intra-mart Accel Platform

IM-JavaEE Framework Specification

2012/10/01 Initial Version

<<Revision History>>

Revision Date	Updated Contents
2012/10/01	Initial Version

<<Table of Contents>>

1 Introduction

1.1 Purpose

This document describes detail specification of IM-JavaEE Framework.

1.2 Scope of this Document

This document describes the specifications of jm-JavaEE Framework that comes with intra-mart AccelPlatform. Each limitation or operation environment is depending on these.

This document describes interface and external specifications of IM-JavaEE Framework. Internal specifications such as how to use component and its procedure are partially supported, but there is no describtion about internal implementation.

This document presents multiple samples to supplement the description. To understand the specifications for each section or chapter comes as the primary purpose, it does not indicate optimal solutions as program or system. Therefore, some exeptional processes are not implemented purposely.

This document contains helpful information even if the readers are related to the upper process, but detailed method such as project management or business analysis.

This document does not describe JavaTM Standard Edition (J2SE) or JavaTM Enterprise Edition (JavaEE) in detail.

1.3 Target Readers of this Document

Following users are the target of this document.

- Designer
  What type of interface specifications are required and how to divide the component are presented for designers.
- Implementor
  Information to create component or its specifications are presented for implementors.

For project managers and business analyst, this document does not present any specific information. It goes only for the reference when they do analysis or management specifically.

Certain understanding for the following is preferred.

- JavaTM Standard Edition
- JavaTM Enterprise Edition
- UML
- Design Pattern
- XML 1.1[3]

1.4 Structure

This document has following structures.

- In [2 Application Structure], the structure when creating application in IM-JavaEE Framework is indicated.
- In [3 Service Framework] - [7 Log Framework], detail operations or interface and how to create the component for sub framework of IM-JavaEE Framework are described.

2 Application Structure

2.1 Overview

IM-JavaEE Framework will be the structure platform of middle to large Web system. Applications operated on the IM-JavaEE Framework are structured according to BluePrints[4] that Sun Microsystems advocates.

2.2 Structure

2.2.1 Overall

Application structure is described as [Figure 2-1 Application Structure].

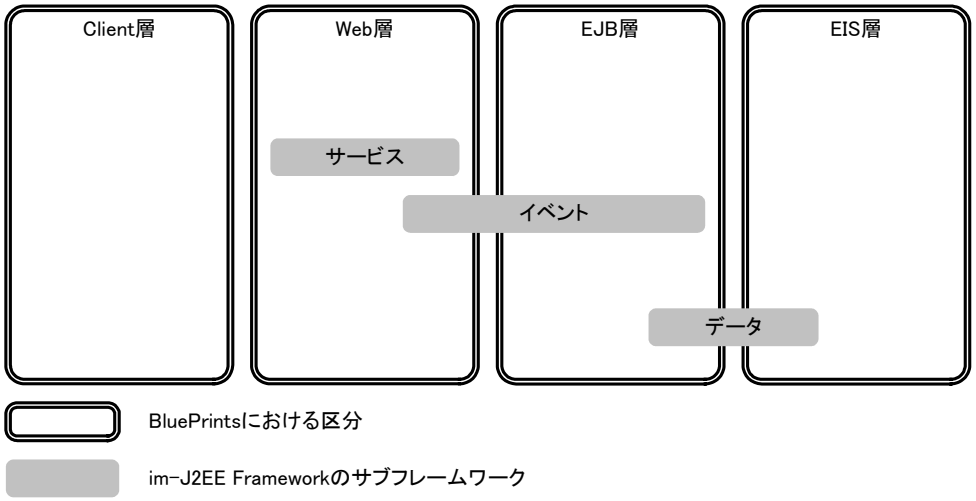


Figure 2-1 Application Structure

Each sub framework overview is described as below.

- Service framework (refer to [3 Service Framework])
  Receive the request from client (mainly from browser) and do screen transition and display.
- Event framework (refer to [4 Event Framework])
  Do some process.
- Data framework (refer to [5 Data Framework])
  Link to the EIS layer (data base or legacy system) and exchange data.

Followings are sub framework related to the system overall.

- Message frame work (refer to [2.2.5 Message Framework])
  Obtain messages that have region support
- Log framework (refer to [7 Log Framework])
  Do log output.

Each sub framework has collaborative relationship as shown in [Figure 2-2 Collaorative Relationship between frameworkss]

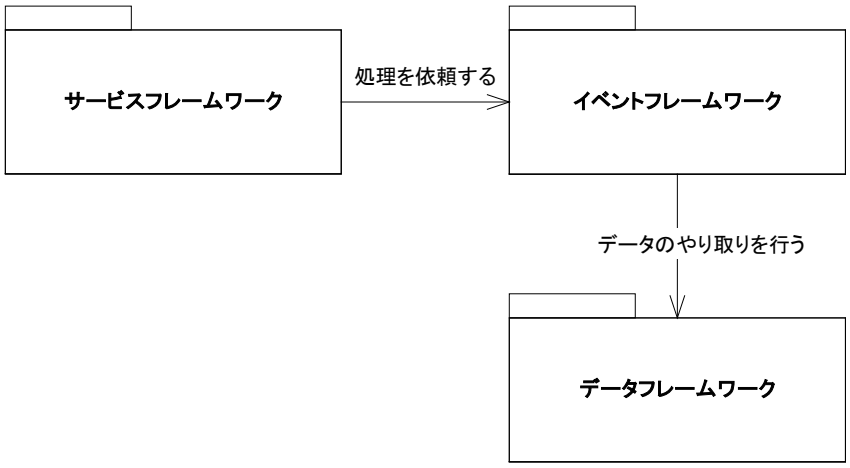


Figure 2-2 Collaorative Relationship between frameworks

2.2.2 Service Framework

In the service framework, the request from client (mainly from broswer) will be processed in Web layer. Service framework receives the input (request from broswer), assigns the necessary process for the request and controls transition for the next screen. Detail process for the request is not described here, do the process in the event framework is recommended.

2.2.3 Event Framework

In the event framework, do some process for the input contents and it has role to control business logic that returns the process result.It is also possible to incorporate business logic in service framework but better to use event framework for a general use.

2.2.4 Data Framework

Data framework links with the EIS layer (data base or legacy system) and exchanges the data. Generally, accessing to the data storewould be different, but data framework helps to provide how to do coding without considering access method from the caller.

2.2.5 Message Framework

Message framework obtains the messages that have region support. In internationalized applications, the way to obtain character string that has region support based on each area locale is provided.

2.2.6 Log Framework

In log framework, log is output. There are various timings, contents, destinations, and format in log output, log framework provide the method to manage them separately.

2.2.7 Property

IM-JavaEE Framework has the concept of application. Application is a unit to integrate multiple components. Application is associated with component by property setting. Property setting information is obtained through PropertyHandler. PropertyHandler is prepared for each sub framework and there is one for application specific as well as one for all applications.

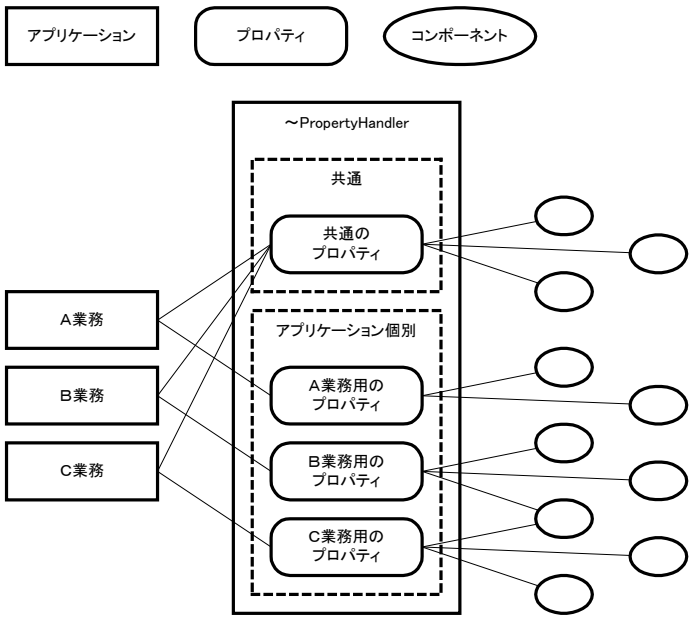


Figure 2-3 PropertyHandler

2.2.7.1 Application

Application is a unit to integrate relationships with multiple components. Individual application is identified by application ID or application name and it will be the base unit of IM-JavaEE Framework. If you obtain property of individual application from PropertyHandler, application ID or application name should be specified.

2.2.7.2 Common

Some properties are not depending on application. There is no application ID or application name in these propertoes. If you obtain property for common application from PropertyHandler, no need to specify application ID or application name.

3 Service Framework

3.1 Overview

When Web system is structured, utilization form from users is generally will be 1) send information to the server from browser, 2) display the process result at server side.

In order to make input and display support multiple areas, internationalization of the application is required.

In service framework, common parts in this series of processes are made into frameworks.

3.2 Structure

3.2.1 Structure Element

In IM-JavaEE Framework, it receives some request from Web client (mainly from browser) and the screen transition for the request is handled by the unit called [Service]. Service can be identified uniquely with the combination of application ID and service ID. For the individual services, component called ServiceController and Transition can be associated to maximum one for each, and more transition destinations can be associated. This relationship is shown in [Figure 3-1 Service Structure].

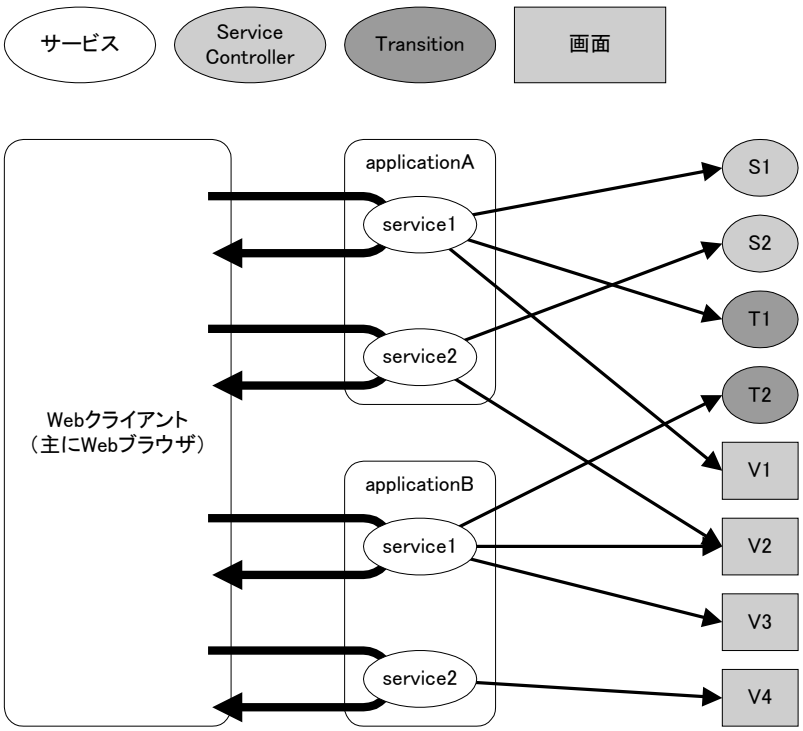


Figure 3-1 Service Structure

Followings are indicated in [Figure 3-1 Service Structure].

- Service that has applicationA as application ID and service1 as service ID is associated with S1 as ServiceController, T1 as Transition, and V1 as transition destination screen.
- Service that has application A as application ID and service2 as service ID is associated with S2 as ServiceControllerm and V2 as transition destination screen, but it is not associated with Transition.
- Service that has applicationB as application ID and service1 as service ID is associated with T2 as Transition and V2 and V3 as the candidate of transition screen, but it is not associated with ServiceController.
- Service that has applicationB as application ID and service2 as service ID is not associated with neither of ServiceController or Transition, but it is associated with V4 as transition destination screen.

3.2.1.1 Service

Service is the most basic unit in the service framework of IM-JavaEE Framework. Each service is uniquely determined by application ID and service ID.

3.2.1.2 ServiceController

ServiceController has a role to do input check and process for the request. ServiceController detail is described in [3.5.5 Process].

3.2.1.3 Transition

Transition has a role to do the preparation when transition to the next screen and do the actual transition. Transition detail is described in [3.5.6 Transition Process].

3.2.2 Operation

Service framework of IM-JavaEE Framework starts each component for the requests by following sequence.

1. Obtain application ID and service ID from the request.
2. Determine the service corresponding to the application ID and the service ID.
3. If ServiceController is associated with the service, input process will be started for the ServiceController.
4. If Transition is associated with the service, transition process will be started for the Transition.

Detail is described in [3.5 Screen Transition].

3.3 Preprocessing at Request Time

When the request is sent to the Web client (mainly from Web browser), several preprocessing are made in IM-JavaEE Framework before it goes to various processes. Preprocessing uses filtering function with Servlet 2.3 specifications. Preprocessing contents after moved to the IM-JavaEE Framework screen will be as [Figure 3-2 Preprocessing at Request Time].

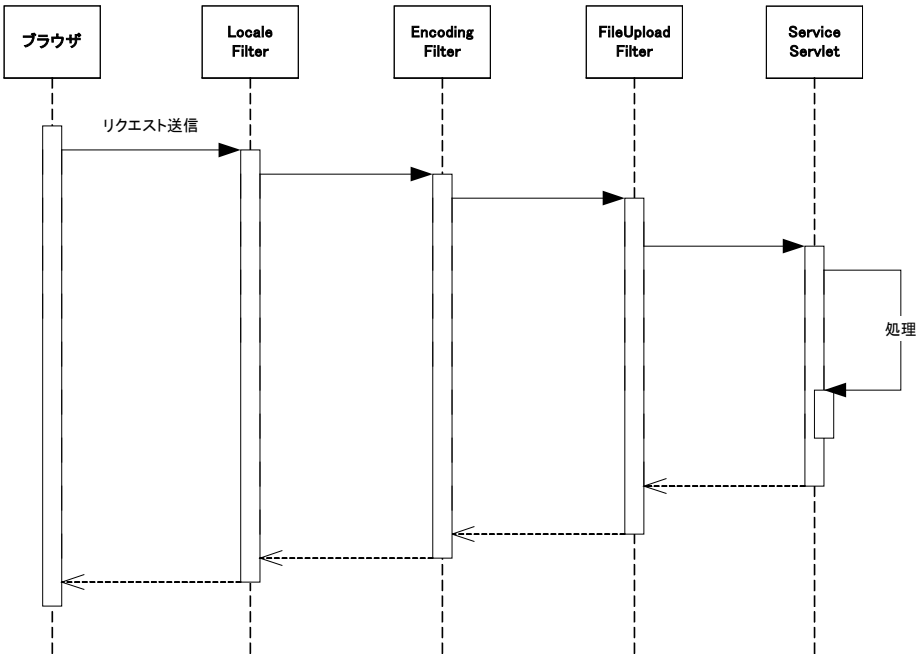


Figure 3-2 Preprocessing at Request Time

3.3.1 Locale

If internationalized application is created, current request supports which locale shouled be determined. If the same user uses the application continuously, normally the locale is not be changed so it is registered to the session for the better use in many cases.

In IM-JavaEE Framework, locale filter is prepared to do these procedure simply. Locale filter determins and obtains current locale through getLocale method of jp.co.intra_mart.framework.base.service.ServiceManager, and registers the contents in javax.servlet.http.HttpSession.

3.3.1.1 Operation

jp.co.intra_mart.framework.base.service.LocaleFilter determins and obtains the locale of the request, and registers the contents in HttpSession. This filter operates as shown in [Figure 3-3 LocaleFilter Process].

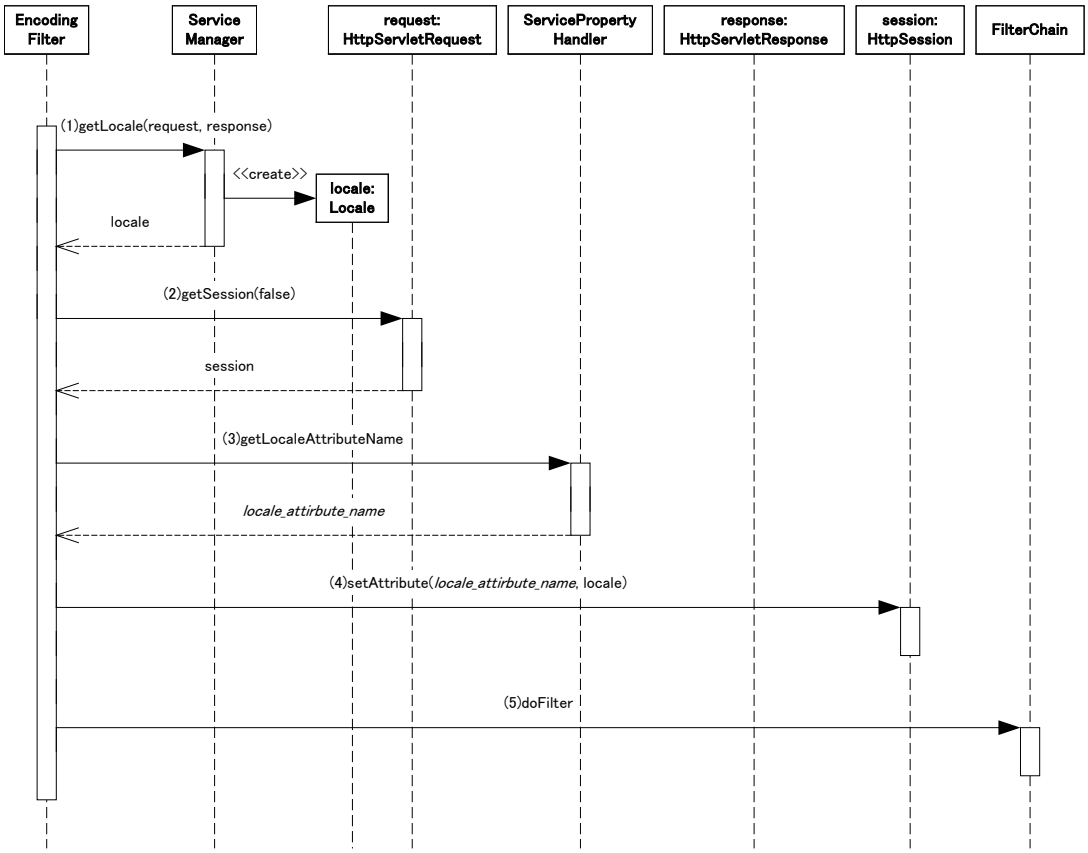


Figure 3-3 LocaleFilter Process

Process contents of locale filter is shown below.

1. Obtain locale for the current request by using getLocale method of ServiceManager.
2. Obtain HttpSession. If there is no HttpSession, further process for the locale will not be performed and pass the process to the next filter.
3. Obtain attribute name when you set locale of client to the session through ServicePropertyHandler (by using getLocaleAttributeName method).
4. Set obtained locale to HttpSession.
5. Pass the process to the next filter.

3.3.1.2 How to determine Locale

In the getLocale method of ServiceManager, locale should be obtained by following sequence. Locale is determined when values other than null is obtained, and it returns the value.

1. Locale registered in javax.servlet.http.HttpSession (refer to [3.3.1.2.1 Locale registered in HttpSession)
2. Locale obtained from ServicePropertyHandler (refer to [3.3.1.2.2 Locale obtained from ServicePropertyHandler])
3. Locale obtained by getLocale method of javax.servlet.ServletRequest
4. Locale obtained by getDefault method of java.util.Locale

If you determine locale by original method, you can set locale to HttpSession in advance. By setting this, since locale filter meets condition (1), locale will be determined in the subsequent access. Even if in the middle of access or process, changed locale will be used for the furthur process if you set this value. For example, if you want to change locale for each log-in user, you can set locale in HttpSession by log-in process. Even if in the middle of accessing after log-in, the process will be continued with new locale in the further access if you change this value by requesting process after locale change.

3.3.1.2.1 **Locale registered in HttpSession**

In this method, locale should be obtained from HttpSession attribute. The attribute name is the one obtained by getLocaleAttributeName method (refer to [3.4.4.1.5 Locale Attribute Name]) of ServicePropertyHandler.

3.3.1.2.2 **Locale obtained from ServicePropertyHandler**

In this method, locale should be obtained by using getClientLocale method (refer to [3.4.4.1.4 Client Locale] of ServicePropertyHandler.

3.3.1.3 Locale Filter Application

Following settins are required for locale filter use.

- web.xml setting
- Locale setting

3.3.1.3.1 **web.xml Setting**

Locale filter uses Filter function of Servlet 2.3. Following class should be set to web.xml as Filter in order to enable this filter.

- jp.co.intra_mart.framework.base.service.LocaleFilter

If you install intra-mart, following setting is performed as standard.

- It is executed before the request is passed to jp.co.intra_mart.framework.base.service.ServiceServlet.

3.3.1.3.2 **Locale Setting**

When you determine locale of request that are sent from client uniquely, you should set in the item defined in [3.4.4.1.4 Client Locale]. Setting method is depending on ServicePropertyHandler to be used, so please refer to related documents.

3.3.2 Encoding

Many of current Web browser does not send encoding when it is sent. In this case, encoding from the client should be processed as ISO-8859-1 in Servlet (refer to [SRV.4.9 Request data encoding] of [JavaTM Servlet Specification Version 2.3] [5]. Therefore, sometimes you cannot obtain correct character string even if you try to obtain the contents sent by using getParameter of javax.servlet.ServletRequest.

Encoding filter is prepared in IM-JavaEE Framework to solve this issue. Encoding filter sets client's encode through setCharacterEncoding of javax.servlet.ServletRequest の setCharacterEncoding. Because of this, you don’t have to do encoding conversion even if getParameter is used.

3.3.2.1 Operation

jp.co.intra_mart.framework.base.service.EncodingFilter determines what encoding type was used for sending the request contents. This filter operates as shown in [Figure 3-4 EncodingFilter Process].

