Annotation for generating `toString` method.
4.	@lombok.EqualsAndHashCode	Annotation for generating `equals` and `hashCode` method.
5.	@lombok.RequiredArgsConstructor	Annotation for generating constructor with the required arguments for those fields (final field etc) where initialization required. If all fields are optional fields, the default constructor (without argument constructor) is generated.
6.	@lombok.AllArgsConstructor	Annotation for generating constructor having all fields in arguments.
7.	@lombok.NoArgsConstructor	Annotation for generating default constructor.
8.	@lombok.Data	Short cut annotation for `@Getter`, `@Setter`, `@ToString`, `@EqualsAndHashCode`, `@RequiredArgsConstructor`. If `@Data` annotation is specified, it has the same meaning as specification of all above five annotations.
9.	@lombok.extern.slf4j.Slf4j	Annotation for generating logger instance of SLF4J.

Creation of JavaBean

If an application is built the way that this guideline is recommended,

- Form class
- Resource class(REST API configured)
- Entity class
- DTO class

it is necessary to create a JavaBean for above classes.

Below is the example of creating a JavaBean.

```
package com.example.domain.model;

import lombok.Data;

@Data // (1)
public class User {

    private String userId;
    private String password;

}
```

Sr.No	Description
(1)	By assigning a `@Data` annotation at class level, • getter/setter method • `equals`/ `hashCode` method • `toString` method • default constructor are created.

How to exclude specific field from toString

At the time of converting the state of object into a string,

- Field that holds an object of cross reference relationship
- Field that holds sensitive information such as personal information and password

etc are required to exclude from the scope of string conversion. If these fields are not excluded from the string conversion,

- The `StackOverflowError` and `OutOfMemoryError` occurs due to circular reference
- There is a possibility of leakage the personal information due to use of converted string

Henceforth it is necessary to take an attention.

Warning: If `@Data` or `@ToString` annotation is used at the Entity class of JPA, it is necessary to keep in mind that it tends to the circular reference.

How to exclude a specific field from string conversions are indicated below.

```
package com.example.domain.model;

import lombok.Data;
import lombok.ToString;

@Data
@ToString(exclude = "password") // (1)
public class User {

    private String userId;
    private String password;

}
```

Sr.No	Description
(1)	Specify the `@ToString` annotation to the class level and list the name of fields that you want to exclude into `exclude` attribute. If you call `toString` method of the class that is generated from the source code of above example, • `User(userId=U00001)` is converted to the string.

How to exclude specific field from equals and hashCode

If `equals` method and `hashCode` method generated using Lombok annotation, field that holds an object of cross reference relationship needs to be removed.

If methods are generated without excluding these fields, the `StackOverflowError` and `OutOfMemoryError` occurs due to circular reference henceforth it is necessary to take an attention.

Warning: If `@Data` annotation, `@Value` annotation, `@EqualsAndHash` annotation is used at the Entity class of JPA, it is necessary to keep in mind that it tends to the circular reference.

How to exclude a specific field is indicated below.

```
package com.example.domain.model;

import java.util.List;

import lombok.Data;

@Data
public class Order {

    private String orderId;
    private List<OrderLine> orderLines;

}
```

```
package com.example.domain.model;

import lombok.Data;
import lombok.EqualsAndHashCode;
import lombok.ToString;

@Data
@ToString(exclude = "order")
@EqualsAndHashCode(exclude = "order") // (1)
public class OrderLine {

    private Order order;
    private String itemCode;
    private int quantity;

}
```

Sr.No	Description
(1)	Specify the `@EqualsAndHashCode` annotation to the class level and list the name of fields that you want to exclude into `exclude` attribute.

Tip: Instead of specifying the field to be excluded, it is also possible to specify to use only specific fields.

```
@Data
@ToString(exclude = "order")
@EqualsAndHashCode(of = "itemCode") // (2)
public class OrderLine {

    private final Order order;
    private final String itemCode;
    private final int quantity;
```

```
}
```

Sr.No	Description
(2)	In case of using only specific fields, specify the name of fields that you want to include into `of` attribute of `@EqualsAndHashCode` annotation. In the above example, `equals` method and `hashCode` method get generated by referring only `itemCode` field.