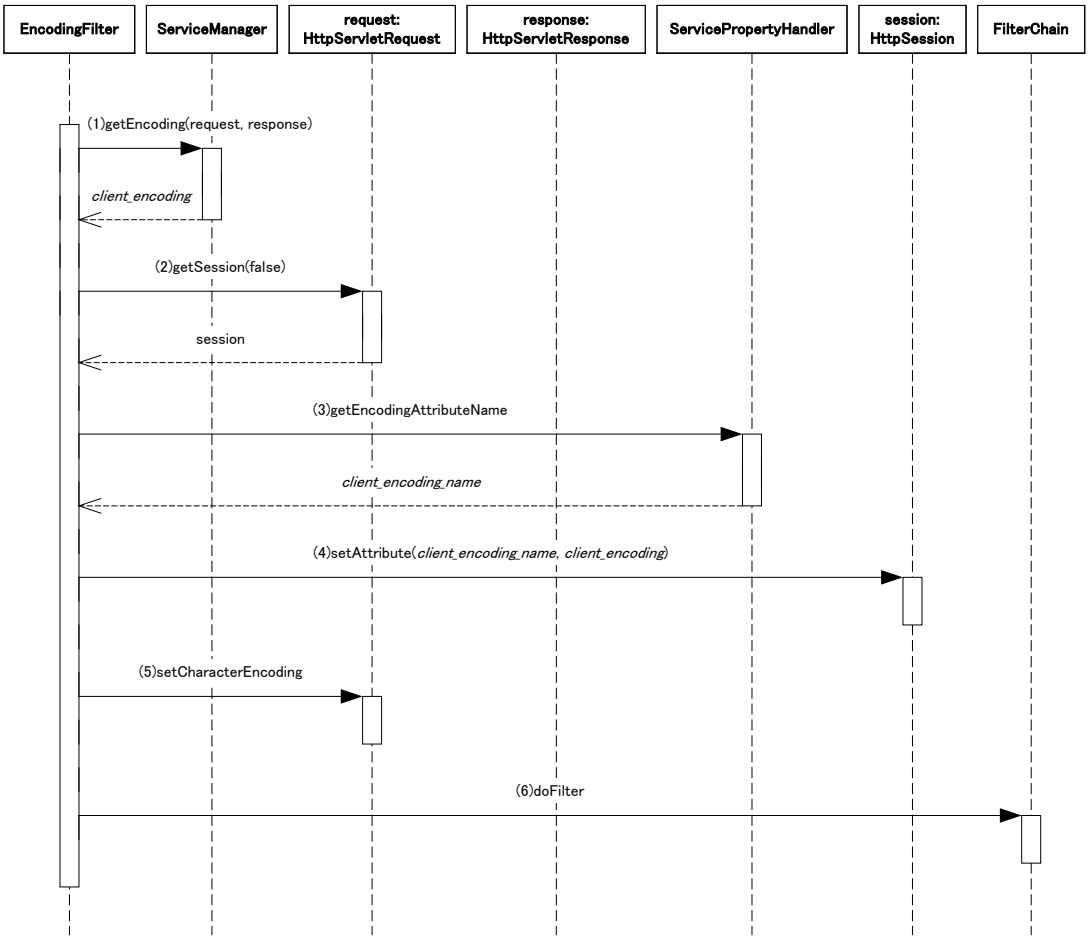


Figure 3-4 EncodingFilter Process

Process contents of encoding filter are described below.

1. Obtain encoding for the request by using getEncoding method of ServiceManager.
2. Obtain HttpSession. If there is no HttpSession, the process of (3)(4) will not be performed.
3. Obtain the attribute name when you set client's encoding to the session through ServicePropertyHandler (by using getEncodingAttributeName method).
4. Set obtained encoding to HttpSession.
5. Set obtained encoding to ServletRequest.
6. Pass the process to the next filter

3.3.2.2 How to determine Encoding

In getEncoding method of ServiceManager, encoding will be obtained by follwing below sequences. Encoding is determined when the value other than null is obtained, and it returns the value.

1. Encoding registered in javax.servlet.http.HttpSession (refer to [3.3.2.2.1 Encoding registered in HttpSession])
2. Encoding obtained from ServicePropertyHandler (refer to [3.3.2.2.2 Encoding obtained from ServicePropertyHandler])
3. Locale obtained by getCharacterEncoding method of javax.servlet.ServletRequest.

3.3.2.2.1 Encoding registered in HttpSession

In this method, encoding should be obtained from the attribute of HttpSession. The attribute name is the one that should be obtained by getEncodingAttributeName method of ServicePropertyHandler (refer to [3.4.4.1.3 Encoding Attribute Name].

3.3.2.2.2 **Encoding obtained from ServicePropertyHandler**

In this way, locale should be obtained by using getClientEncoding method (refer to [3.4.4.1.2 Client Encoding] of ServicePropertyHandler.

3.3.2.3 Encoding Filter Application

Following settings are required to use encoding filter

- web.xml setting
- Encoding setting

3.3.2.3.1 **web.xml Setting**

Encoding filter uses Filter function of Servlet 2.3. Following class should be set to web.xml to enable this filter.

- jp.co.intra_mart.framework.base.service.EncodingFilter

If you install intra-mart, follwings are set as standard.

- It is executed before the request passed to jp.co.intra_mart.framework.base.service.ServiceServlet.
- It is executed first in the filter related to above servlet.

3.3.2.3.2 **Encoding Setting**

Encoding setting of request that is sent from the client should be performed by the item defined in [3.4.4.1.2 Client Encoding]. Setting method is depending on the ServicePropertyHandler to be used, so please refer to the related documents.

3.3.3 File Upload

When you receive the request from browser that uploads file, followings procedures are the general way to obtain file data:

1. Obtain input stream (by getInputStream of javax.servlet.ServletRequest) from the request.
2. Obtain file data from the obtained input stream.

However, if you obtaine the stream from the request once, method (getParameter etc.) related to parameter obtain will not be able to be used. You can obtain parameter information by originally analize the input stream contents but certain efforts must be required.

In IM-JavaEE Framework, file upload filter is prepared to solve this problem. When file upload filter receives the file upload request, it not only provides the method (getParameter etc.) to obtain the parameter as the same as the normal request but also provides the method to obtain the uploaded file.

3.3.3.1 Operation

jp.co.intra_mart.framework.base.service.FileUploadFilter provides simple method to obtain both parameters and uploaded files by jp.co.intra_mart.framework.base.service.ServiceControllerAdapter subclass when the request is a file upload. By using this filter, you can obtain the file uploaded by the method shown in [Figure 3-5 ServiceControllerAdapter when FileUploadFilter is used], and the same method (getParameter method etc.) as for the normal request can be used for obtaining the parameter too.

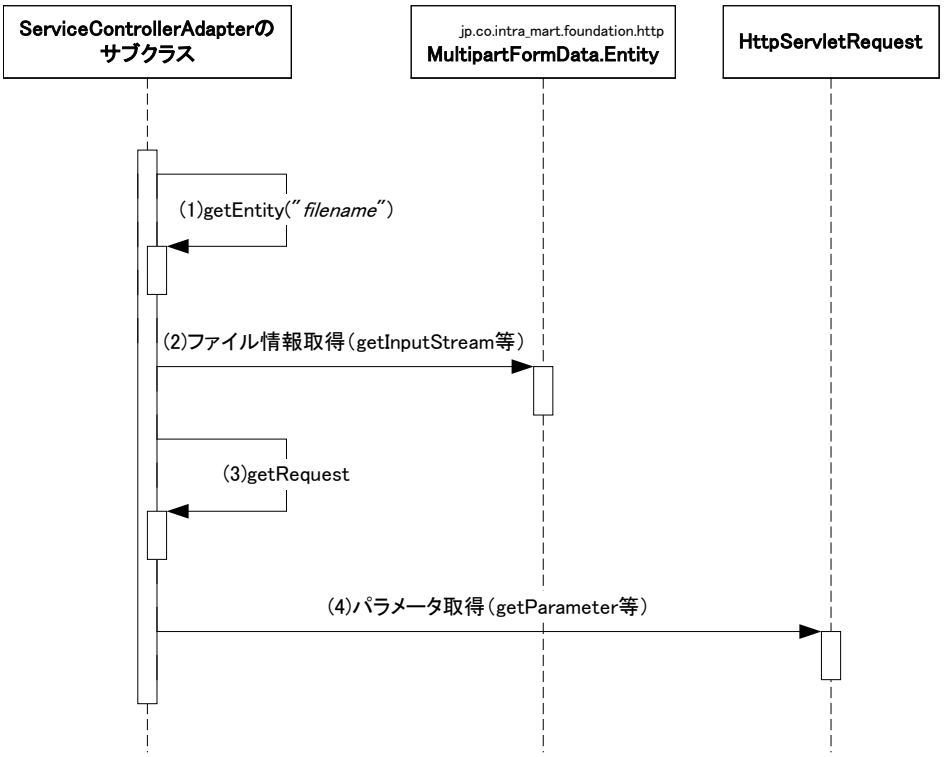


Figure 3-5 ServiceControllerAdapter when FileUploadFilter is used

Followings are the method to obtain files and parameters when you use file upload filter.

1. Obtain MultiPartFormData.Entity by using getEntity method of ServiceControllerAdapter.
2. Obtain file contents from MultiPartFormData.Entity (through getInputStream method)
3. Obtain HttpServletRequest by using getRequest method of ServiceControllerAdapter.
4. Obtain parameter of the request.

3.3.3.2 File Upload Filter Application

Following settings are required to use file upload filter.

- Request from the browser
- web.xml setting

3.3.3.2.1 Request from the Browser

File upload filter is applied only when all the following conditions are met for the request from the browser.

- Method of request should be "POST". In this case both capital and small characters can be used.
- ContentType of the requiest should be "multipart/form-data".

If the request from the browser does not meet above conditions, file upload filter does nothing.

3.3.3.2.2 web.xml Setting

File upload filter uses Filter function of Servlet 2.3. Following class should be set to web.xml as Filter in order to enable this filter.

- jp.co.intra_mart.framework.base.service.FileUploadFilter

When you install intra-mart, followings are set as standard.

- It is executed before the request passed to jp.co.intra_mart.framework.base.service.ServiceServlet.
- It is executed after encoding filter (refer to encoding filter ([3.3.2 Encoding].

Coding such as switching the requests after file upload filter is not recommended. In this case, the operation of getEntity of ServiceControllerAdapter is not guaranteed. Followings are examples when not recommended.

- Original HttpServletRequest should be generated when it is requested, and the filter that executes doFilter should be added, so that it is executed after the upload filter.
- It generates original HttpServletRequest after upload filter execution and does forward.

3.4 Property related to the Service

Various properties can be set externally in service framework of IM-JavaEE Framework. Service property will be obtained from the class that implements jp.co.intra_mart.framework.base.service.ServicePropertyHandler interface. In IM-JavaEE Framework provides multiple implementation classes that implement this interface as standard (refer to [Figure 3-6 ServicePropertyHandler]). Setting method of service property is not specified in IM-JavaEE Framework and it depends on the class that implements the interface described before.

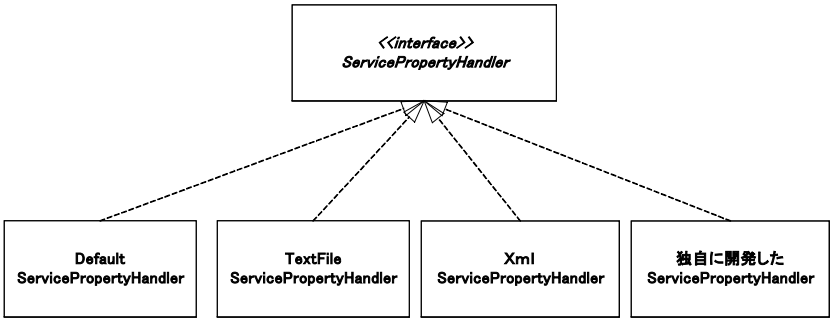


Figure 3-6 ServicePropertyHandler

3.4.1 Obtaining Property about the Service

Property relaated the service is obtained from ServicePropertyHandler. ServicePropertyHandler can be obtained by getServicePropertyHandler method of jp.co.intra_mart.framework.service.ServiceManager. ServicePropertyHandler must be obtained through this method and the developers should not generate the implementation class of this ServicePropertyHandler explicitly by themselves (generation by new, newInstance method of java.lang.Class, or instance that uses reflection).

Procedures related to obtaining ServicePropertyHandler and property are shown in [Figure 3-7 Obtaining ServicePropertyHandler].

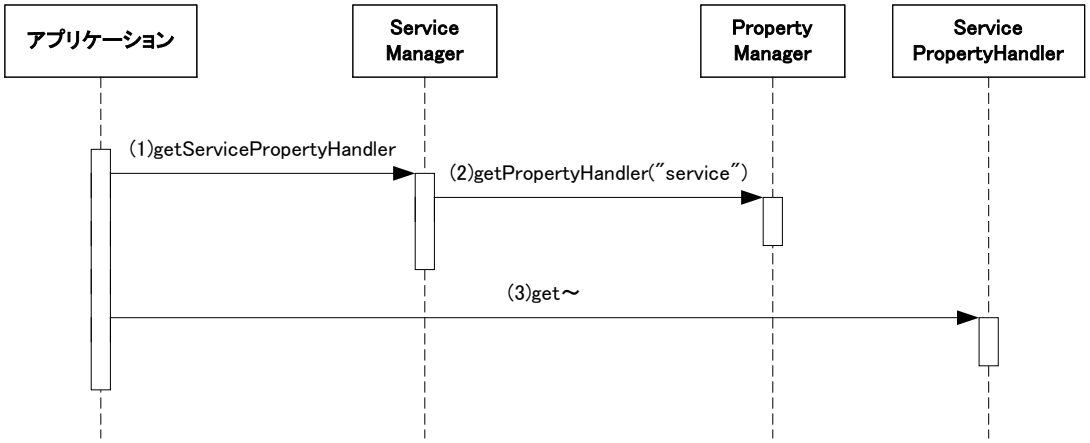


Figure 3-7 Obtaining ServicePropertyHandler

1. Obtain ServicePropertyHandler from ServiceManager.
2. Obtain ServicePropertyHandler from PropertyManager in ServiceManager and it returns it to the application. This part is performed inside the ServiceManager and the developer does not have to consider this issue.
3. Obtain each property by using ServicePropertyHandler.

3.4.2 ServicePropertyHandler provided as Standard

In IM-JavaEE Framework, several classes that implement jp.co.intra_mart.framework.base.service.ServicePropertyHandler are provided. Setting method or its features are depending on each, the operator can switch thoes accordingly.

3.4.2.1 DefaultServicePropertyHandler

It is provided as jp.co.intra_mart.framework.base.service.DefaultServicePropertyHandler. Property setting is performed by resource file. The contents of resource file can be set with the format of [property name=property value]. Characters that can be used are according to the java.util.ResourceBundle.

This resource file should be placed in the class path that can be obtained from application to be used. Refer to API list for the detail of file name of resource file or property name to be sed.

3.4.2.2 TextFileServicePropertyHandler

It is provided as jp.co.intra_mart.framework.base.service.TextFileServicePropertyHandler. Resource file that has the same format with DefaultServicePropertyHandler will be used, but followings are the difference.

- No need to pass a class path.
- If the location can be accessed from the application, it can be placed at any location in the file system. Application should not be stopped when resource file is reloaded according to the setting.

Please refer to API list for the detail of file name of resource file or property to be set.

3.4.2.3 XmlServicePropertyHandler

It is provided as jp.co.intra_mart.framework.base.service.XmlServicePropertyHandler. Property setting is done in XML format. It should be placed in the class path which can be obtained by the application. Unique ID and Java package path are recognized as application ID. For example, if application ID is"foo.bar.example", it is placed as" foo/bar/service-config-example.xml" in class path.

XmlServicePropertyHandler supports dynamical loading. Please refer to API list for details.

3.4.3 Original ServicePropertyHandler

If the developer creates ServicePropertyHandler originally, following requirements must be met.

- jp.co.intra_mart.framework.base.service.ServicePropertyHandler interface is implemented.
- public default constructor (with no argument) is defined.
- Appropriate value must be returned to all methods (refer to [3.4.4 Property Contents]).
- If isDynamic() method returns false, the value of method that obtains property will not change unless application server is restarted.

3.4.4 Property Contents

Setting method of property related service is depending on the type of ServicePropertyHandler that is to be used when operation, but the concept is the same.

Most of properties (other than several exceptions) are internationalized and the value that has region support can be set.

Contents of property related to the service are described as below.

3.4.4.1 Common

3.4.4.1.1 Dinamic Loading

It can be obtained by isDynamic() method. It does not have region support.

If the return value of this method is true, each property obtaining method (get~method) defined in this interface should load setting information initially everytime as the implementation. If it is false, each property obtaining method can cache the value to be obtained internally considering the performance.

3.4.4.1.2 Client Encoding

It can be obtained by getClientEncoding() method. It does not have region support.

It sets encoding of the request contents that is sent from the browser. If it is omitted, it does not convert character code for the requests.

3.4.4.1.3 Encoding Attribute Name

It can be obtained by getEncodingAttributeName() method. It does not have region support.

It sets attribute name when saving encode used by log-in user in HttpSessionn. If it is not set, it is considered as the value of ServicePropertyHandler.DEFAULT_ENCODING_ATTRIBUTE[1] is already set.

[1] In IM-JavaEE Framework 5.0 or later, this value is "jp.co.intra_mart.framework.base.service.encoding".

3.4.4.1.4 **Client Locale**

It can be obtained by getClientLocale() method. It does not have region support.

It sets the locale used by client. If it is omitted, it considered as null is set.

3.4.4.1.5 **Locale Attribute Name**

It can be set by getLocaleAttributeName() method. It does not have region support.

It sets attribute name when saving encode used by log-in user in HttpSessionn. If it is not set, it is considered as the value of ServicePropertyHandler.DEFAULT_LOCALE_ATTRIBUTE[2] is already set.

3.4.4.1.6 **Context Path**

Remarks : This property is not recommended and better not to set.

It can be obtained getContextPath() method. It does not have region support.

It sets a context path of Web application when move IM-JavaEE Framework. Path specification must be started with "/". If it is omitted, it returns null.

3.4.4.1.7 **Exceptional Attribute Name**

It can be obtained by getExceptionAttributeName() method. It does not have region support.

It sets attribute name of request when passing exception to the exception page. If it is omitted, it returns the value of jp.co.intra_mart.framework.base.service.ServicePropertyHandler.DEFAULT_EXCEPTION_ATTRIBUTE ("exception").

3.4.4.1.8 **Default Input Error Page Path**

It can be obtained by getInputErrorPagePath() method or getInputErrorPagePath(Locale locale) method. It does not have region support.

It sets default input error page path. It is specified by relative path from context root. Path specification should start with "/". If it is not set, ServicePropertyException will be throw.

3.4.4.1.9 **Default Service Error Page Path**

It can be obtained by getServiceErrorPagePath() method or getServiceErrorPagePath(Locale locale) method. It has region support.

It sets default service error page path. It is specified by relative path from context root. Path specification should start with "/". If it is not set, ServicePropertyException will be throw.

3.4.4.1.10 **Default System Error Page Path**

It can be obtained by getSystemErrorPagePath() method or getSystemErrorPagePath(Locale locale) method. It has region support.

It sets default system error page path. It is specified by relative path from context root. Path specification should start with "/". If it is not set, ServicePropertyException will be throw.

[2] In IM-JavaEE Framework 5.0 or later, this value is "jp.co.intra_mart.framework.base.service.locale".

3.4.4.2 Application Individual

3.4.4.2.1 ServiceController

It can be obtained by getServiceControllerName(String, String) method or getServiceControllerName(String, String, Loccale) method. It has region support.

It sets ServiceController class that suports application ID and service ID. If it has not set, it returns null. The class to be specified here should implement jp.co.intra_mart.framework.base.service.ServiceController interface.

3.4.4.2.2 Transition

It can be obtained by getTransitionName(String, String) method or getTransitionName(String, String, Locale) method. It has region support.

It sets class name of Transition that supports application ID and service ID. If it has not set, it returns null. The class to be specified here should extend jp.co.intra_mart.framework.base.service.Transition class.

3.4.4.2.3 Transition Path with Key

It can be obtained by getNextPagePath(String application, String service, String key) method or getNextPagePath(String application, String service, String key, Locale) method. It has region support.

It sets path of the next transition destination that supports application ID, service ID and key. It is specified by relative path from context root. Path specification should start with "/". This is used when you want to specify the transition destination in more detailed than combination of application ID and service ID. If it is not set, ServicePropertyException will be throw.

3.4.4.2.4 Transition Path

It can be obtained by getNextPagePath(String application, String service) method or getNextPagePath(String application, String service, Locale locale) method. It has region support.

It sets path of the next transition destination that supports application ID and service ID. It is specified by relative path from context root. Path specification should start with "/". If it is not set, ServicePropertyException will be throw.

3.4.4.2.5 Input Error Page Path with Key

It can be obtained by getInputErrorPagePath(String application, String service, String key) method or getInputErrorPagePath(String application, String service, String key, Locale locale) method. It has region support.

It sets path of input error page that supports application ID, service ID and the key. It is specified by relative path from context root. Path specification should start with "/". If it is not set, the result will be the same when getInputErrorPagePath(String application, String service) is called. This is used when you want to specifiy the transition destination of input error in more detailed than the combination of application ID and service ID.

3.4.4.2.6 Input Error Page Path

It can be obtained by getInputErrorPagePath(String application, String service) method or getInputErrorPagePath(String application, String service, Locale locale) method. It has region support.

It sets path of default input error page that supports application ID and service ID. It is specified by relative path from context root. Path specification should start with "/". If it is not set, the result will be the same when

getInputErrorPagePath(String application) is called.

3.4.4.2.7 **Default Input Error Page Path**

It can be obtained by getInputErrorPagePath(String application) method or getInputErrorPagePath(String application, Locale locale) method. It has region support.

It sets default input error page path that supports application ID. It is specified by relative path from context root. Path specification should start with "/". If it is not set, the result will be the same when getInputErrorPagePath() is called.

3.4.4.2.8 **Service Error Page Path with Key**

It can be obtained by getServiceErrorPagePath(String application, String service, String key) method or getServiceErrorPagePath(String application, String service, String key, Locale locale) method. It has region support.

It sets service error page path that supports application ID, service ID and the key. It is specified by relative path from contexit root. Path specification should start with "/". If it is not set, the result will be the same when getServiceErrorPagePath(String application, String service) is called. This is used when you want to specify the transition destination of service error in more detailed than the combination of application ID and service ID.

3.4.4.2.9 **Service Error Page Path**

It can be obtained by getServiceErrorPagePath(String application, String service) method or getServiceErrorPagePath(String application, String service, Locale locale) method. It has region support.

It sets path of default service error page that supports application ID and service ID. It is specified by relative path from context root. Path specification should start with "/". If it is not set, the result will be the same when getServiceErrorPagePath(String application) is called.

3.4.4.2.10 **Default Service Error Page Path**

It can be obtained by getServiceErrorPagePath(String application) method or getServiceErrorPagePath(String application, Locale locale) method. It has region support.

It sets path of default service error page that supports application ID. It is specified by relative path from context root. Path specification should start with "/". If it is not set, the result will be the same when getServiceErrorPagePath() is called.