How to generate constructor for field initialization

If you want to create an instance of JavaBean from the implementation code of application, it is more convenient that having constructor where initial value of the field can be passed as an argument, to eliminate the redundant code.

If you create an instance using the default constructor, the code would be like below.

```
public void login(String userId, String password) {
    User user = new User();
    user.setUserId(userId);
    user.setPassword(password);
    // ...
}
```

How to generate a constructor that specifies the initial values of the field are indicated below.

```
package com.example.domain.model;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import lombok.ToString;

@Data
@AllArgsConstructor // (1)
```

```
@NoArgsConstructor  // (2)
@ToString(exclude = "password")
public class User {

    private String userId;
    private String password;

}
```

```
public void login(String userId, String password) {
    User user = new User(userId, password); // (3)
    // ...
}
```

Sr.No	Description
(1)	Specify the `@AllArgsConstructor` annotation to the class level, to generate a constructor that takes the initial values of all fields as an argument.
(2)	Specify the `@NoArgsConstructor` annotation to the class level, to generate a default constructor. It is necessary to generate a default constructor if going to be used as JavaBean.
(3)	Create an instance of JavaBean by calling the constructor that having initial values of the field. If default constructor is used, instance can be generated in one step instead of 3 steps.

Tip: If you want to create above `User` class as Immutable class instead of JavaBean, it is preferable to use `@lombok.Value` annotation. Please refer Lombok reference for `@Value` annotation.

Creating logger instance

If it is necessary to generate a logger instance for output a debug log and application log, it is preferable to use annotations for creating a logger instance.

If you want to create a logger instance without using Lombok annotations, below could be code.

```
package com.example.domain.service;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
```

```
import org.springframework.stereotype.Service;

@Service
public class AuthenticationService {

    private static final Logger log = LoggerFactory.
↪getLogger(AuthenticationService.class);

    public void login(String userId, String password) {
        log.info("{} had tried login.", userId);
        // ...
    }

}
```

How to create a logger instance using Lombok annotation is described below.

```
package com.example.domain.service;

import org.springframework.stereotype.Service;

import lombok.extern.slf4j.Slf4j;

@Slf4j // (1)
@Service
public class AuthenticationService {

    public void login(String userId, String password) {
        log.info("{} had tried login.", userId); // (2)
        // ...
    }

}
```

Sr.No	Description
(1)	Generate SLF4J logger instance by specifying `@Slf4j` annotation at the class level. In this guideline, it is prerequisite to output a log using `org.slf4j.Logger` of SLF4J. By default, FQCN class that granted the annotation (In above example `com.example.domain.service.LoginService`) is used as the logger name and, logger instance corresponding to the logger name is set to field called `log`.
(2)	Output the log by calling the method of SLF4J logger instance that has been generated by Lombok. In above example, • `11:29:45.838 [main] INFO c.e.d.service.AuthenticationService -U00001 had tried login.` will be output.

Tip: If you want to change the logger name that is used by default, specify optional logger name in the `topic` attribute of `@Slf4j` annotation.

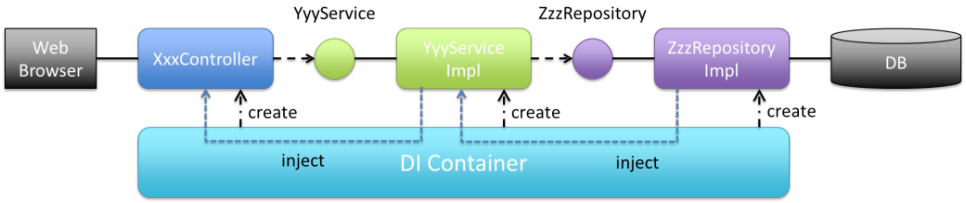
11.3 Reference Books

This guideline was prepared after referring to the following books. Refer to them as per the requirements.

Book name	Publisher	Remarks
Spring 徹底入門	Shoeisha Co., Ltd	Japanese
Pro Spring 4th Edition	APress	
Pro Spring 3	APress	
Pro Spring MVC: With Web Flow	APress	
Spring Persistence with Hibernate	APress	
Spring in Practice	Manning	
Spring in Action, Third Edition	Manning	
Spring Data Modern Data Access for Enterprise Java	O'Reilly Media	
Spring Security 3.1	Packt Publishing	
Spring3 入門—Java フレームワーク・より良い設計とアーキテクチャ	Gihyo Digital Publishing	Japanese
Beginning Java EE 6 GlassFish 3 で始めるエンタープライズ Java	Shoeisha Co., Ltd	Japanese
Seasar2 と Hibernate で学ぶデータベースアクセス JPA 入門	Mynavi Corporation	Japanese

11.4 Spring Framework Comprehension Check

1. Fill (1)-(4) such that the Bean dependency relation is as follows. Skip import statement.




```
@Controller
public class XxxController {
  (1)
  protected (2) yyyService;

  // omitted
}
```

```
@Service
@Transactional
public class YyyServiceImpl implements YyyService {
  (1)
  protected (4) zzzRepository;

  // omitted
}
```

Note: `@Service` , `@Controller` are `org.springframework.stereotype` package annotations and `@Transactional` is the annotation of `org.springframework.transaction.annotation`.

---

2. Explain when to use `@Controller`, `@Service`, and `@Repository` respectively.

Note: Each of them is `org.springframework.stereotype` package annotation.

---

3. Explain the difference between `@Resource` and `@Inject`.

Note: `@Resource` is `javax.annotation` package annotation, and `@Inject` is `javax.inject` package annotation.

---

4. Explain the difference between singleton scope and prototype scope.

5. Fill (1)-(3) in the following Scope related description. However, either of “singleton” or “prototype” can be entered in (1) and (2), but same value cannot be entered for both. Skip the import statement.

```
@Component
(3)
public class XxxComponent {
  // omitted
}
```

Note: `@Component` is `org.springframework.stereotype.Component`.

Scope of bean with `@Component` is (1) by default. When changing scope to (2), it is better to add (3) (refer to above source code).

6. In case of following Bean definition, what type of Bean is registered in DI container?

```
<bean id="foo" class="xxx.yyy.zzz.Foo" factory-method="create">
    <constructor-arg index="0" value="aaa" />
    <constructor-arg index="1" value="bbb" />
</bean>
```

7. Fill (1)-(3) of the following Bean definition such that contents in `com.example.domain` package becomes target of component scan.

```
<context:(1)  (2)="(3)" />
```

Note: Bean definition file should include the definition of

xmlns:context=”http://www.springframework.org/schema/context”

---

8. Fill (1)-(2) in the following Properties file related description. Skip import statement.

   It is possible to read the Bean definition file in `${key}` format by removing the settings in properties file, if properties file path is set in the `locations` attribute of `<context:property-placeholder>` element. Specify as shown in (1) to read any properties file under META-INF/spring directory under the class path. Moreover, @(2) annotation should be added as shown in the following codes where the read properties value can also be injected in Bean.

```
<context:property-placeholder locations="(1)" />
```

```
emails.min.count=1
emails.max.count=4
```

```
@Service
@Transactional
public class XxxServiceImpl implements XxxService {
  @xxx("${emails.min.count}") // (2)
  protected int emailsMinCount;
  @xxx("${emails.max.count}") // (2)
  protected int emailsMaxCount;
  // omitted
}
```

Note: Bean definition file should include the definitions of

xmlns:context="http://www.springframework.org/schema/context"

---

9. Fill (1)-(5) in the following description for AOP Advice of Spring. The contents of (1)-(5) are all different.

Note: Advice (1) should be used when interrupting a process before calling a specific method, Advice (2) should be used when interrupting a process after calling a specific method. Advice (3) should be used when interrupting a process before and after calling a specific method. Advice (4) should be used only when the process is ended normally and Advice (5) should be used when there is an exception.

---

10. Insert (*) of following Bean definition for performing transaction management using `@Transactional` annotation.

```
<tx:(*) />
```

Note: Bean definition file should include the definitions of

xmlns:tx="http://www.springframework.org/schema/tx"