3.4.4.2.11 **System Error Page Path with Key**

It can be obtained by getSystemErrorPagePath(String application, String service, String key) method or getSystemErrorPagePath(String application, String service, String key, Locale locale) method. It has region support.

It sets path of system error page that supports application ID, service ID and the key. It is specified by relative path from context root. Path specification should start with "/". If it is not set, the result will be the same when getSystemErrorPagePath(String application, String service) is called. This is used when you want to specify the transition destination of input error in more detailed than the combination of application ID and service ID.

3.4.4.2.12 **System Error Page Path**

It can be obtained by getSystemErrorPagePath(String application, String service) method or

getSystemErrorPagePath(String application, String service, Locale locale) method. It has region support.

It sets path of default system error page that supports application ID and service ID. It is specified by relative path from context root. Path sprcification should start with "/". If it is not set, the result will be the same when getSystemErrorPagePath(String application) is called.

3.4.4.2.13 **Default System Error Page Path**

It can be obtained by getSystemErrorPagePath(String application) method or getSystemErrorPagePath(String application, Locale locale) method. It has region support.

It sets path of default system error page that supports application ID. It is specified by relative path from context root. Path specification should start with "/". If it is not set, the result will be the same when getSystemErrorPagePath() is called.

3.5 Screen Transition

Process overview when screen transition is performed in IM-JavaEE Framework is shown in [Figure 3-8 Screen Transition Overview].

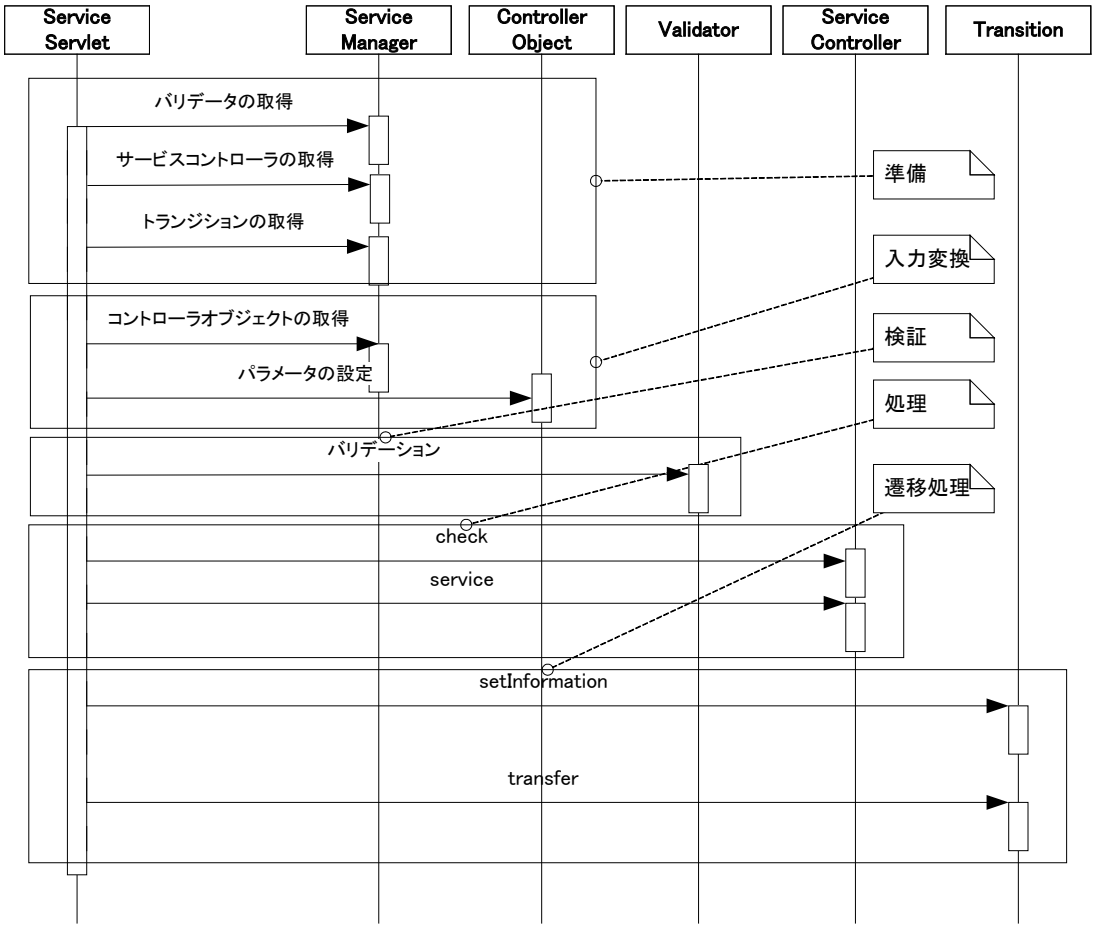


Figure 3-8 Screen Transition Overview

3.5.1 ServiceServlet

If you see [Figure 3-8 Screen Transition Overview], all the requests are managed in 1 servlet. This servlet is implemented as jp.co.intra_mart.framework.base.service.ServiceServlet.

In [Preparation], application ID and service ID will be obtained, corrensponding ControllerObject, Validator、ServiceController and Transition, and allocate the request parameter to ControllerObject.

In [Input Process], request contents are input validated by Validator, and check by check method of ServiceController, and processed specifically by service method.

In [Transition Process], preparation for moving to the next screen is done by setInormation method of Transition, and actual transition is made by transfer method.

In [Figure 3-8 Screen Transition Overview], entire process overview when screen is moved are described, but not all processes are described in detail. Detail contents are described in following sequence.

- Preparation
- Input conversion
- Validation
- Process
- Transition process

3.5.2 Preparation

In the IM-JavaEE Framework, ServiceController receives the process and it will be moved to the next screen by Transition. Here, application service and service ID are obtained from the request, and corresponding ServiceController and Transition are obtained.

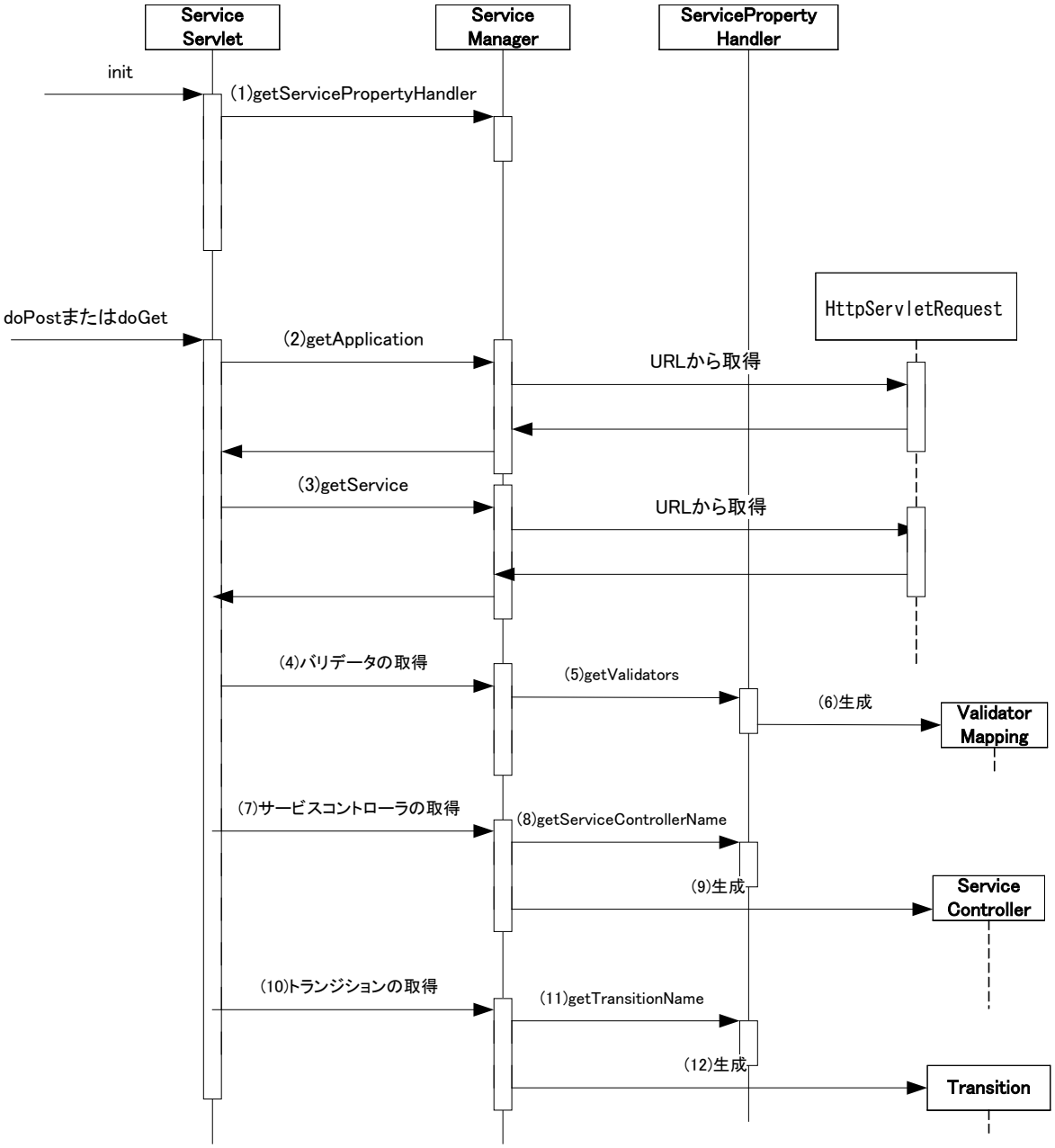


Figure 3-9 Service Framework (preparation)

3.5.2.1 Obtaining Application ID and Service ID

ServiceServlet obtains application ID and service ID from getApplication method and getService method of ServiceManager ((2)(3) of Figure 3-9 Service Framework (preparation)]). HttpServletRequest is given as the argument at this point. ServiceManager obtains ID from URL of the request and returns the value to ServiceServlet. ServiceServlet can obtain ID from HttpServletRequest directly, but you should not do so since ServiceManager is made into capsule.

In current version IM-JavaEE Framework, application ID and service ID are passed as the parameter of request, but this may be changed in later versions. Therefore, the developer should not create program based on the concept that above parameters (application ID and service ID) are exist.

3.5.2.2 Obtaining Validator

ServiceServlet obtains Validator corresponding to the application ID and the service ID from ServiceManager ((4)of [Figure 3-9 Service Framework (preparation)]. getValidatorMapping method of ServiceManager is used at this poing. List of corresponding ValidatorMapping is obtained through getValidators of ServicePropertyHandler in this method ((6) of [Figure 3-9 Service Framework (preparation)]).

3.5.2.3 Obtaining ServiceController

ServiceServlet obtains ServiceContoller that corresponds to application ID and service ID from ServiceManager ((7) of [Figure 3-9 Service Framework (preparation)]. getServiceController method of ServiceManager is used at this point. This method obtains ServiceController class name through getServiceControllerName of ServicePropertyHandler ((12) of [Figure 3-9 Service Framework (preparation)] and generates corresponding ServiceController newly ((9) of [Figure 3-9 Service Framework (preparation)].

If ServiceController class name that corresponds to application ID and service ID is not specified, ServiceManager returns null.

3.5.2.4 Obtaining Transition

ServiceServlet obtains Transition that corresponds to application ID and service ID from ServiceManager ((9) of [Figure 3-9 Service Framework (preparation)]. getTransition method of ServiceManager is used at this point. This method obtains Transition class name through getTransitionName of ServicePropertyHandler ((10) of [Figure 3-9 Service Framework (preparation)], and generates corresponding Transition newly ((15) of [Figure 3-9 Service Framework (preparation)]).

If Transition class name that corresponds to application ID and service ID is not specified, ServiceManager generates jp.co.intra_mart.framework.base.service.DefaultTransition newly and returns.

3.5.3 Input Conversion

If the request is set to the server from Web client (mainly from browser), in component (Servlet, JSP etc) inside the Web container, the contents can be used as the format in javax.servlet.HttpServletRequest. However, normally this does not make the developer understand intuitively. This is because when you obtain information from HttpServletRequest, normally getParameter method or getParameterNames method, following points should be awared in order to use these methods.

- These methods can assume [Obtain Parameter] from the method name, but actual parameters obtained need to be determined by the argument etc.
- Since the return value of these methods are fixed java.lang.String, the developer needs to convert the parameter into appropriate primitive type or object if necessary.
- In service framework, request contents are expressed as JavaBeans called ControllerObject, and conversion from request to ControllerObject can be performed by the component called ControllerConverter. (refer to

[Figure 3-10 Converting from Request to ControllerObject].

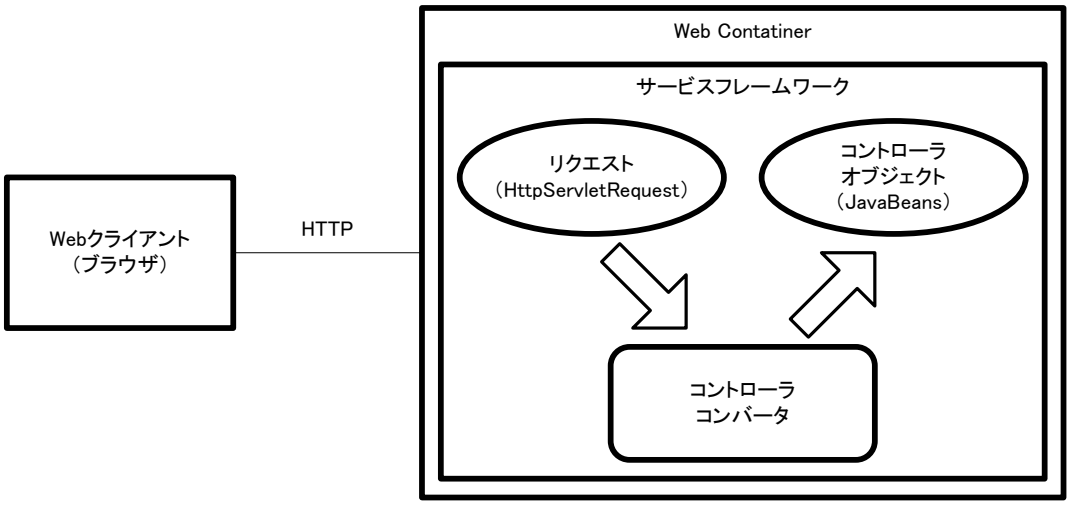


Figure 3-10 Converting from Request to ControllerObject

ControllerConverter can be assingned only one for one request. You don't have to assign ControllerConverter.

3.5.3.1 ControllerObject

ControllerObject is an input information which was converted from the request contents. ControllerObject should meet the following conditions.

- It meets JavaBeans requirments.
- It implements jp.co.intra_mart.framework.base.service.controller.ControllerObject interface.

3.5.3.2 ControllerConverter

ControllerConverter converts javax.servlet.HttpServletRequest contents into ControllerObject. ControllerConverter should meet to the following conditions.

- It implements jp.co.intra_mart.framework.base.service.controller.ControllerConverter interface.
- Constructor (default constructor) without argument exists in public
- Init method and destroy method are implemented.
- It is implemented to return appropriate ControllerObject by convert method.

In service framework, ControllerConverter is managed to show in [Figure 3-11 ControllerConverter Management].

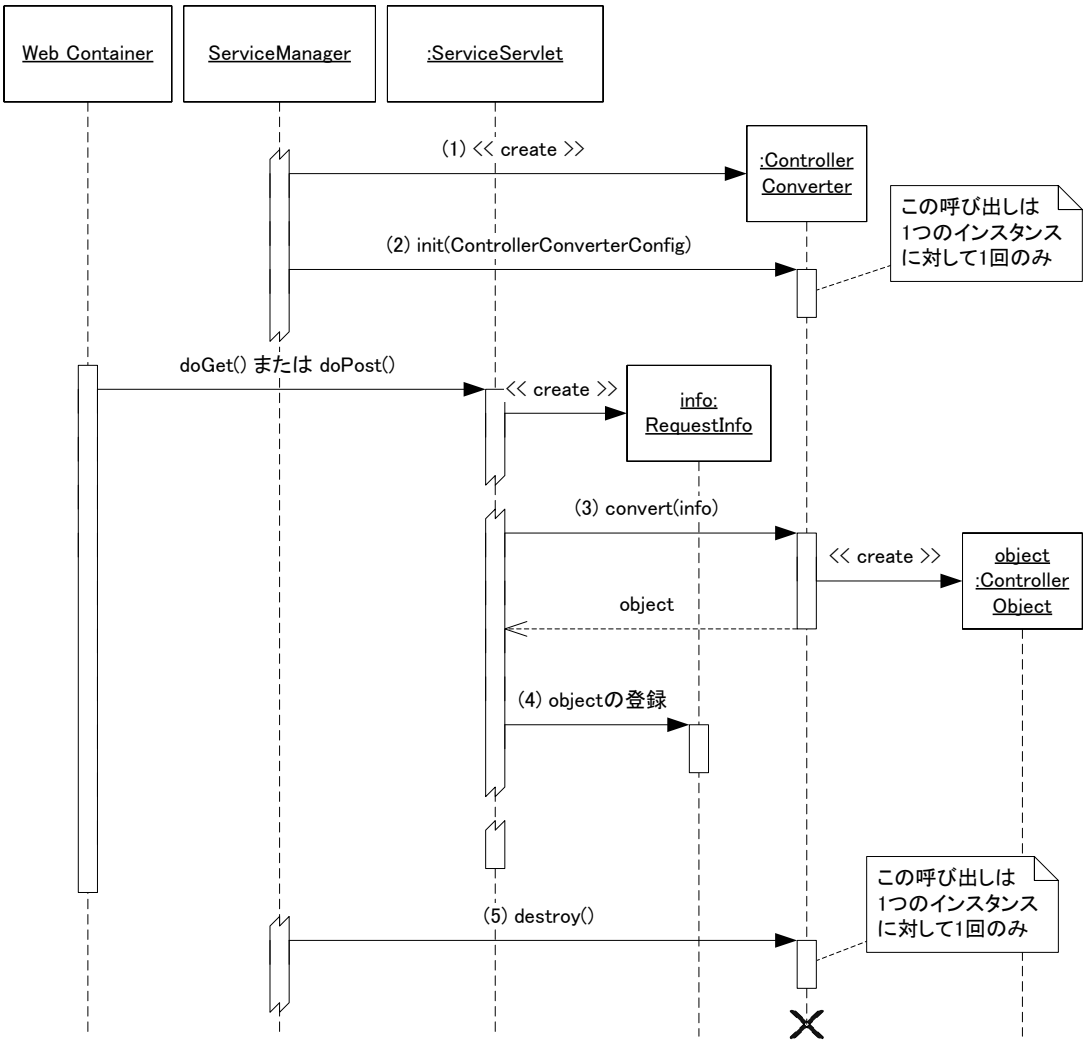


Figure 3-11 ControllerConverter Management

1. It calls init method of ControllerConverter when the request is come up, and initializes ControllerConverter. Init method is called only once for the same instance.
2. It calls convert method of ControllerConverter, and generates ControllerObject.
3. Generated ControllerObject is registered in RequestInfo. ControllerObject that is registered here can be obtained by using getControllerObject method of RequestInfo in later phase.
4. After ControllerObject generation, it calls destroy method of ControllerConverter, and releases ControllerConverter. destroy method is called only once for the same instance.

3.5.3.2.1 **ControllerConverter provided as Standard**

In service framework, basic ControllerConverter is provided as standard.

3.5.3.2.1.1 SimpleControllerConverter

jp.co.intramart.framework.base.service.controller.SimpleControllerConverter is ControllerConverter that sets request parameters to the specified ControllerObject. SimpleControllerConverter will set values for all parameters included in the request (group of parameters that are obtained by getParameterNames method of javax.serv.et.ServletRequest) to ControllerObject according to the following rules.

- If the return value of getParameterValues method is the array of size 1, its value is set to corresponding property of ControllerObject as it is.
- If the return value of getParameterValues method is the array of size 2 or greater, its value is set to corresponding property of ControllerObject as an array. Sequence inside the array will be the same with the array obtained by getParameterValues method.

As an example, the case ControllerObject shown in [List 3-2 MyControllerObject.java] is generated by SimpleControllerConverter from the request sent by HTML shown in [List Request that includes 3-1 ControllerObject Data] will be presented.

List Request that includes 3-1 ControllerObject Data

```
...
<INPUT type="hidden" name="name" value="foo">
<INPUT type=" hidden" name="password" value="bar">
<INPUT type=" hidden" name="hobby" value="baseball">
<INPUT type=" hidden" name="hobby" value="soccer">
<INPUT type=" hidden" name="hobby" value="reading">
...
```

List 3-2 MyControllerObject.java

```
...
public class MyControllerObject implements ControllerObject {
	private String name = null;
	private String password = null;
	private String[] hobbies = new String[3];

	public String getName() {
		return this.name;
	}

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

	public String getPassword() {
		return this.password;
	}

	public void setPassword(String password) {
		this.password = password;
	}

	public String getHobby(int index) {
		return this.hobbies[index];
	}

	public void setHobby(int index, String hobby) {
		this.hobbies[index] = hobby;
	}
}
...
```

In this case, getter method of MyControllerObject converted from the request returns the value that are shown in [Table 3-1 Set Value].

Table 3-1 Set Values

getter method	Values to be obtained
getName()	foo
getPassword()	bar
getHobby(0)	baseball
getHobby(1)	soccer
getHobby(2)	reading

3.5.3.2.2 **Original ControllerConverter**

ControllerConverter shown in [3.5.3.2.1.1 SimpleControllerConverter] just sets parameter (String) of the request to property (String) of ControllerObject. If you want to generate ControllerObject that does not have this limitation (objects other than String or has primitive type such as int as property), you can create ControllerConverter originally.

Conditions that ControllerConverter should meet are shown in [3.5.3.2 ControllerConverter]

3.5.3.3 ControllerConverterConfig

init method of ControllerConverter receives initialization information to the argument from service framework. This information is the instance of class that jp.co.intra_mart.framework.base.service.controller.ControllerConverterConfig is implemented. Following methods are provided to obtain initialization information in ControllerConverterConfig.

- public String getInitParameter(String name)

It obtains initialization information that has the specified parameter name.

- public Enumeration getInitParameterNames()
  It obtains parameter name list.

The developer of ControllerConverter can obtain unique initialization information by using ControllerConverterConfig in init method.

3.5.4 Validation

Normally, the contents are validated before any pocess are performed for the request from Web client side. If browser is used as Web client, certain level of validation is possible if you use JavaScript, the validation must be performed at server side for the absolute protection against system invalid data.

Service framework can validate the request contents by the component called Validator. (refer to [Figure 3-12 Request Validation]

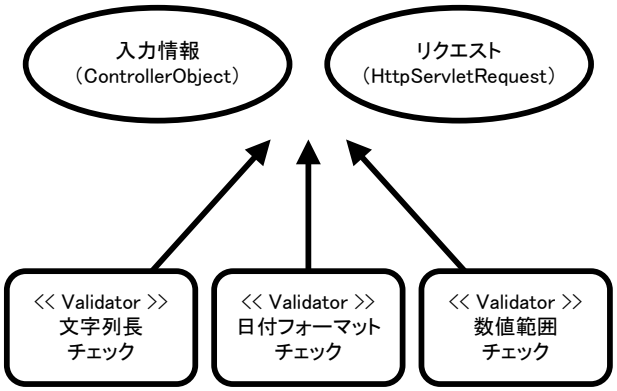


Figure 3-12 Request Validation

Validator can be multiply assigned for one request. You don't have to assingn Validator.

3.5.4.1 Validator

Validator validates the contents of information (mainly ControllerObject) that can be obtained by the request. Validator have to meet the following conditions.

- It implements jp.co.intra_mart.framework.base.service.validator.Validator interface.
- Public constructor without argument (default constructor) exists.
- init method and destroy method are implemented.
- It is implemented to return appropriate ValidationExceptionDetail by validate method. Implementation of Validate method should return null if the validation result is valid, and return ValidationExceptionDetail including detail information if the result is invalid.

In service framework, Validator is managed to show in [Figure 3-13 Validator Management].

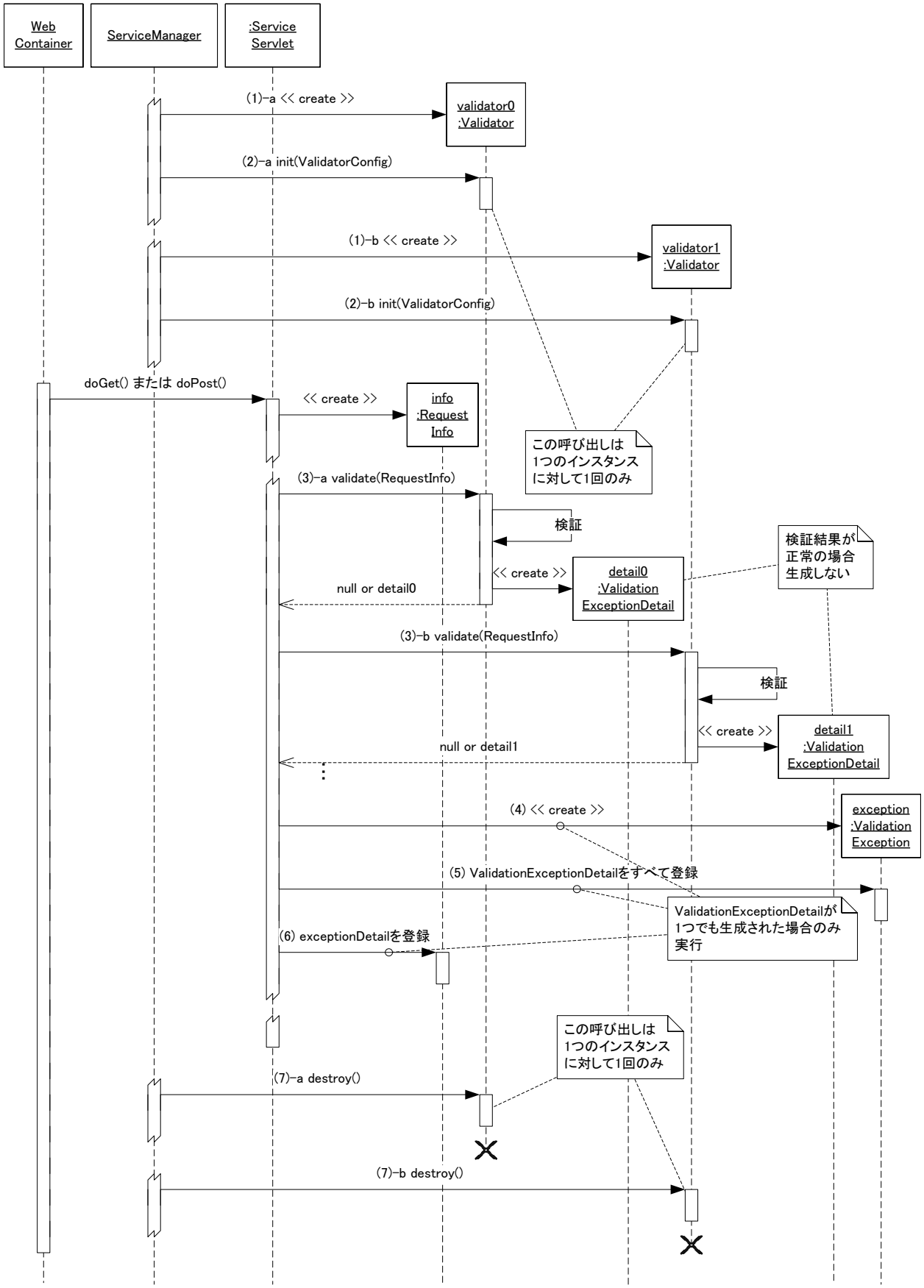


Figure 3-13 Validator Management

1. It calls init method of Validator when request is come up and initializes the Validator. init method is called only once for the same instance.

2. If there is a request for Validator (request validation), all corresponding validate methods of Validator are called. If multiple Validators are corresponded for one request, validate method is called in the sequence of the set Validator. In validate method, if the contents of request are judged as invalid, it generates ValidationExceptionDetail as the detail information when the validation is failed and returns. It returns null if the request contents are judged as valid.
3. If validate method in any of the corresponding validator in (2) returns ValidationExceptionDetail, ValidationException is generated.
4. All ValidationExceptionDetail are registered in generated ValidationException.
5. ValidationException which was generated in (2) and detail information at validation failure time of it being set in (3) will be registered to RequestInfo. ValidationException registered here can be obtained by using getThrowable method of RequestInfo in later phase.
6. It calls destroy method of Validator and releases Validator. destroy method is called only once.

3.5.4.1.1 **Validator provided as Standard**

In service framework, basic Validator is provided as standard.

3.5.4.1.1.1 FormatValidator

jp.co.intra_mart.framework.base.service.validator.FormatValidator validates the format in input character string. In this class, property is obtained by considering ControllerObject as JavaBeans, and validation for the matching with the format in regular expression is made.

Parameters can be set are described in [Table 3-2 FormatValidator Parameter].

Table 3-2 FormatValidator Parameter

Parameter name	Description
property	Property name of controller object to be the validation target
regex	Format in regular expression
message	Message when it is invalid character string

Parameter property is the parameter that needs to be set. Property of ControllerObject that is to be specified by parameter property must be java.lang.String.

Parameter regax is the parameter that needs to be set. Parameter regex should be able to be set to the argument of matches method of java.lang.String.

Parameter message is the parameter that can be set uniquely.

As an example, the case when ControllerObject shown in [List 3-3 FormatSampleControllerObject.java] is evaluated by FormatValidator is considered.

List 3-3 FormatSampleControllerObject.java

```
...
public class FormatSampleControllerObject implements ControllerObject {

    public String getLocalPhone() {
        return "01-2345-6789";
    }

    public String getInternationalPhone() {
        return "(+81) 1 2345 6789";
    }
}
```

When ControllerOnject shown in [List 3-3 FormatSampleControllerObject.java] is evaluated,example of parameter setting and its validation results of FormatValidator will be described in [Table 3-3 Parameters and Evaluation Contents of FormatValidator].

Table 3-3 Parameters and Evaluation Contents of FormatValidator

property	**regex**	**Evaluation Contents**
localPhone	¥d{2}¥-¥d{4}-¥d{4}	Evaluation : Normal Message : (None)
	¥d{10}	Evaluation : Abnormal Message : Value of parameter message
internationalPhone	¥d{2}¥-¥d{4}-¥d{4}	Evaluation : Abnormal Message:Value of parameter message
	¥(¥+¥d{2}¥) ¥d ¥d{4} ¥d{4}	Evaluation : Normal Message : (None)
(No setting)	aaaa	Exception when initialization (parameter property should be set)
localPhone	(No setting)	Exception when initialization (parameter regex should be set)
other	aaaa	Exception when validation (there is no property other in FormatSampleControllerObject)

If parameter message is omitted and the character string does not match to the format, ValidationExceptionalDetail that includs the message prepared in system is returned.

3.5.4.1.1.2 LengthValidator

jp.co.intra_mart.framework.base.service.validator.LengthValidator validates the length of input character string. In this class, property (character string) is obtained by considering ControllerObject as JavaBeans, and it is verified if the length of the character string is within the specified scope.

Parameters can be set are shown in [Table 3-4 LengthValidator Parameter].

Table 3-4 LengthValidator Parameter

Parameter Name	Description
property	Property name of controller object to be the validation target.
min	Miminum length of character string
max	Maximum length of character string
message_short	Message when the character string is shorter than the minumum length.
message_long	Message when the character string is longer than the maximum length.

Parameter property is the parameter that needs to be set. Property of ControllerObject specified by parameter property must be java.lang.String.
Parameter min is the parameter that can be set uniquely. Parameter min should be greater that 0 and less than the maximum value of int (java.lang.Integer.MAX_VALUE). If parameter min is omitted, it is assumed that 0 is set.

Parameter max is the parameter that can be set uniquely. Parameter max should be greater than 9 and less thatn the maximum valueofint (java.lang.Integer.MAX_VALUE). If parameter max is omitted, it is assumed that the maximum value of int is set.

Parameter message_short is the parameter that can be set uniquely.

Parameter message_long is the parameter that can be set uniquely.

As an example, the case when ControllerObject shown in [List 3-4 LengthSampleControllerObject.java] is evaluated by LengthValidator is considered.

List 3-4 LengthSampleControllerObject.java

```
...
public class LengthSampleControllerObject implements ControllerObject {

    public String getPassword() {
        return "mypassword";
    }
}
```

When ControllerObject described in [List 3-3 FormatSampleControllerObject.java] is evaluated, parameter setting and its validation results of FormatValidator are shown in [Table 3-5 Parameters and Evaluation Contents of LengthValidator].

Table 3-5 Parameters and Evaluation Contents of LengthValidator

property	min	max	Evaluation Contents
password	8	20	Evaluation : Normal Message: (None)
	4	8	Evaluation : Abnormal Message : Value of parameter message_long
	12	20	Evaluation : Abnormal Message : Value of parameter message_short
	10	10	Evaluation : Normal Message : (None)
	(No setting)	20	Evaluation : Normal Message : (None)
	8	(No setting)	Evaluation : Normal Message : (None)
	(No setting)	(No setting)	Evaluation : Normal Message : (None)
(No setting)	8	20	Exception when initialization (parameter property needs to be set)
password	20	8	Exception when initialization (parameter min should be less than parameter max)
other	8	20	Exception when validation (there is no property other in LengthSampleControllerObject)

If parameter message_short is omitted and the length of character string is shorter than the minimum value, ValidationExceptionalDetail that includes the message prepared in system will be returned. It will be the same if parameter message_long is omitted and the length of character string is longer than the maximum value.

3.5.4.1.1.3 NumericValidator

jp.co.intra_mart.framework.base.service.validator.NumericValidator validates whether the input value is valid as integer or not. In this class, property is obtained by considering ControllerObject as JavaBeans, it is verified if the value is valid as numeric value.

Parameters that can be set are described in [Table 3-6 NumericValidator Parameter].

Table 3-6 NumericValidator Parameter

Parameter Name	Description
property	Property name of contoller object to be the validation target.
message	Message when it is not valid as integer

Parameter property is the parametter that needs to be set.

Parameter message is the parameter that can be set uniquely.

As an example, the case when ControllerObject shown in [List 3-5 NumericSampleControllerObject.java] is evaluated by NumericValidator is considered.

List 3-5 NumericSampleControllerObject.java

```
・・・
public class NumericSampleControllerObject implements ControllerObject {

    public int getIntNum() {
        return ・・・;
    }

    public long getLongNum() {
        return ・・・;
    }

    public String getPositive () {
        return "123456789";
    }

    public String getNegative() {
        return "-123456789";
    }

    public String getNotNum() {
        return "abcd";
    }
}
```

When ControllerObject shown in [List 3-5 NumericSampleControllerObject.java] is evaluated, examples of parameter setting of NumericValidator and its validation results are shown in [Table 3-7 Parameters and Evaluation Contents of NumericValidator].

Table 3-7 Parameters and Evaluation Contents of NumericValidator

property	Evaluation Contents
intNum	Evaluation : Normal Message : (None)
longNum	Evaluation : Normal Message : (None)
positive	Evaluation : Normal Message : (None)
negative	Evaluation : Normal Message : (None)
notNum	Evaluation : Abnormal Message : Value of parameter message
(No setting)	Exception when initialization ()(parameter property needs to be set)
other	Exception when validation (There is not property other in FormatSampleControllerObject)

If parameter message is omitted and property is not valid as integer, ValidationExceptionDetail that includes message provided in system will be returned.

3.5.4.1.1.4 NumericRangeValidator

jp.co.intra_mart.framework.base.service.validator.NumericRangeValidator validates the size of input numeric value. In this class, property (numeric value) is obtained by considering ControllerObject as JavaBeans, and it is verified if the numeric value is within the specified scope.

Parameters that can be set are described in [Table 3-8 NumericRangeValidator Parameter].

Table 3-8 NumericRangeValidator Parameter

Parameter Name	Description
property	Property name of controller object to be validation target
min	Minumum value of the numeric value
max	Maximum value of the numeric value
message	Message if the numeric value is out of the specified scope

Parameter property is the parameter that needs to be set. Property of ContollerObject specified by parameter property should be valid as integer.

Parameter min is the parameter that can be set uniquely. If parameter min is omitted, it is considered as there is no lower limit.

Parameter max is the parameter that can be set uniquely. If parameter max is omitted, it is considered as there is no upper limit.

Parameter message is the parameter that can be set uniquely.

As an example, the case when ControllerObject shown in [List 3-6 RangeSampleControllerObject.java] is evaluated by RangeValidator is considered.

List 3-6 RangeSampleControllerObject.java

```
...
public class RangeSampleControllerObject implements ControllerObject {

    public int getIntNum() {
        return 500;
    }

    public long getLongNum() {
        return -1234L;
    }

    public String getStringNum() {
        return "123456";
    }

    public String getIllegalNum() {
        return "abcd";
    }
}
```

When ContollerObject shown in [List 3-6 RangeSampleControllerObject.java] is evaluated, examples of parameter setting of RangeValidator and its validation result are described in [Table 3-9 Parameter and Evaluation Contents of NumericRangeValidator].

Table 3-9 Parameter and Evaluation Contents of NumericRangeValidator

property	min	max	Evaluation Contents
intNum	100	1000	Evaluation : Normal Message : (None)
	600	(No setting)	Evaluation : Abnormal Message : Value of parameter message
intLong	(No setting)	-1000	Evaluation : Normal Message : (None)
	(No setting)	(No setting)	Evaluation : Normal Message : (None)
stringNum	-100000	200000	Evaluation : Normal Message : (None)
illegalNum	10	100	Evaluation : Abnormal Message : Value of parameter message
stringNum	200000	-100000	Exception when initialization (parameter min should be less than parameter max)
(No setting)	10	100	Exception when initialization (parameter property needs to be set)
other	10	100	Exception when validation (there is no property other in LengthSampleControllerObject)

If parameter message is omitted and the numeric value is out of the specified scope, ValidationExceptionDetail that includes message provided in system will be returned.

3.5.4.1.2 Original Validator

Validator described from [3.5.4.1.1.1 FormatValidator] to [3.5.4.1.1.4 NumericRangeValidator] only validates property of ControllerObject. If you want to generate Validator (concurrent validation of multiple properties or comparison with HttpSession contents) outside this limitation, you can create your own Validator.

Conditions that Validator has to meet are described in [3.5.4.1 Validator].

3.5.4.2 ValidatorConfig

int method of Validator receives initialization information to the argument from service framework. This information is the class instance that jp.co.intra_mart.framework.base.service.validator.ValidatorConfig interface is implemented. In ValidatorConfig, the following methods are provided in order to obtain initialization information.

- public String getInitParameter(String name)
  It obtains initialization information that has the specified parameter.
- public Enumeration getInitParameterNames()
  It obtains a list of parameter name.

The developer of Validator can obtain unique initialization information in init method by using ValidatorConfig.

3.5.5 Process

In IM-JavaEE Framework, ControllerObject, Validator and ServiceController process the requests. Followings are the role of ServiceController :

- Check of request contents
- Process for the requests

Class structure of ServiceController is described as [Figure 3-14 ServiceController Structure]. In this figure, those parts that have [~ServiceController] or [~ServiceResult] should be the class implemented by the developer.

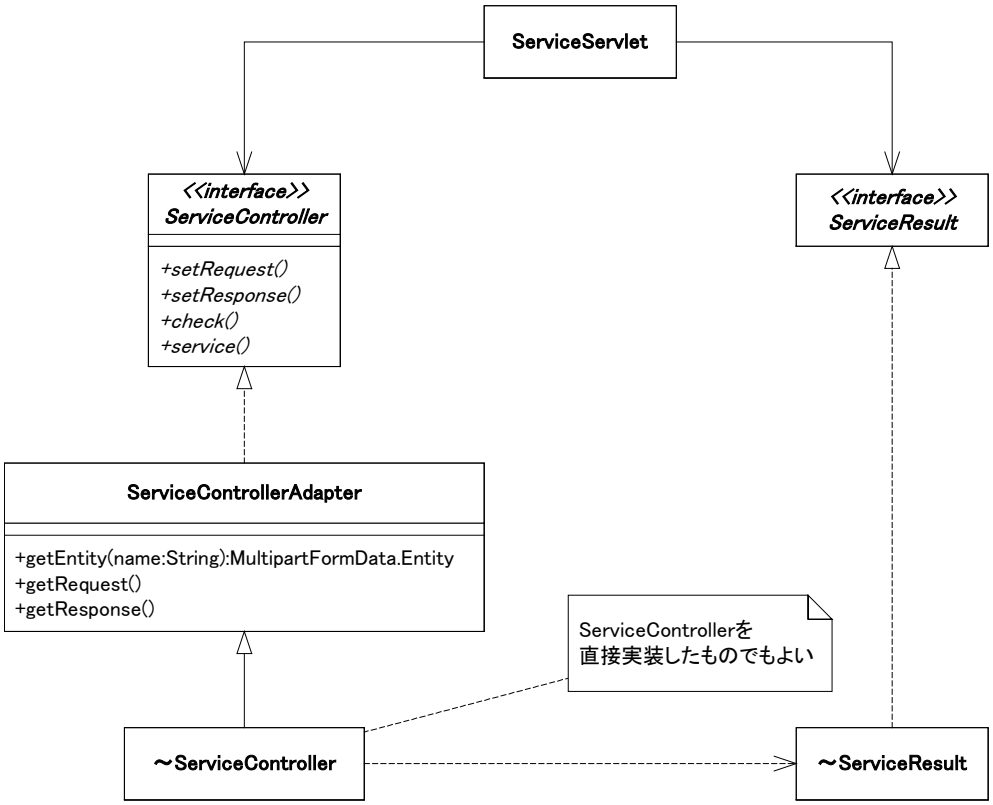


Figure 3-14 ServiceController Structure

When the request is received by ServiceServlet, it checks if ServiceContoller has been set. If ServiceController is set, it calls the method of the ServiceController as [Figure 3-15 Service Framework (Input Process)].

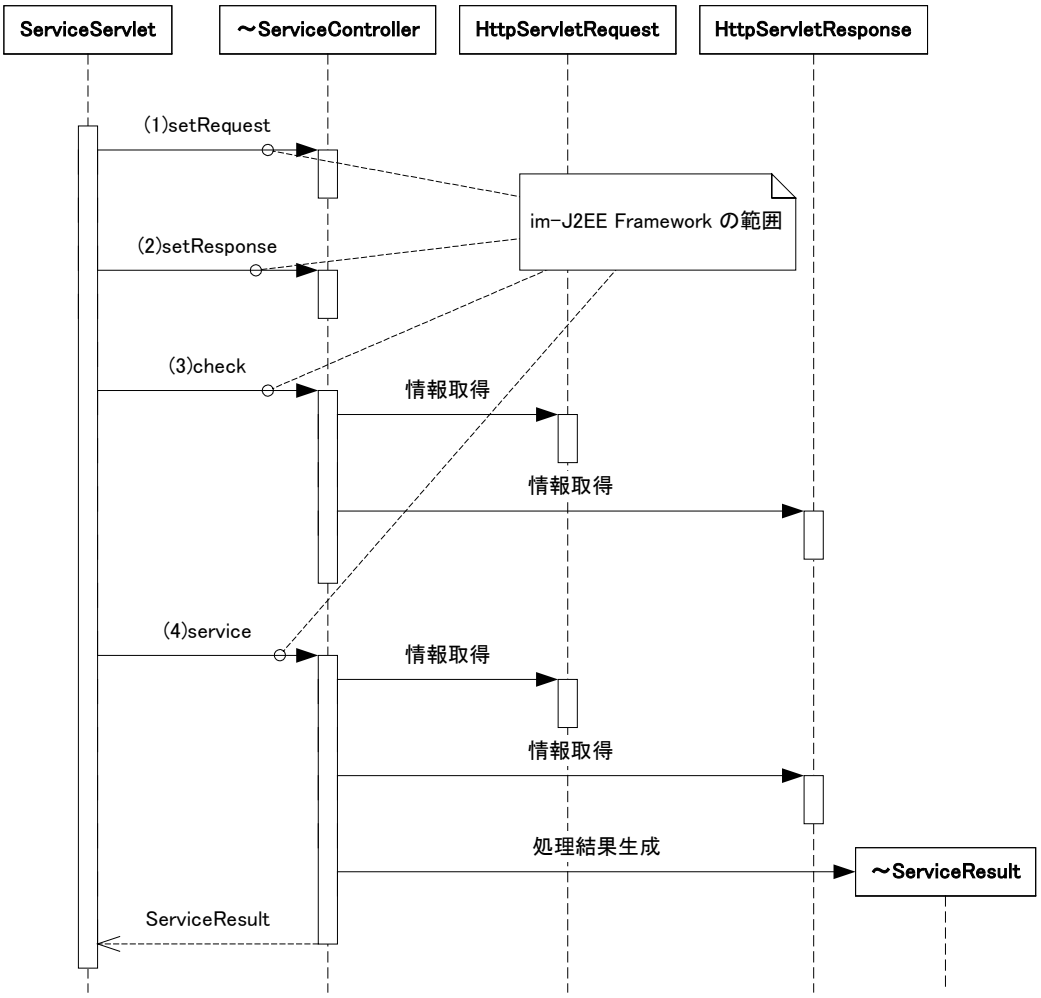


Figure 3-15 Service Framework (Input Process)

1. It calls setRequest(javax.servlet.HttpServletRequest request) method, passes the request to ServiceController.
2. It calls setResponse(javax.servlet.HttpServletResponse response) method and passes the response to ServiceController.
3. It calles check() method and do input check. If you can obtain HttpServletRequest or HttpServletResponse that is previously set in the ServiceController, you can use the contents.
4. It calls service() method and obtains ServiceResult, which is the process resutl. If you can obtain HttpServletRequest or HttpServletResponse that is previously set in the ServiceController, you can use the contents.

3.5.5.1 ServiceControllerAdapter

Since the jp.co.intra_mart.framework.base.service.ServiceController is an interface, all methods should be implemented if the developper creates ServiceController by using this interface. This indicates, if request or response is used by check method or service method, the request or the response has to be set as instance variable by setRequest method or setResponse method. Therefore, even if request or response is not necessary or if input check is not performed, each type of method (setRequest, setResponse, and check) should be implemented as null.

In order to remove the burden of these, the developer can recommend how to inherit jp.co.intra_mart.framework.base.service.ServiceControllerAdapter class, not ServiceController interface, when the developer creates Service Controller. If the ServiceController is created by inheriting this class, following points are given as advantage.

- Only required methods should be implemented.

- setRequest or setResponse is already implemented, and methods that
- obtain set request or response (getRequest、getResponse) are provided.
- If the request if file upload, method (getEntity) that obtains the contents is provided.
- Each utility method (createEvent or dispatchEvent etc.) related to the event framework introduced in [4 Event Framework] are prepared.

3.5.5.2 Input Check

In IM-JavaEE Framework, check method is prepared for input check. This method implementation should be transferred to the developer. The developer should check the request contents in this method.

If the input contents are invalid, this method should be implemented to throw either jp.co.intra_mart.framework.base.service.RequestException or its sub class. Please refer to [3.7.1 Exception Process at Input Time] for details.

3.5.5.3 Process

If you successfully validate request by check method, IM-JavaEE Framework will call the service method continuously.The developer should implement as requesting the process based on the request contents in this method. In the implementation, the developer requests that the process should be made based on the contents of request in this method. It is described as [Requesting Process] here, because detail business logic should not be described in this method, but it had better be described outside (event framework of IM-JavaEE Framework etc.) for better maintenance and general usability. It is recommended that actual processing itself should not be done within the service method and it should play only the window role.

Followings are how to use external business logic

- Use event framework from ServiceControllerAdapter method
- Use event framework of IM-JavaEE Framework directly

3.5.5.3.1 Use Event Framework from ServiceControllerAdapter Method

The method to request the process to event framework using ServiceControllerAdapter method is the most simple implementation method. The developer can use event framework as described in [Figure 3-16 Use Event Framework from ServiceControllerAdapter]

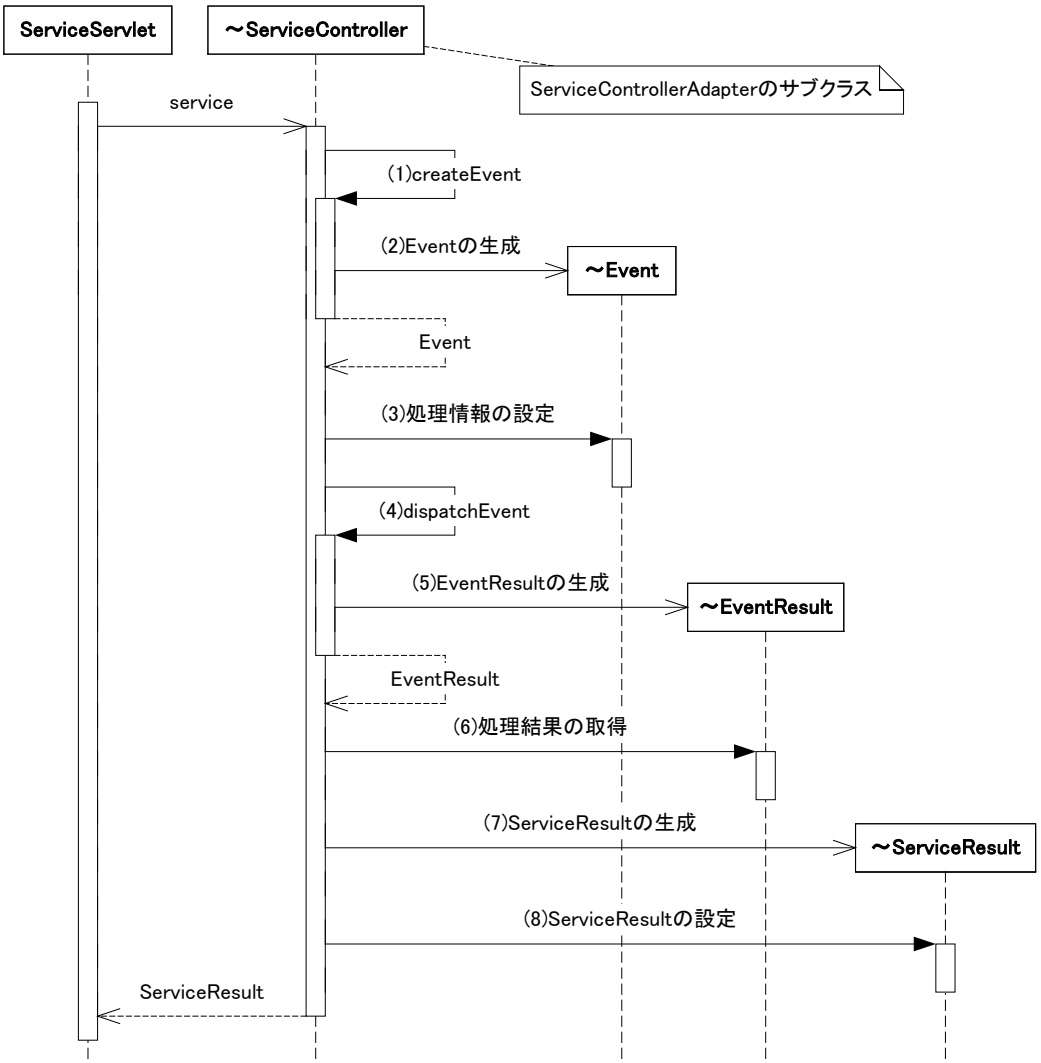


Figure 3-16 Use Event Framework from ServiceControllerAdapter

1. It calls createEvent method and obtains Event.
2. ServiceControllerAdapter obtains corresponding Event by using event framework (refer to EventManager.createEvent method). At this point, log-in user ID and log-in group ID are obtained from the session and will be used when generating Event.
3. It sets information to do the process on the obtained Event.
4. It calls dispatchEvent method and does the process (refer to EventManager.dispatch method).
5. ServiceControllerAdapter executes the process by using event framework and returns the process result, EventResult.
6. It obtains process result information from EventResult.
7. It generates ServiceResult which would be the return value of ServiceController.
8. It sets the result in ServiceResult.

Among the items listed above, [(2)Event Generation] and [(5)EventResult Generation] are already implemented in ServiceControllerAdapter, so the develper does not have to implement this part.

3.5.5.3.2 Use Event Framework of IM-JavaEE Framework directly

If ServiceController interface is directly implemented, event framework if IM-JavaEE Framework should be directly used when you execute event. Here, how the event framework is used inside the ServiceControllerAdapter is illustrated as sample in [Figure 3-17 Use Event Framework of IM-JavaEE Framework directly]. In actual ServiceControllerAdapter, obtaining log-in user ID and log-in group ID are complicated.

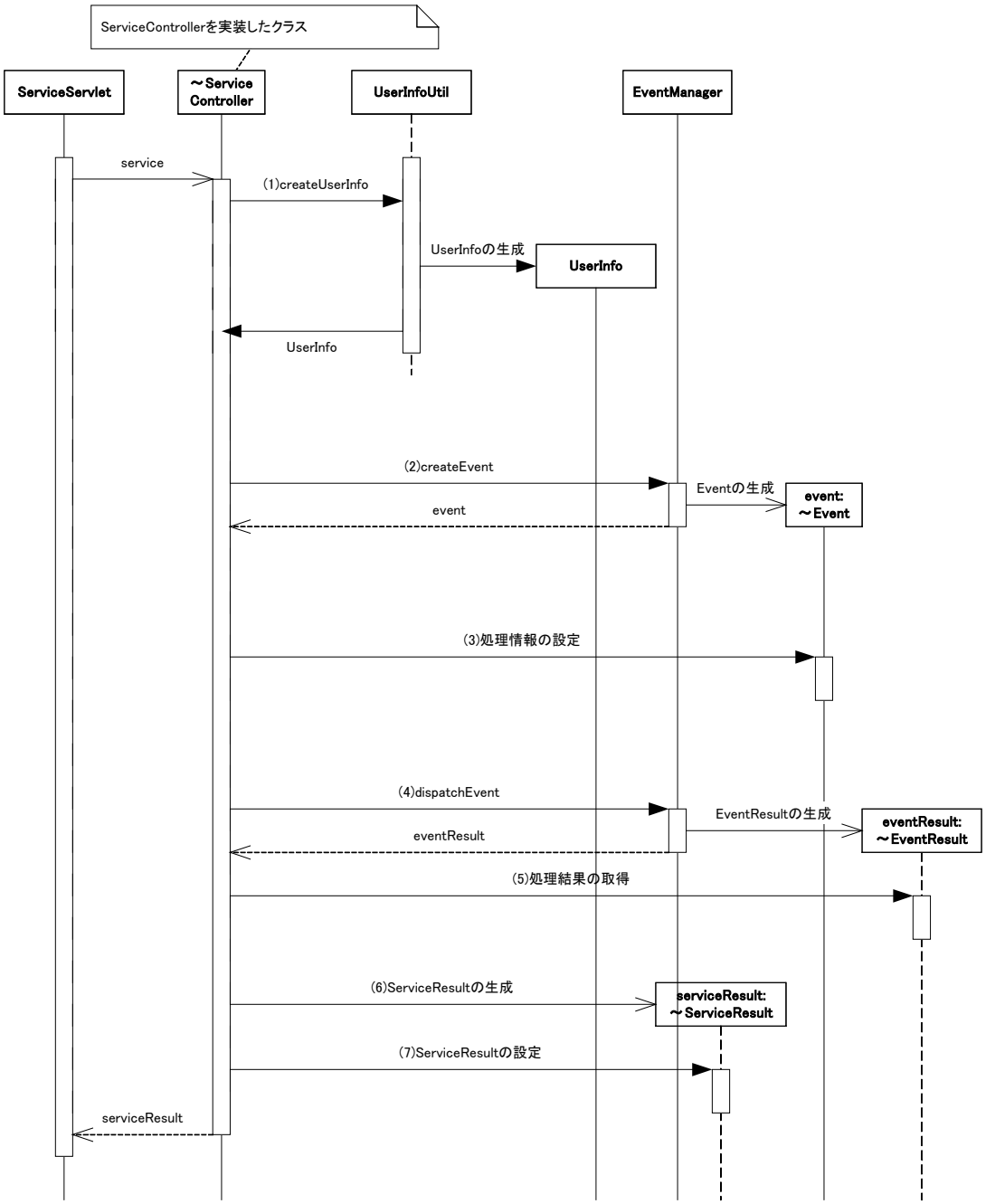


Figure 3-17 Use Event Framework of IM-JavaEE Framework directly

1. It generates UserInfo instance from UserInfoUtil method.
2. It calls EventManager.createEvent method and obtains Event.
3. It sets information to do the process on the obtained Event.
4. It calls EventManager.dispatch method and does the process.
5. It obtains process result information from EventResult.
6. It generates ServiceResult which would be the return value of ServiceController.
7. It sets the result in ServiceResult.

3.5.6 Transition Process

Transition does transition process for the request in IM-JavaEE Framework. Followings are the role of Transition :

- Preparation for the transition destination
- Process for the request

Class structure of ServiceController is described as [Figure 3-18 Transition Structure]. In this figure, those parts that have [~Transition] or [~ServiceResult] should be the class implemented by the developer.

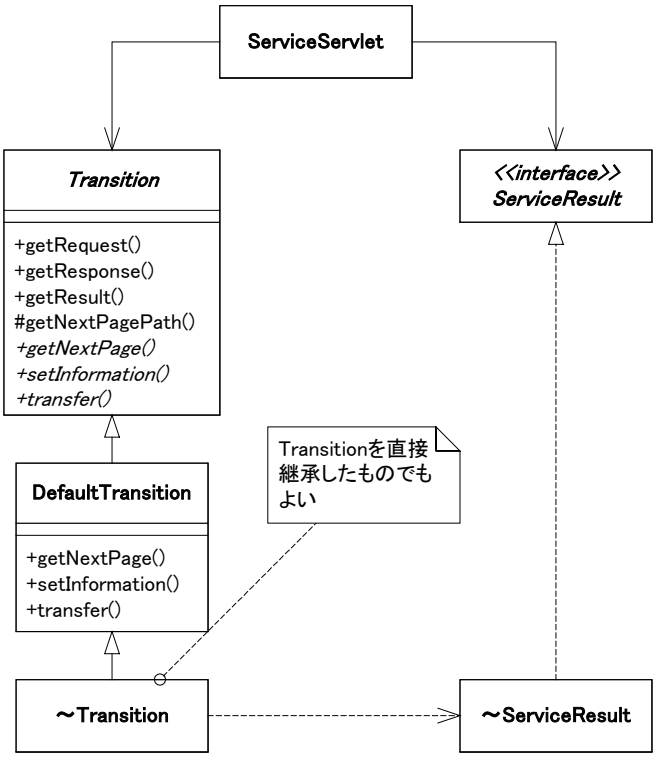


Figure 3-18 Transition Structure

When the request is received by ServiceServlet, it checks if corresponding Transition has been set. If Transition is set, it calls the method of the Transition as [Figure 3-19 Service Framework (Screen Transition))].

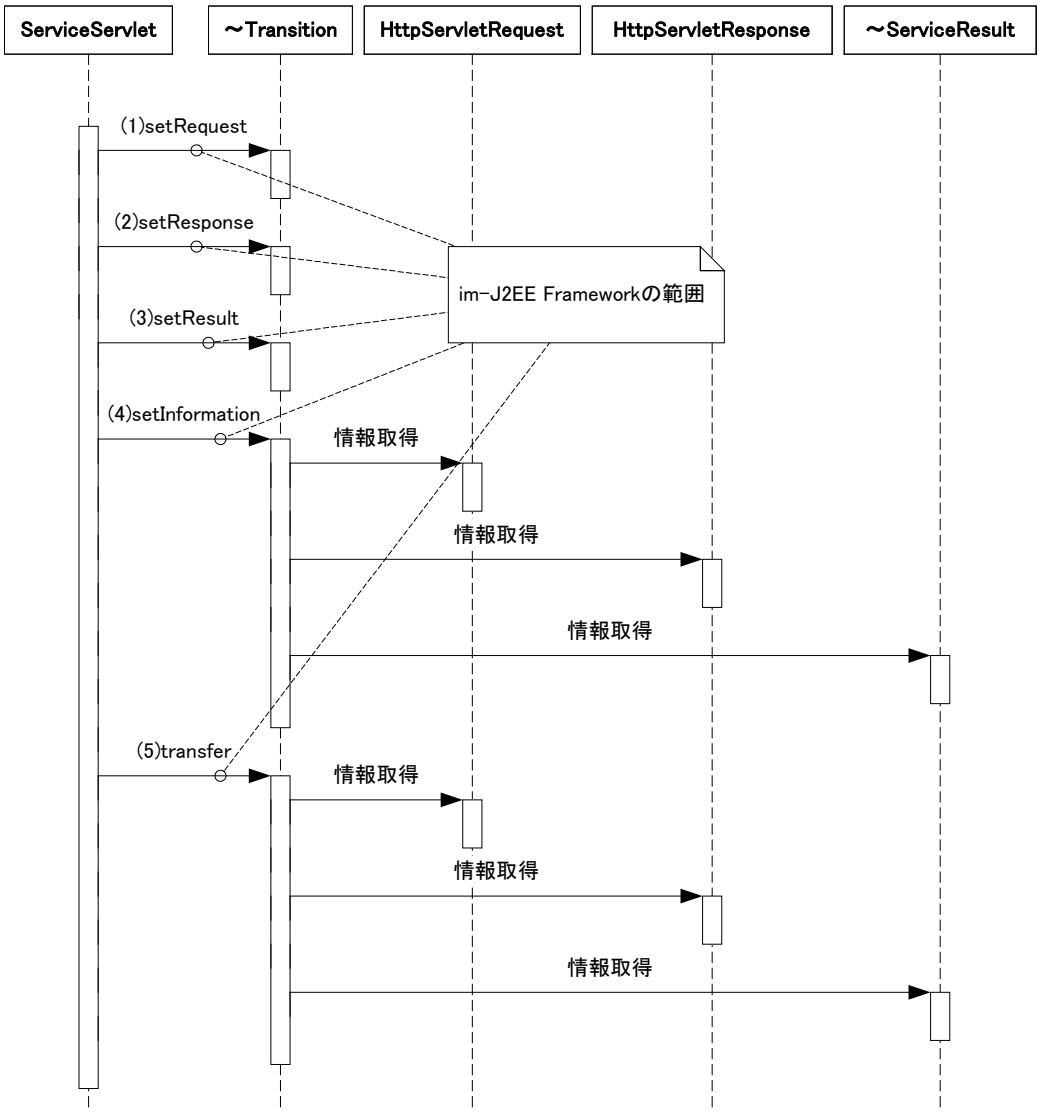


Figure 3-19 Service Framework (Screen Transition)

1. It calls setRequest(javax.servlet.http.HttpServletRequest request) method and passes the request to Transition.
2. It calls setResponse(javax.servlet.http.HttpServletResponse response) method and passes the response to Transition.
3. It calls setResult(jp.co.intra_mart.framework.base.service.ServiceResult result) method and passes the service process result to Transition. If ServiceController is not specified, null will be passed.
4. It calls setInformation() method and do preparation for moving to the next screen. At this point, if HttpservletRequest, HttpServletResponse or ServiceResult that is previously set in Transition can be obtained, the contents can be used.
5. It calls transfer() method and move to the next screen. At this point, if HttpservletRequest, HttpServletResponse or ServiceResult that is previously set in Transition can be obtained, the contents can be used.

Since jp.co.intra_mart.framework.base.service.Transition is abstract class, all the abstract methods should be implemented if the developer creates Transition by using this class.

In IM-JavaEE Framework, following Transition are provided for the purpose in advance :

- jp.co.intra_mart.framework.base.service.DefaultTransition
- jp.co.intra_mart.framework.base.service.IntramartPageBaseTransition

If there is no special circumstances about the screen transition, it is recommended to inherit these classes.

3.5.6.1 Transition

Transition class is an abstract class which is the base for every Transition. In this class, several methods that may be used frequently by thedeveloper are implemented.

3.5.6.1.1 getNextPagePath()

getNextPagePath() method of Transition obtains transition path (refer to [3.4.4.2.4 Transition Path]) by using getNextPagePath(String application, String service) of ServicePropertyHandler from application ID and service ID specified by the request. This operation is shown in [Figure 3-20 getNextPathPage (with no key) of Transition].

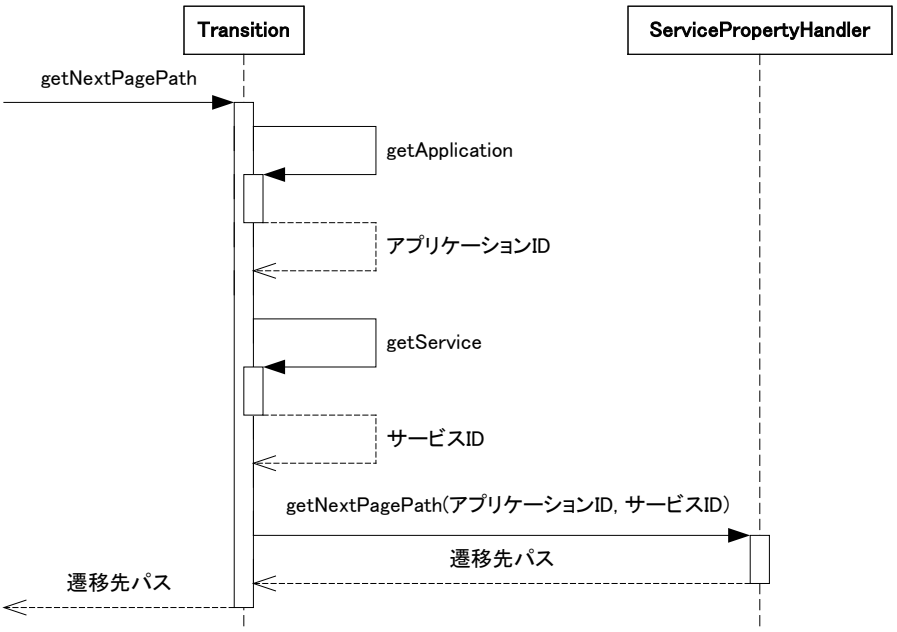


Figure 3-20 getNextPathPage (with no key) of Transition

There are 2 methods called getApplication() and getService() in the sequence diagram shown in [Figure 3-20 getNextPathPage (with no key) of Transition]. These methods each will obtain application ID and service ID for the request. These values are set from ServiceServlet to Transition.

If you override these methods, application ID or service ID can be made irrelavant to the request, but this is not recommended since the maintenance of screen transition may be more difficult.

3.5.6.1.2 getNextPagePath(String key)

getNextPagePath(String key) method of Transition obtains transition path by using getNextPagePath(String application, String service, String key) method of ServicePropertyHandler from application ID, service ID, and key that are specified by the request (refer to [3.4.4.2.3 Transition Path with Key]). This operation is shown in [Figure 3-21 getNextPathPage (with key) of Transition].

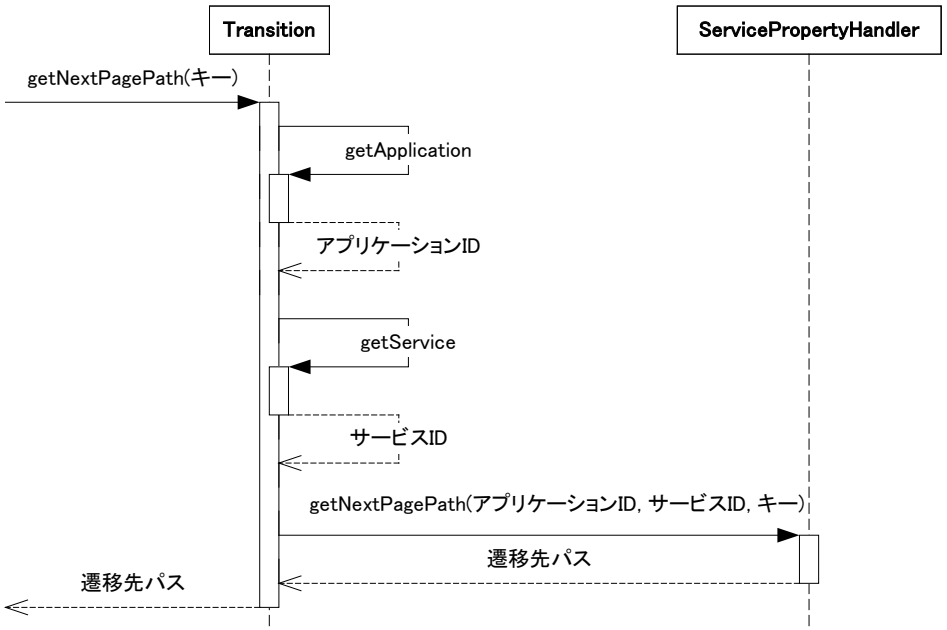


Figure 3-21 getNextPathPage (with key) of Transition

In the sequence diagram shown in [Figure 3-21 getNextPathPage (with key) of Transition], contents of getApplication() and getService() are the same which was described in [3.5.6.1.1 getNextPagePath()].

3.5.6.2 DefaultTransition

If this development assumes IM-JavaEE Framework and simply forwards it to the next screen without any processing, the developer does not have to create/set Transition. If Transition is not set for the request, it is assumed that DefaultTransition is set in IM-JavaEE Framework (If the Transition that corresponds to the request is not set, return value of getTransition method of ServiceManager will be DefaultTransition). setInformation method or transfer method of DefaultTransition does not do any particular process and it simply obtains the property of the next screen and forward it. The operation of DefaultTransition is shown in [Figure 3-22 DefaultTransition Operation].

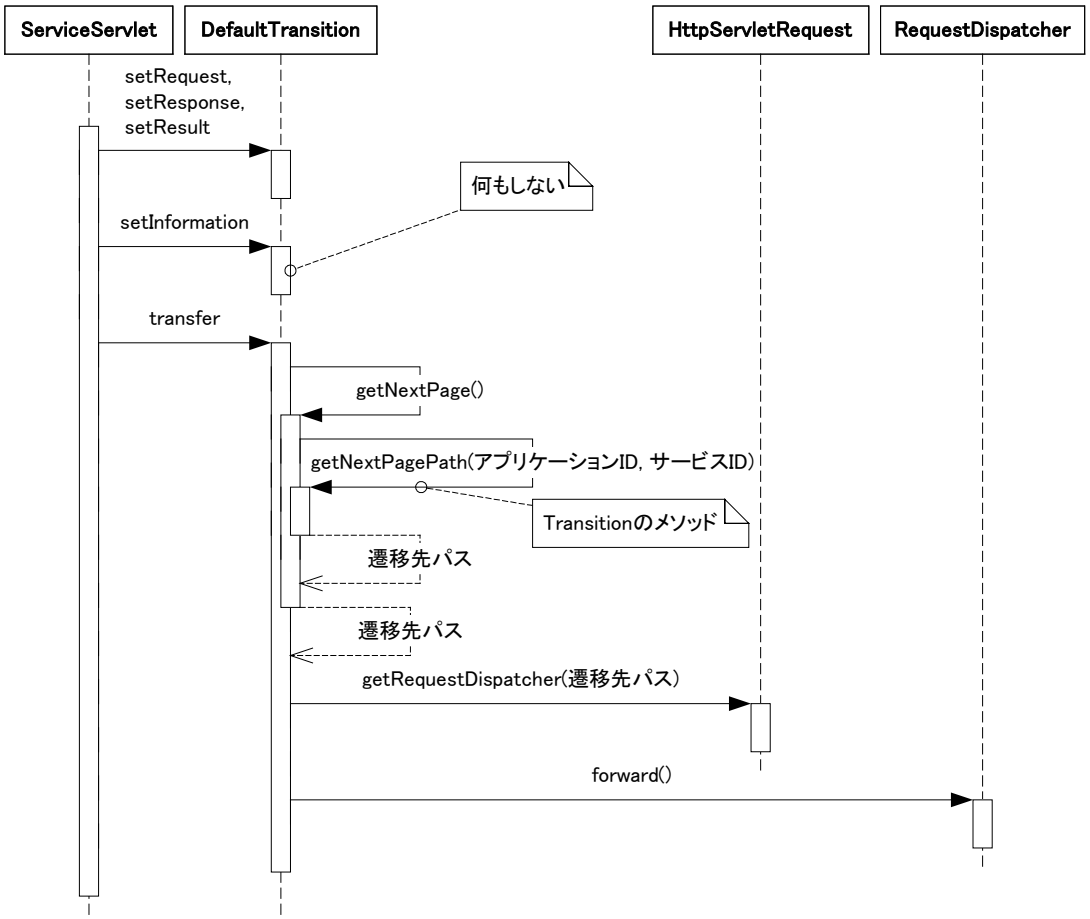


Figure 3-22 DefaultTransition Operation

Following points are given as advantages if Transition is created by inheriting DefaultTransition.

- Only required methods should be implemented.
- If simple transfer method is already implemented, and it is the forward for JSP or Servlet, it is necessary to only set the transition destination to property.
- If the request is file upload, method (getEntity) that obtains the contents is prepared.

In transfer method of [Figure 3-22 DefaultTransition Operation], it is eventually forwarded to the path to be obtained by getNextPagePath(String application, String service) method of ServicePropertyHandler (refer to [3.4.4.2.4 Transition Path]). Therefore, if complicated transition is not required, the developer only needs to set the property to be specified in [3.4.4.2.4 Transition Path].

3.5.6.3 IntramartPageBaseTransition

IntramartPageBaseTransition is Transition that does the forward from JSP page of IM-JavaEE Framework to the screen created in intra-mart script development model.

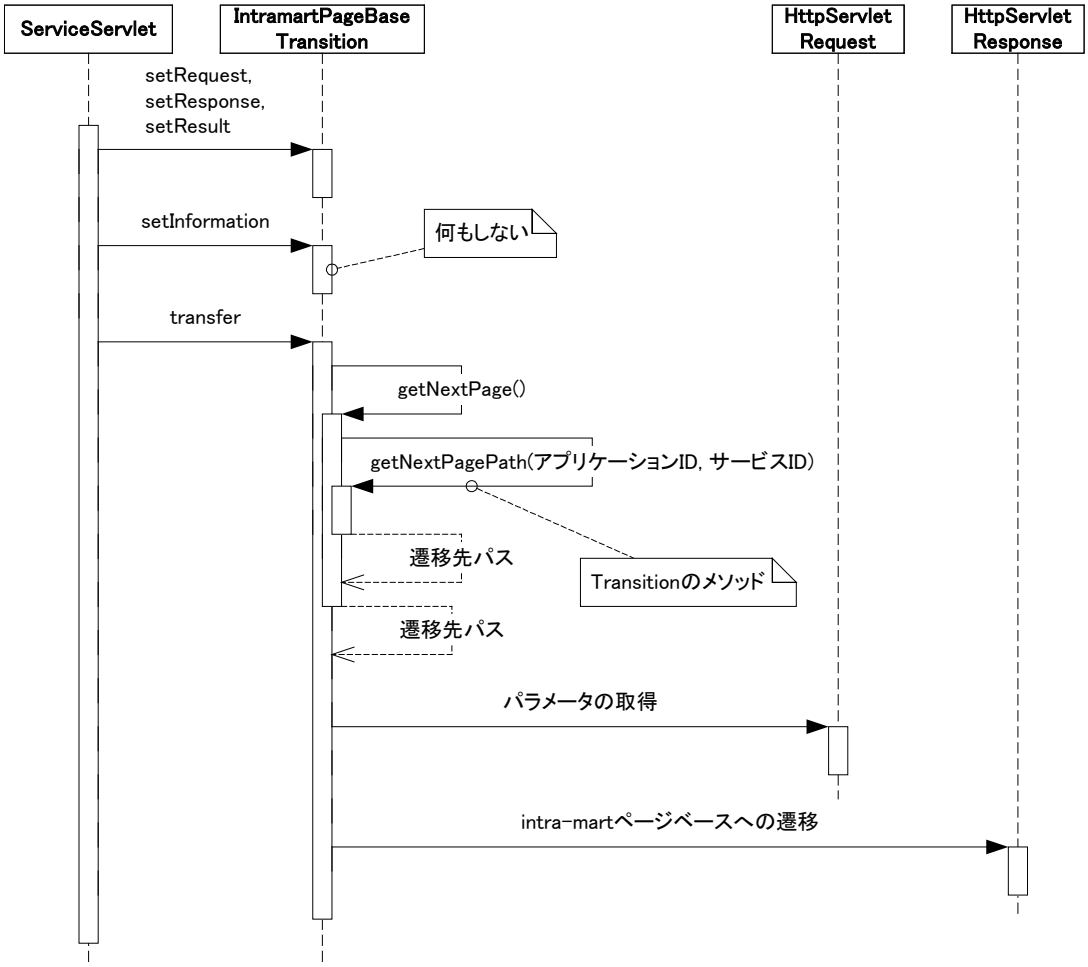


Figure 3-23 IntramartPageBaseTransition Operation

3.5.6.3.1 Simple Transition

The simplest method to move to the next screen by using service framework of IM-JavaEE Framework is to set the property. The property is set so that the value obtained by getNextPagePath(String application, String service) method of ServicePropertyHandler is made the page path of transition destination. Here, application and service each indicate application ID and service ID that are specified in the request.

In this case, no need to set Transition.

3.5.6.3.2 Transition that passes Information to the Next Screen

If the screen at transition destination needs to receive some process results or needs keyword for screen display, the information should be set in setInformation method of Transition. The example of the simplest implementation is shown in [List 3-7 Transition that passes Information to the Next Screen].

List 3-7 Transition that passes Information to the Next Screen

```
...
public class MyTransition extends DefaultTransition {
    public void setInformation() throws TransitionException {
        HttpServletRequest request = getRequest();
        request.setAttribute("search", "keyword");
    }
}
...
```

In [List 3-7 Transition that passes Information to the Next Screen], value "keyword" is set to the request attribute "search". Since it inherits DefaultTransition and methods other than setInformation are not overridden, only simple transition as shown in [3.5.6.3.1 Simple Transition] is made.

3.5.6.3.3 Multiple Transition Destinations

If only one transition destination is fixed, it is easy to move to the screen of IM-JavaEE Framework if you use DefaultTransition. However, if you want to change transition destinations according to the conditions, this method as it is will not work. If there are several destination transition candidates and if you want to assign them dynamically, original Transition should be created.

In this case, following specific methods can be listed:

- It creates class that inherited DefaultTransition and overrides getNextPage method.
- It creates class that inherited DefaultTransition and overrides getNextPagePath method.
- It creates class that inherited Transition (or DefaultTransition) and overrides transfer method.

If transition destination is IM-JavaEE Framework, the first method is the simplest way to implement in many cases. Example of this case is described in [List 3-8 Multiple Transition Destinations].

This Transition is supposed to receive the original ServiceResult called MyResult. Numeric value is set as the process result in MyResult, and it should be obtained by getNumber method. In [List 3-8 Multiple Transition Destinations], the key is redefined based on this numeric value (setInformation method) and it is used to obtain the transition destination (getNextPage method).

List 3-8 Multiple Transition Destinations

```
...
import jp.co.intra_mart.framework.base.service.*;

public class ConditionalTransition extends DefaultTransition {

    private String condition = null;

    public void setInformation() {
        MyResult result = (MyResult)getResult();
        if (result.getNumber() == 1) {
            this.condition = "cond1";
        } else if (result.getNumber() == 2) {
            this.condition = "cond2";
        } else if (result.getNumber() == 3) {
            this.condition = "cond3";
        }
    }

    public void getNextPage() throws ServicePropertyException {
```

```
            return getNextPagePath(this.condition)
        }
}
```

3.5.6.3.4 **Other Transition Destinations**

If the transition destination is other than USP or servlet, it cannot be supported by simple screen transition. Followings are the examples.

- PDF file display
- File download
- Redirect to other contents

In this case, the simplest method is to move to JSP or servlet temporarily and output or redirect to output stream in javax.servlet.http.HttpServletResponse.

Examples of Transition and servlet when downloading the file are each shown in [List 3-9 Transition for File Download] and [List 3-10 Servlet for File Download].

This Transition is supposed to receive the original ServiceResult called DownloadResult. Name and the actual content of the file to be sent to the browser side are set to DownloadResult as process result when downloading, and they should be obtained by getFilename method and getFiledata method. In [List 3-9 Transition for File Downloading], these contents are obtained (setInformation method) and the contents are sent to the browser in [List 3-10 Servlet for File Download] (doGet method)[3].

List 3-9 Transition for File Download

```
...
import jp.co.intra_mart.framework.base.service.*;

public class DownloadTransition extends DefaultTransition {

    public void setInformation() {
        DownloadResult result = (DownloadResult)getResult();
        getRequest().setAttribute("filename", result.getFilename());
        getRequest().setAttribute("filedata", result.getFiledata());
    }
}
```

[3] doGetmethod of HttpServlet is used here. Some modification like changing it to doPost will be necessary according to the actual request.

List 3-10 Servlet for File Download

```
・・・
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class DownloadServlet extends HttpServlet {
・・・

    protected void doGet(HttpServletRequest request,
                         HttpServletResponse response)
                  throws IOException, ServletException {

        // Obtaining information
        String filename = (String)request.getAttribute("filename");
        byte[] filedata = (byte[])request.getAttribute("filedata");

        // output header
        response.setContentType("application/octet-stream");
        response.setHeader("Content-Disposition",
                           "attachment; filename=¥"" + filename + "¥"");

        // output to output stream
        OutputStream os = response.getOutputStream();
        os.write(filedata);
        os.flush();
        os.close();
    }
}
```

You can override transfer method of Transition, and achieve the similar function, although it is not recommended. As you can see in [Figure 3-19 Service Framework (Screen Transition)], the way to move to the next screen (display) is handled by transfer method[4]. Therefore, you can download the file if you do the contents shown in [List 3-10 Servlet for File Download] in transfer method. However, overriding of transfer method is the special method and it is not for general use. Thus this method is not recommended.

3.6 Screen Display

There are several ways for screen display. There are several methods to do the screen display, the most standard method in IM-JavaEE Framework is described here.

3.6.1 JSP

If the development is made in IM-JavaEE Framework, JSP would be the most popular transition destination. Ordinary JSP would suffice for simple display but you can use the followings if you want t obtain contents necessary for initial display or continue session to the next screen.

- Tag library
- HelperBean

[4] This part was implemented in DafaultTransition or IntramartPageBaseTransition, so there is no need to redifine it if these classes are inherited.

3.6.1.1 Tag Library

In IM-JavaEE Framework, following JSP entension tags are provided to keep sessions or support screen display.

- Form
- Link
- Frame
- Param
- Submit
- SubmitLink
- HelperBean
- Message

These JSP extended tags are provided by URI shown in [List 3-11 Tag Library URI of IM-JavaEE FrameworkI], which are privided as standard in IM-JavaEE Framework.

List 3-11 Tag Library URI of IM-JavaEE FrameworkI

```
http://www.intra-mart.co.jp/taglib/core/framework
```

In JSP, if you specify URL shown in [List 3-11 Tag Library URI of IM-JavaEE FrameworkI], you can use tag library of im-JavaEE. Description example of JSP when you use tag library of IM-JavaEE Framework is described in [List 3-12　JSP that uses Tag Library of IM-JavaEE Framework]. In this example, "imartj2ee" is specified as prefix of tag library of IM-JavaEE Framework, and Form tag is used.

List 3-12　JSP that uses Tag Library of IM-JavaEE Framework

```
・・・
<%@ taglib prefix="imartj2ee" uri="http://www.intra-mart.co.jp/taglib/core/framework" %>
・・・
<imartj2ee:Form application="sample" service="test" method="POST">
・・・
</imartj2ee:Form>
```

3.6.1.1.1 Form

Form tag format is almost the same with normal HTML <FORM> tag, but following points are different.

- There is no action attribute.
- Attribute name should consists of all small letters.

On the Form tag, application attribute and service attribute exist instead of action attribute. These attributes indicate application ID and service ID respectively..

3.6.1.1.2 Link

Link tag format is almost the same with normal HTML <A> tag, but following points are different.

- There is no href attibute.
- Attribute name should consists of all small letters.

On the Link tag, application attribute and service attribute exist instead of href attribute. These attributes indicate application ID and service ID respectively.

On the Link tag, you can use Param tag in nested. In this case, Param tag contents will be the parameter of request that is sent by Link tag. In this case, request will be sent by GET.

3.6.1.1.3 **Frame**

Frame tag format is almost the same with normal HTML <FRAME> tag, but following points are different.

- There is no src attribute
- Attribute name should consists of all small letters.

On the Frame tag, application attribute and service attribute exist instead of src attribute. These attributes indicate application ID and service ID respectively.

On the Frame tag, you can use Param tag in nested. In this case, Parm tag contents will be the parameter of request that is sennt by Link tag. In this case, request will be sent by GET.

3.6.1.1.4 **Param**

Param tag sets parameter of request which is sent from browser to server by Frame tag. Param tag is always used with Link tag or Frame tag and it cannot exist independently. Fomat is described below.

<prefix:Param name="name_string" value="exp" />

In the above, prefix denotes the prefix when using tag library of IM-JavaEE Framework, name_string denotes parameter name when request is sent, and exp denotes its value.

If the request is sent by Link or Frame tag, the value can be obtained by getParameter of javax.servlet.http.HttpServletRequest at server side.

3.6.1.1.5 **Submit**

Submit tag changes the service specified by Form tag. Submit tag can be used only in the Form tag.

In many cases, you want to place multipe Submit buttons in Form tag and do the different processes depending on the pressed button. If you try to use <INPUT type="submit"> of HTML as Submit button, this type of process cannot be performed. In this case, Submit button generated by Submit tag makes above process more flexible. The format of Submit tag is the same as that of <INPUT> tag of HTML except for the followings.

- There is no type attribute.
- Attribute name should consists of all small letters.

Since this tag itself indicates Submit button, type attribute does not exist and application attribute and service attribute exist. These attributes indicate application ID and service ID respectively.

Both Form tag and Submit tag can specify application ID and service ID. If both are specified, the value of Submit tag that is actuallypressed is prioritized.

3.6.1.1.6 **SubmitLink**

SubmitLink tag is the link that submits form specified by Form tag. SubmitLink tag can be specified by both from inside and outside the Form tag.

If you want to submit the form not by Submit button from inside or outside of Form tag, but by link, this type of process cannot be performed directly in <A> tag of HTML. In this case, if the link is generated by SubmitLink tag, above process will be performed more flexibly. The format of SubmitLink tag is almost the same with <A> tag of HTML except for the followings.

- Following attributes do not exist.
  - charset
  - type
  - hreflang
  - rel
  - rev
  - target
- Attribute name should consists of all small characters.

Since this tag delegates Submit button, the attribute which should be defined by <FORM> tag of HTML cannot be specified. And application attribute and service attribute exist. These attributes indicate application ID and service ID respectively. There is also form attribute that specifies form to be submitted.

application attribute, service attribute and form attribute in SubmitLink tag are not required but the following limitations are set.

- If you specify this tag to the outside of Form tag, form attribute should be specified. In this case, submit is performed for the form specified by the form tag.
- If you specify this tag inside the Form tag, you cannot specify form attribute. In this case, submit is performed for the form that includs this tag.
- If you specify application attribute, you should specify service attribute.
- If you omit application attribute, you cannot specify service attribute.
- You can omit application attribute and specify service attribute. In this case, it is assumed that application attribute of the form submitted by this tag is specified as the application attribute.

3.6.1.1.7 **HelperBean**

HelperBean tag is the tag to use HelperBean. HelperBean is described in [3.6.1.2 HelperBean]. Format of HelperBean tag is described below.

<prefix:HelperBean id="bean_name" class="class_name" />

By using this tag, instance bean_name of class class_name can be used in JSP.

3.6.1.1.8 **Message**

Message tag is the tag to display message that has region support. This tag simply displays message that can be obtained in message framework (refer to [2.2.5 Message Framework]).

Following attributes can be set in Message tag.

- application
  It specifies application ID.
- key
  It specifies message key.
- locale
  It specifies locale when message is structured. If it is omitted, the locale that is determined by service framework will be used (refer to [3.3.1 Locale]).

MessageParam tag can also be specified as internal tag. MessageParam tag is the value that will be replaced to the variable in the message.

This tag uses getMessage(String, String, Object[], Locale) method of jp.co.intra_mart.framework.base.message.MessageManager internally. In this case, application attribute, key attribute, messageParam tag contents, and locale attribute support 1~4 argument of method respectively.

If you use Message tag and MessageParam tag as shown in [List 3-13 Example of Message Tag], same value as the result of getMessage method call of MessageManager is displayed just like the code in [List 3-14 Obtaining Message].

List 3-13 Example of Message Tag

```
<prefix:Message application="myapp" key="mykey" locale="ja_JP">
    <prefix:MessageParam value="hello" />
    <prefix:MessageParam value="world" />
</prefix:Message>
```

List 3-14 Obtaining Message

```
MessageManager manager = MessageManager.getMessageManager();
Object[] parameters = {"hello", "world"};
String message =
    manager.getMessage("myapp", "mykey", parameters, new Locale("ja", "JP"));
```

If Message tag and MessageParam tag are used by omitting locale as shown in [List 3-15 Example of Message Tag (locale is omitted], same value as the result of specified locale determined by getLocale method of ServiceManager is displayed.

List 3-15 Example of Message Tag (locale is omitted)

```
<prefix:Message application="myapp" key="mykey">
    <prefix:MessageParam value="hello" />
    <prefix:MessageParam value="world" />
</prefix:Message>
```

List 3-16 Obtaining Message (locale is omitted)

```
MessageManager manager = MessageManager.getMessageManager();
Object[] parameters = {"hello", "world"};
ServiceManager service = ServiceManager.getServiceManager();
Locale locale = service.getLocale(request, response);
String message =
      manager.getMessage("myapp", "mykey", parameters, locale);
```

3.6.1.1.9 **MessageParam**

MessageParam tag is the tag that specifies parameter which should be passed to Message tag. MessageParam tag can be used only inside the Message tag. This tag has value attribute only. If you specify multiple parameters, this tag should be described multiple times. In this case, it is equivalent to the array with its elements in the same sequence stated. (refer to [3.6.1.1.8 Message])

3.6.1.2 HelperBean

If you obtain data for initial display, two methods can be considered. One is to obtain the data and pass the data for display when screen is moved. Another one is to obtain the data when screen is displayed. In the latter situation, it will be more useful if you obtain the data for initial display by using HelperBean.

HelperBean structure is described in [Figure 3-24 HelperBean Structure].

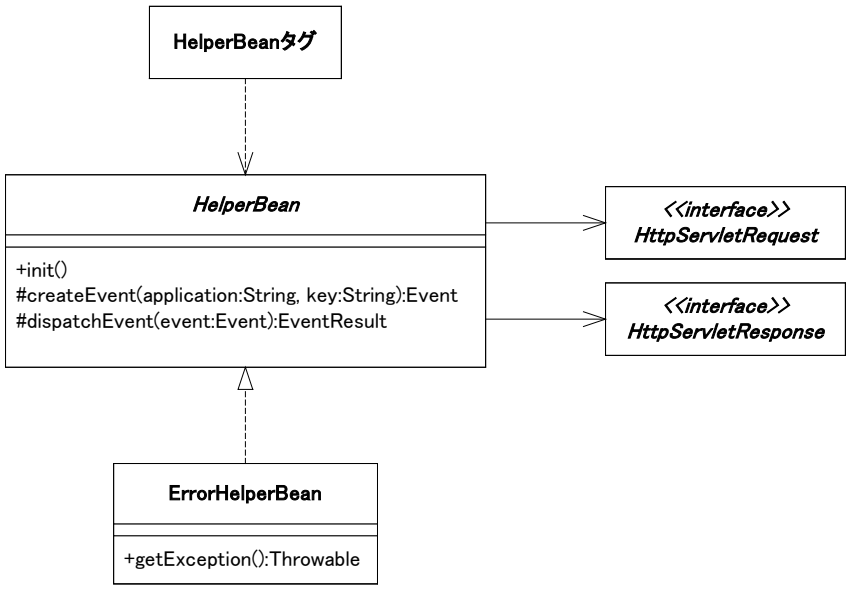


Figure 3-24 HelperBean Structure

HelperBean can be used by using following tag library.

```
<%@ taglib prefix="prefix" uri="http://www.intra-mart.co.jp/taglib/core/framework" %>
・・・
<prefix:HelperBean id="bean_name" class="class_name" />
```

In the above, prefix denotes the prefix when using tag library of IM-JavaEE Framework, bean_name denotes variable name when used as script variable in JSP, and class_name denotes the class name of HelperBean.

Class that can be specified in class_name should meet the following conditions.

- It should be the subclass of jp.co.intra_mart.framework.base.web.bean.HelperBean.

- Constructor that meets all the following conditions should be there.
  - It is public
  - No argument.
  - jp.co.intra_mart.framework.base.web.bean.HelperBeanException or its subclass is specified in throws section.

In this tag, class specified by class_name is newly generated, and request and response are set and the initialization method init() will be called. This condition is shown in [Figure 3-25 Initialization of HelperBean].

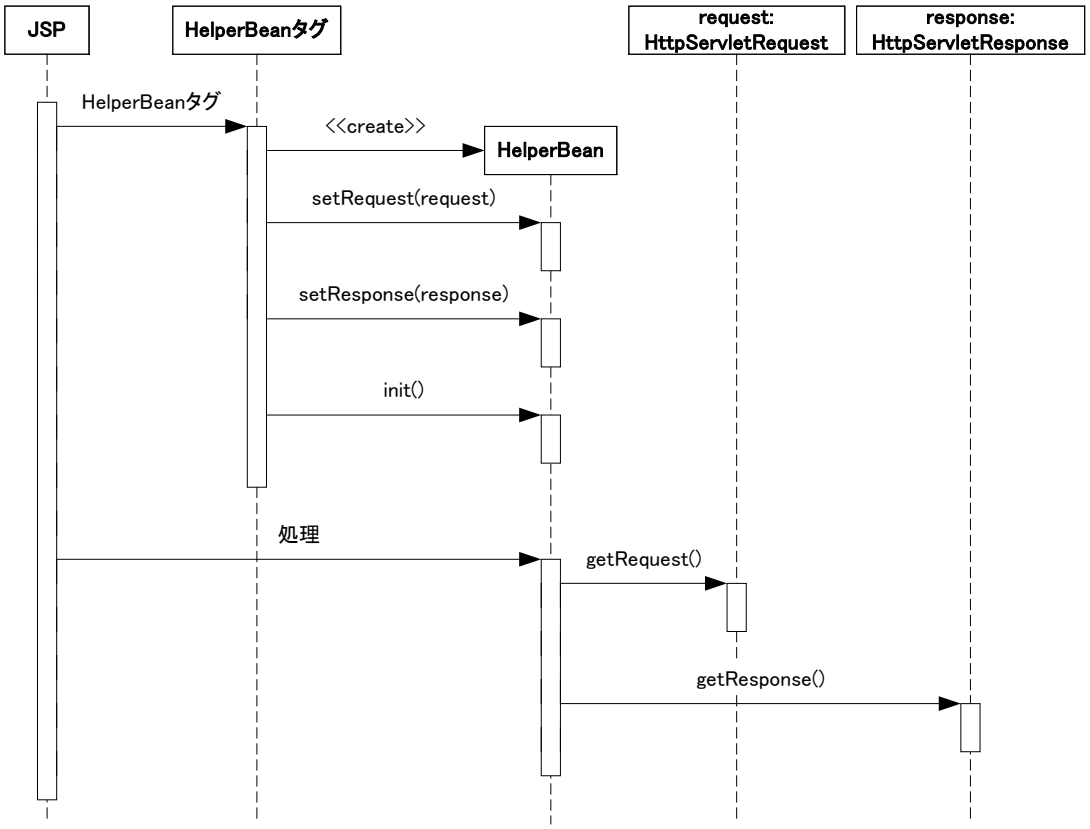


Figure 3-25 Initialization of HelperBean

HelperBean method, which is called after initialization in init method or HelperBean tag, can obtain request or response by getRequest method or getResponse method. If getRequest method or getResponse method is called in the constructor, null will be returned since nothing is set.

In HelperBean, createEvent method or dispatchEvent method that does event process simply is provided. Its usage is the same with the contents shown in [Figure 3-16 Use Event Framework from ServiceControllerAdapter]. Sample code when you call the event from inside the HelperBean is shown in [List 3-17 HelperBean].

List 3-17 HelperBean Sample

```
・・・
import java.util.Collection;
import jp.co.intra_mart.framework.base.web.bean.HelperBean;
import jp.co.intra_mart.framework.base.web.bean.HelperBeanException;
・・・

public class TestHelperBean extends HelperBean {

	private String keyword;

	public TestHelperBean() throws HelperBeanException {
		super();
		this.keyword = null;
	}

	public void init() {
		this.keyword = (String)(getRequest().getAttribute("keyword"));
	}

	public Collection getSearchResult() {
		SearchEvent searchEvent = (SearchEvent)createEvent("sample", "search");
		searchEvent.setKeyword(this.keyword);
		SearchEventResult result = (SearchEventResult)dispatchEvent(searchEvent);

		return result.getSearchList();
	}
}
```

[List 3-17 HelperBean] searches information based on keyword specified by request attribute"keyword". Information search is obtained by SearchEvent class and SearchEventResult class using event framework of IM-JavaEE Framework. Detail of event framework of IM-JavaEE Framework is described in [4 Event Framework].

3.6.2 Screens other than JSP

In IM-JavaEE Framework, it is possible to be moved on screens other than JSP, but there is no special definition for standard usage. In this case, contents of screen display should be originally implemented depending on the transition destination screen (If transition is made to Servlet, output is made to HttpServletResponse, it is redirected to completely different URL and so on). On these original screens, tag library or HelperBean described in [3.6.1 JSP] cannot be used.

3.7 Exception Process

In the service framework of IM-JavaEE Framework, exception can be detected at locations shown in [Figure 3-26 Exception of Service Framework].

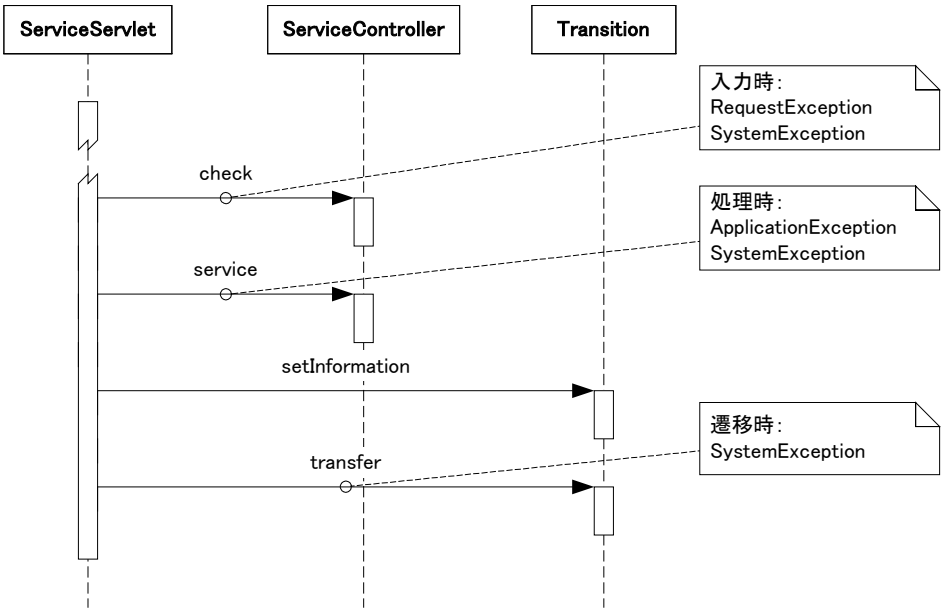


Figure 3-26 Exception of Service Framework

- At input time (check method of ServiceController)
- At process time (service method of（ServiceController)
- At screen transition time (transfer method of Transition)

3.7.1 Exception Process at Input Time

In check method of ServiceController, following exceptions or its subclasses can be thrown.

- jp.co.intra_mart.framework.base.service.RequestException
- jp.co.intra_mart.framework.system.exception.SystemException

When the developer implements check method of ServiceController, RequestException and its sub class should be generated if there is any error or defficiency in the input contents. Other exceptions should be generated as SystemException and its sub class.

Behavior when RequestException is occurred in check method of ServiceController is shown in [Figure 3-27 At RequestException Occurrence Time].

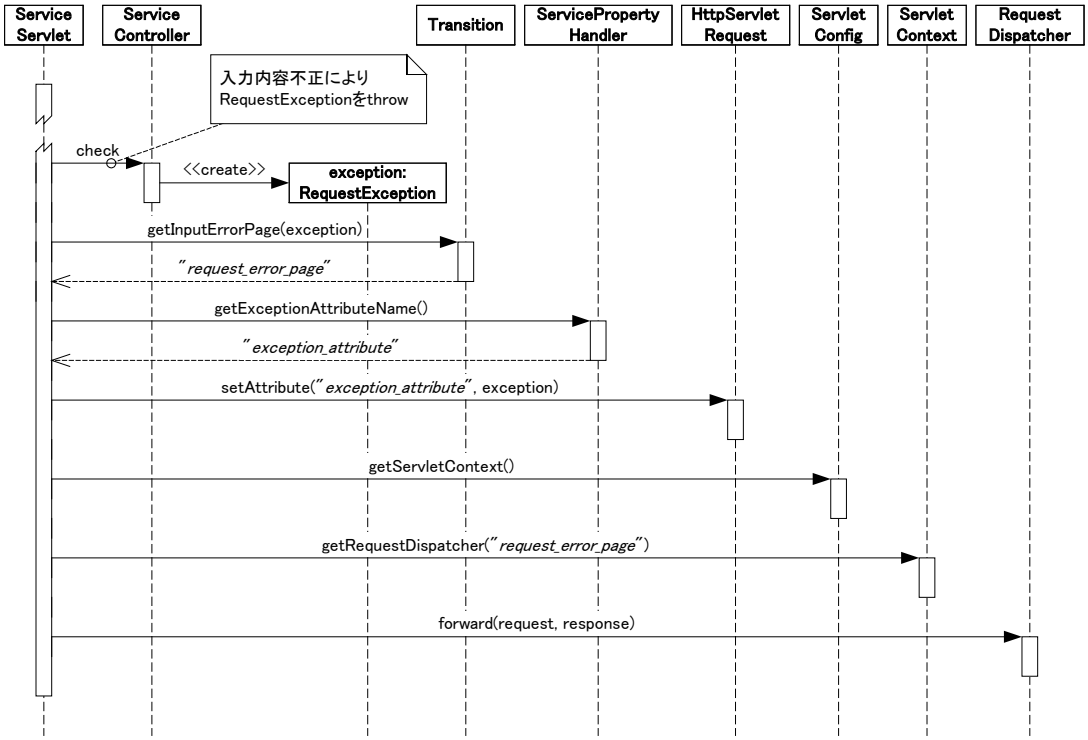


Figure 3-27 At RequestException Occurrence Time

Behavior when SystemException is occurred in check method of ServiceController is shown in [Figure 3-28 At SystemException Occurrence Time].

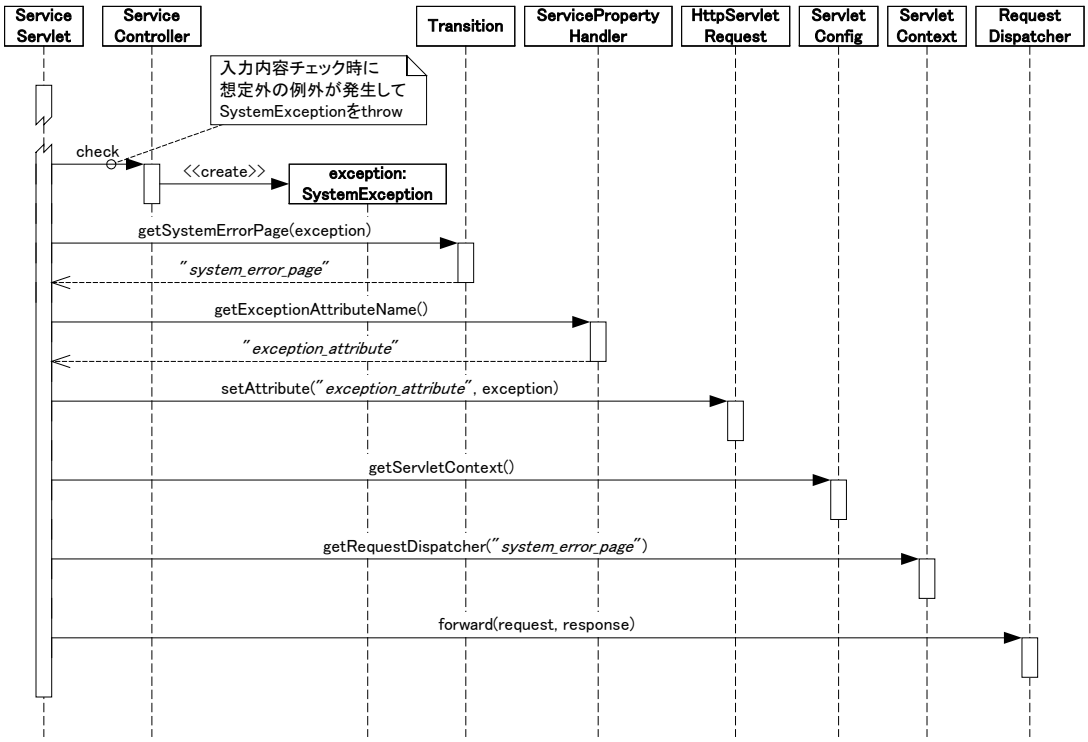


Figure 3-28 At SystemException Occurrence Time (check method)

The only difference betwee [Figure 3-27 At RequestException Occurrence Time] and [Figure 3-28 At SystemException Occurrence Time] is the operation by exception type which was thrown by check method. If

checke method throw RequestException (or its subclass) as an exception, ServiceServlet calls getInputErrorPage method of Transition, obtains the input exception page and moves to the page. On the other hand, if check method throw SystemException (or its subclass) as an exception, ServiceServlet calls getSystemErrorPage method of Transition, obtain the system exception page and forward to the page.

Exception information (exception that was throw by check method) will be set to attribute of the request. Attribute name was set by the property, and will be obtained by getExceptionAttributeName of ServicePropertyHandler. You can obtain exception information through this attribute on the forwarded transition destination page.

3.7.1.1 Log Output

ServiceServlet outputs the log when the exception is thrown by check method. Followings are exception types and the log levels to be output.

Exception Types	Log Level
RequestException	Does not output
SystemException	error

3.7.2 Exception Process at Process Time

In service method of ServiceController, the following exceptions or its subclass can be thrown.

- jp.co.intra_mart.framework.system.exception.ApplicationException
- jp.co.intra_mart.framework.system.exception.SystemException

When the developer implements service method of ServiceController, ApplicationException and its sub class should be generated if there are such defects that are expected in user operations as duplicate registration of data with the same key or access to the data already deleted. Other exceptions should be generated as SystemException or its sub class.

Behavior when ApplicationException is occurred in service method of ServiceController is shown in [Figure 3-29 At ApplicationException Occurrence Time].

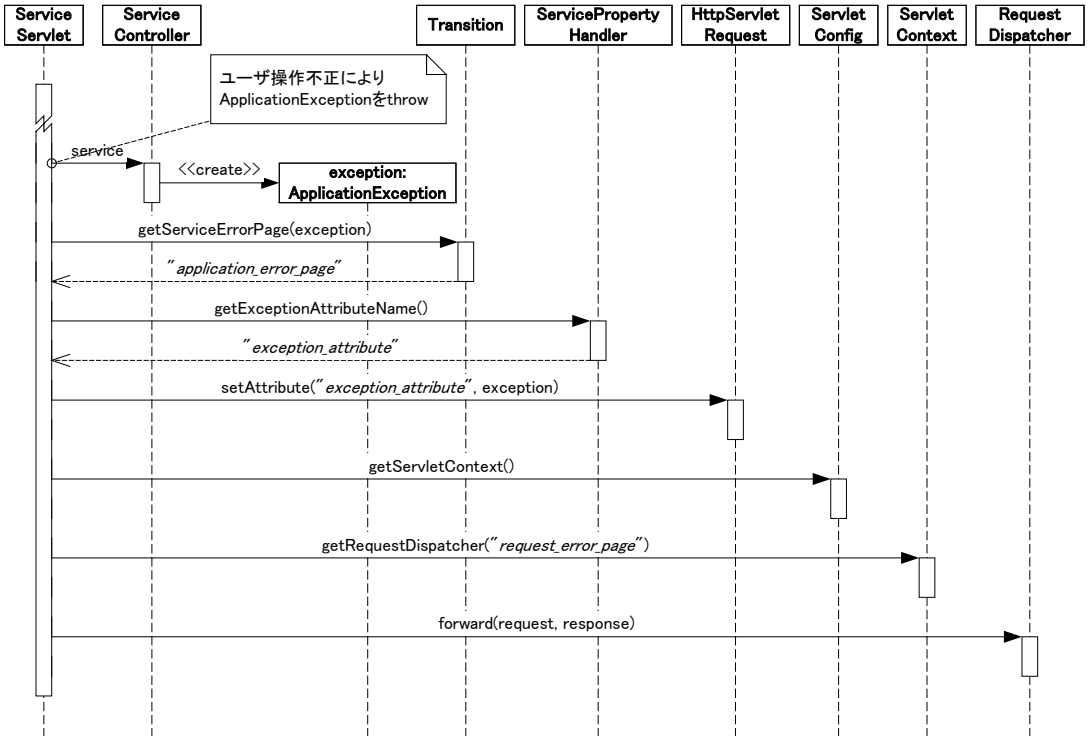


Figure 3-29 At ApplicationException Occurrence Time

Behavior when SystemException is occurred in service method of ServiceController is shown in [Figure 3-30 At SystemException Occurrence Time (service method)].

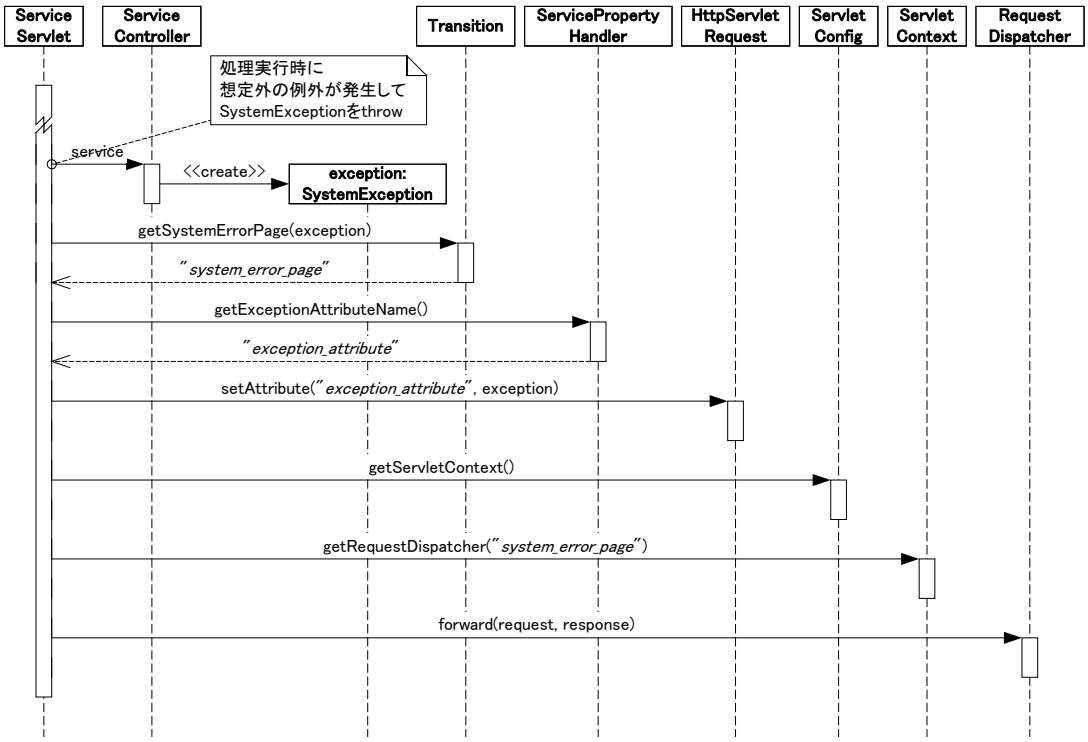


Figure 3-30 At SystemException Occurrence Time (service method)

The only difference between [Figure 3-29 At ApplicationException Occurrence Time] and [Figure 3-30 At SystemException Occurrence Time (service method)] is the operation by exception type that is thrown by service

method. If service method throws ApplicationException (or its subclass) as exception, ServiceServlet calls getServiceErrorPage method of Transition, obtains the application exception page and moves to the page. On the other hand, if service method throws SystemException (or its subclass) as exception, ServiceServlet calls getSystemErrorPage method of Transition, obtains the system exeption page and forword to the page.

Exception information (exception that is thrown by service method) is set to attribute of the request. Attribute name was set by the property and will be obtained by getExceptionAttributeName of ServicePropertyHandler. Exception information can be obtained through this attribute on the forwarded transition destination page.

3.7.2.1 Log Output

ServiceServlet outputs log when the exception is thrown from service method. Followings are the exception type and the log level to be output.

Exception Types	Log Level
ApplicationException	warn
SystemException	error

error level was output if ApplicationException is thrown before Version 7.1, but it has been changed to output warm level later than 7.2. The purpose of this change was to control the log output when business exception is occurred, since ApplicationException is used as business exception.
Log output at ApplicationException occurrence time can be changed not to output the log by changing the logger setting. Setting example when setting file of Logback is used is described below.

```
   ・・・

<logger name="foo.bar.*">
    <level value="warn" />
</logger>

   ・・・
```

Setting file is stored in the below directly path in intra-mart.

- conf/log/im_logger.xml

3.7.3 Exception Process at Screen Transition Time

After the process performed by service method of ServiceController is completed successfully or if ServiceController is not associated for the request, it will be moved to the next page by transfer method of Transition. Since the exception at screen transition time is the exception outside the scope of IM-JavaEE Framework, specification of transfer method is defined to throw jp.co.intra_mart.framework.system.exception.SystemException or its sub class as an exception.

Sequence figure when system exception is occurred in transfer method is shown in [「Figure 3-31 At SystemException Occurrence Time (transfer method)].

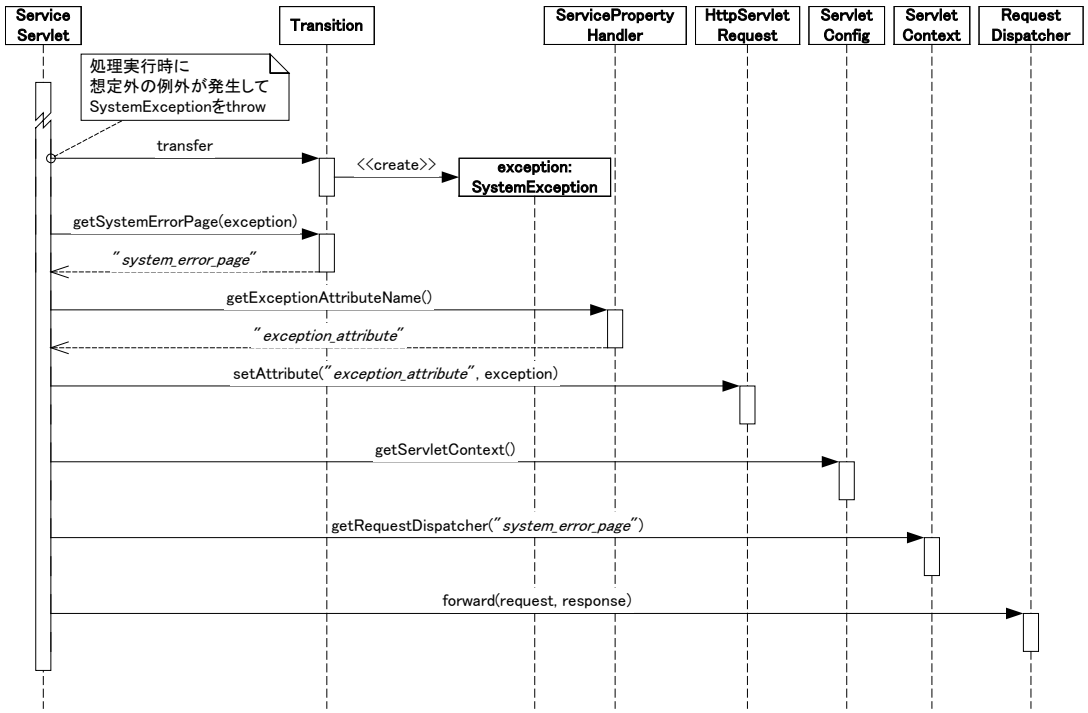


Figure 3-31 At SystemException Occurrence Time (transfer method)

The operation after transfer method of Transition throw SystemException or its subclass is the same with the case of SystemException shown in [3.7.1 Exception Process at Input Time] or [3.7.2 Exception Process at Process Time]

3.7.4 Exception Process at Screen Output Time

There is no special definition about exception process at screen output time in IM-JavaEE Framework.

3.7.5 Obtaining Error Page

In IM-JavaEE Framework, transition destination for error is obtained from Transition. How error page is obtained in DefaultTransition, which is a sub class of Transition, is shown in [Figure 3-32 Obtaining Error Page].

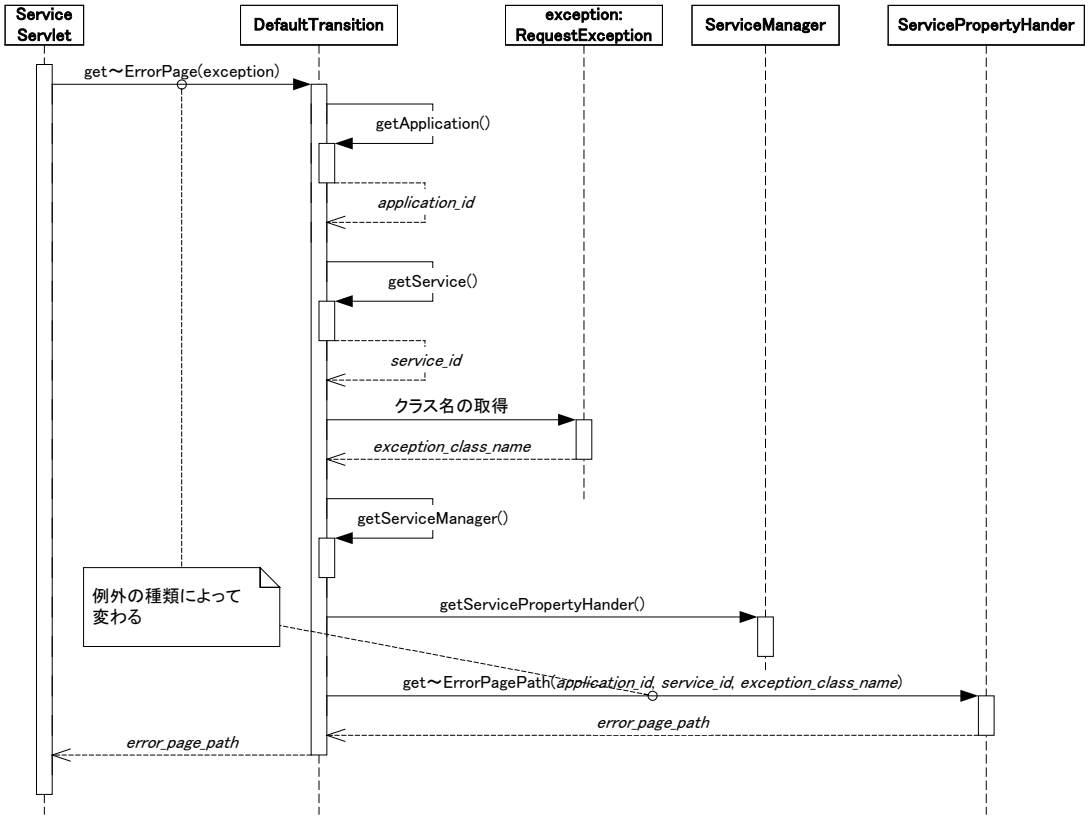


Figure 3-32 Obtaining Error Page

In [Figure 3-32 Obtaining Error Page], get~ErrorPage or get~ErrorPagePath will vary depending on the type and location of exceptions that occurred. List of method called by [Table 3-10 Method at Exception Occurrence Time] is shown.

Table 3-10 Method at Exception Occurrence Time

Type of exception occurred	Main method exception occurrs	Method to be called
RequestException	check() of ServiceController	getInputErrorPage getInputErrorPagePath
ApplicationExcpetion	service() of ServiceController	getApplicationErrorPage getApplicationErrorPagePath
SystemException	check() of ServiceController service() of ServiceController transfer() of Transition	getSystemErrorPage getSystemErrorPagePath

3.7.6 Displaying Error Page

Exception process shown in [3.7.1 Exception Process at Input Time] ~ [3.7.3 Exception Process at Screen Transition Time] is performed and it is moved to the error page obtained by [3.7.5 Obtaining Error Page] and displayed. Exception information (exception that was thrown) can be obtained from attribute of the request at this point. Attribute name of the request that includes the exception information can be obtained by getExceptionAttributeName of ServicePropertyHandler.

If JSP is used as error page, exception information can be obtained by using HelperBean tag and jp.co.intra_mart.framework.base.web.bean.ErrorHelperBean which are the extended tag of IM-JavaEE Framework. In this case, completed class name that includes ErrorHelperBean or its subclass is specified to the class attribute of

HelperBean[5]. ErrorHelperBean will be returned to default by this and the exception information can be obtained by getException method.

Example that obtains exception information in JSP is shown in [List 3-18 Obtaining Exception Information by ErroHelperBean]. foo.SampleErrorHelperBean is the subclass of jp.co.intra_mart.framework.base.web.bean.ErrorHelperBean here.

List 3-18 Obtaining Exception Information by ErroHelperBean

```
    ・・・
<%@ taglib prefix="im_j2ee" uri="http://www.intra-mart.co.jp/taglib/core/framework" %>
    ・・・
<im_j2ee:HelperBean id="errorBean" class="foo.SampleErrorHelperBean">
    ・・・
<%
    Throwable e = errorBean.getException();
    ・・・
%>
```

3.8 Internationalization

Service framework of IM-JavaEE Framework is designed by considering internationalization. Followings are the target of the internationalization.

- Display
- Transition

3.8.1 Internationalization of Display

If the region support is provided for display, the simplest method is to internationalize JSP. In this case, use layout which is common in any locale and switch the displaying contents only by using Message tag (refer to [Figure 3-33 Internationalization by Message Tag].

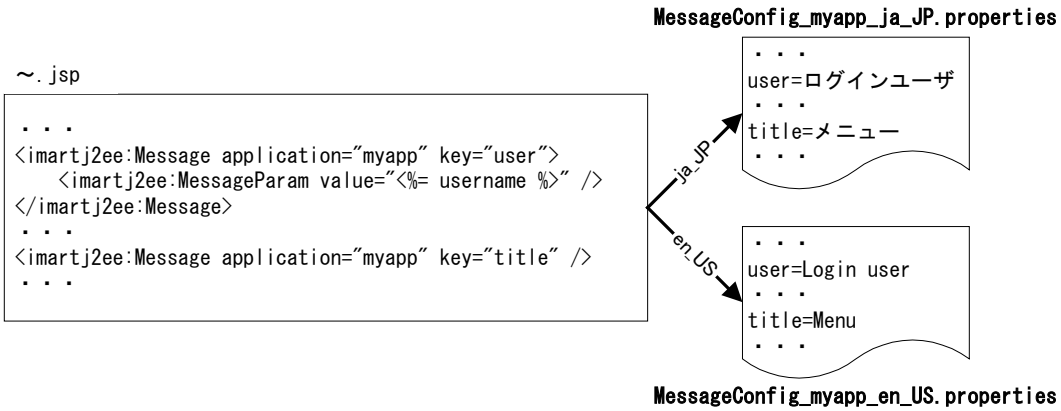


Figure 3-33 Internationalization by Message Tag

3.8.2 Internationalization of Transition

In service framework, the value that has region support can be provided for the following properties for each locale.

- ServiceController
- Transition

[5] Please refer to [3.6.1.2 HelperBean] for detail of HelperBean tag.

- Transition destination

By using this, different processes or transition destination can be set for each locale. In this case, property should be provided for each locale if necessary (refer to [Figure 3-34 Internationalization by Property])

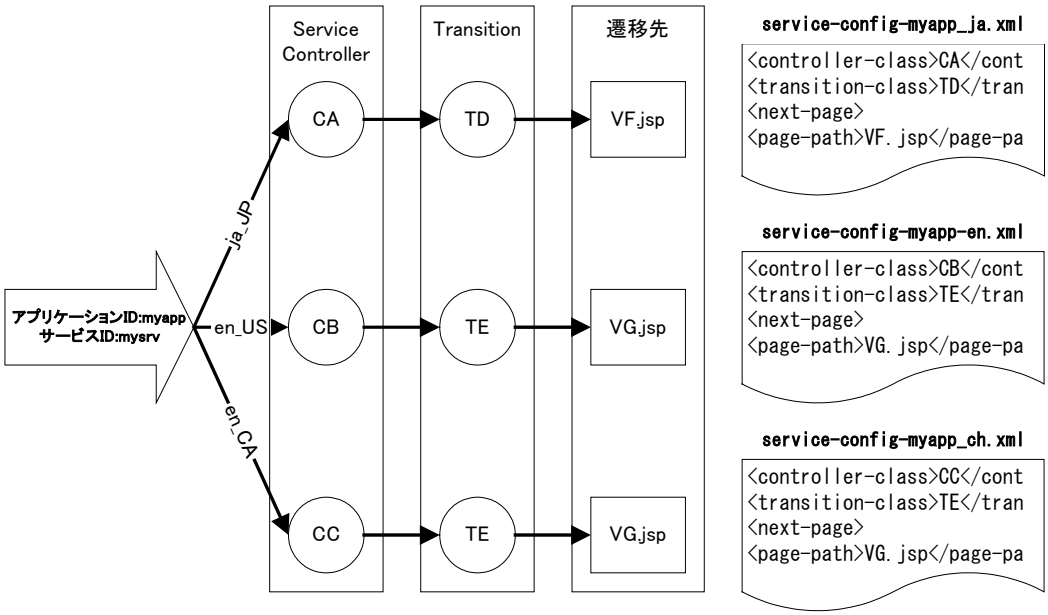


Figure 3-34 Internationalization by Property

With regard to the transition destinations, you can make the transition to the same one JSP and achieve region support by different character strings as shown in [3.8.1 Internationalization of Display]. If you want to change the layout depending on the region, different JSPs may better be provided as transition destinations as shown in [Figure 3-34 Internationalization by Property].Both methods have merits and demerits, so it is better to select the method which is appropriate for the system to be used.

4 Event Framework

4.1 Overview

There are various configurations for process method of business logic, and it is difficult to integrate the methods. However, the procedure has the common points for 1) Generate input information that are necessary for the process, 2) Processes based on the input information, and 3) Return the process process.

On the event framework, common parts are made into framework with these train.

Remarks:
In this chapter, possibility of excecuting process through EJB is described, but J2EE application server should support EJB when you use this function.

4.2 Structure

4.2.1 Structure Element

Event Framework is structured by the followings.

- Event
- EventListenerFactory
- EventListener
- EventResult
- EventTrigger

The relationship of these are described in [Figure 4-1 Class Figure of Event Framework].

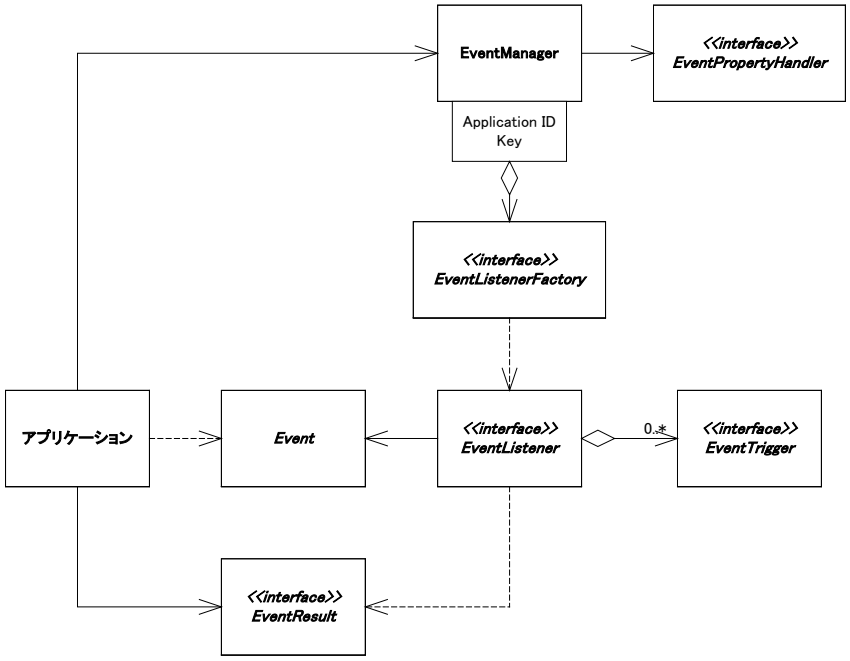


Figure 4-1 Class Figure of Event Framework

4.2.2 Event Process

Overview when you move business logic by event framework of IM-JavaEE Framework is shown in [Figure 4-2 Event Process Overview].

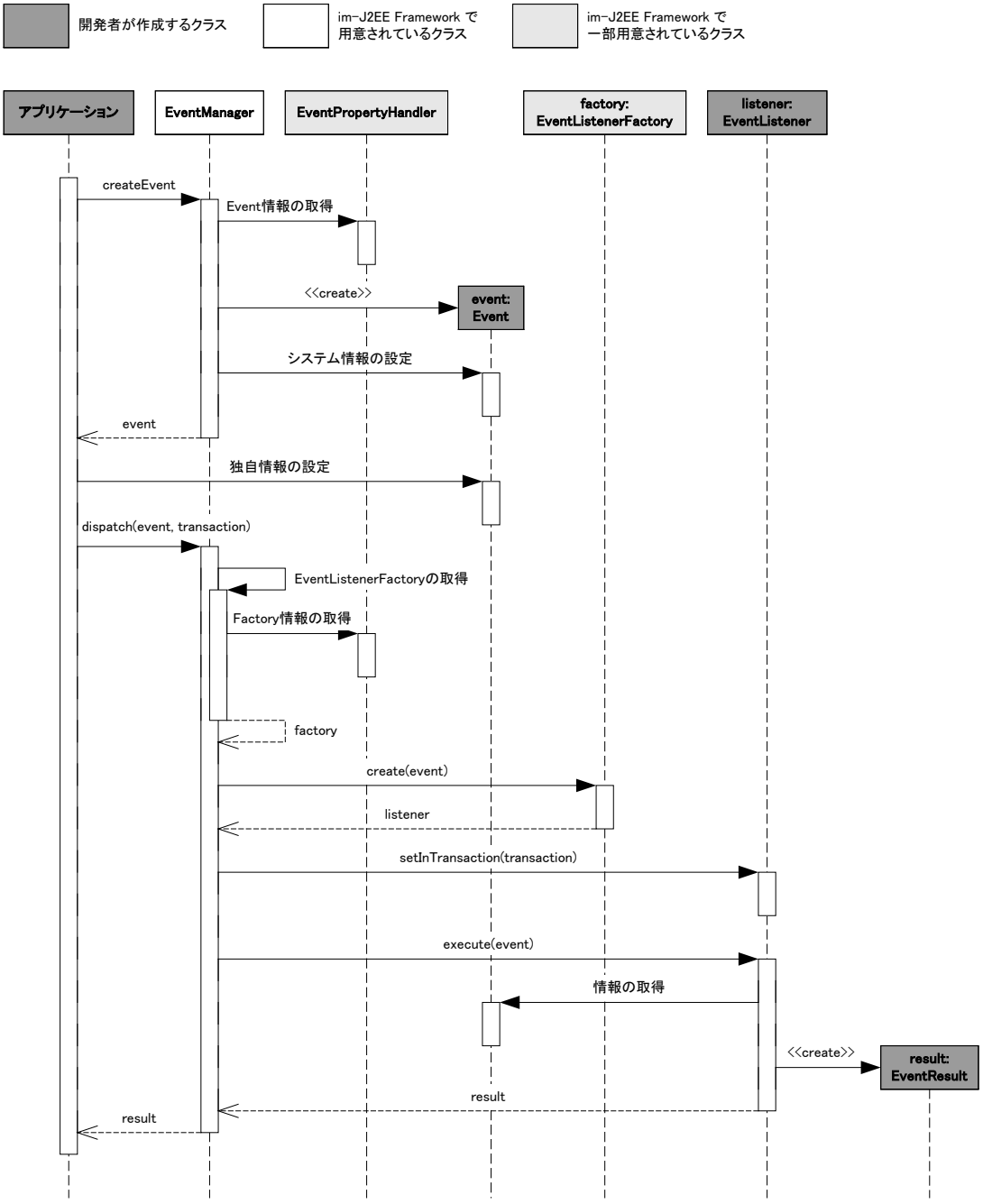


Figure 4-2 Event Process Overview

1. Application generates and obtains Event that corresponds to the application ID and the event key by using createEvent method of EventManager.
2. Application sets the information which will be required when business logic processing for the Event.
3. Application does process request of business logic by using dispatch method of EventManager.
4. EventManager obtains EventListenerFactory that corresponds to the application ID and the event key by using EventPropertyHandler.
5. EventManager obtains EventListener by using create method of EventListenerFactory.
6. EventManager passes the flag that judges if it is in the transaction by using setInTransaction method of EventListener.
7. EventManager does process request by using execute method of EventListener and passing the Event.
8. Eventlistener, which is the business logic, does the process and returns EventResult as results.

4.3 Structure Element Detail

4.3.1 Event

Event is the input information of EventListener which is to be the business logic. The developer of event process should create the Event class which meets the following conditions.

- Inherit jp.co.intra_mart.framework.base.event.Event class.
- public default constructor (constructor with no argument) exsists.
- Instance variable that has the following field name is not declared.
  - application
  - key
  - info
- It can be serialized. If you use EJB, Event will be passed to the remote environment. Event contents may be currupted if serializing cannot be performed.

If you execute business logic by using event framework of IM-JavaEE Framework, Event should call createEvent method of EventManager and obtain. The developer should not generate Event derectly by new or reflection. After obtaining Event from EventManager, input information required when starting business logic for the Event will be set. These flow are shown in [Figure 4-3 Event Generation].

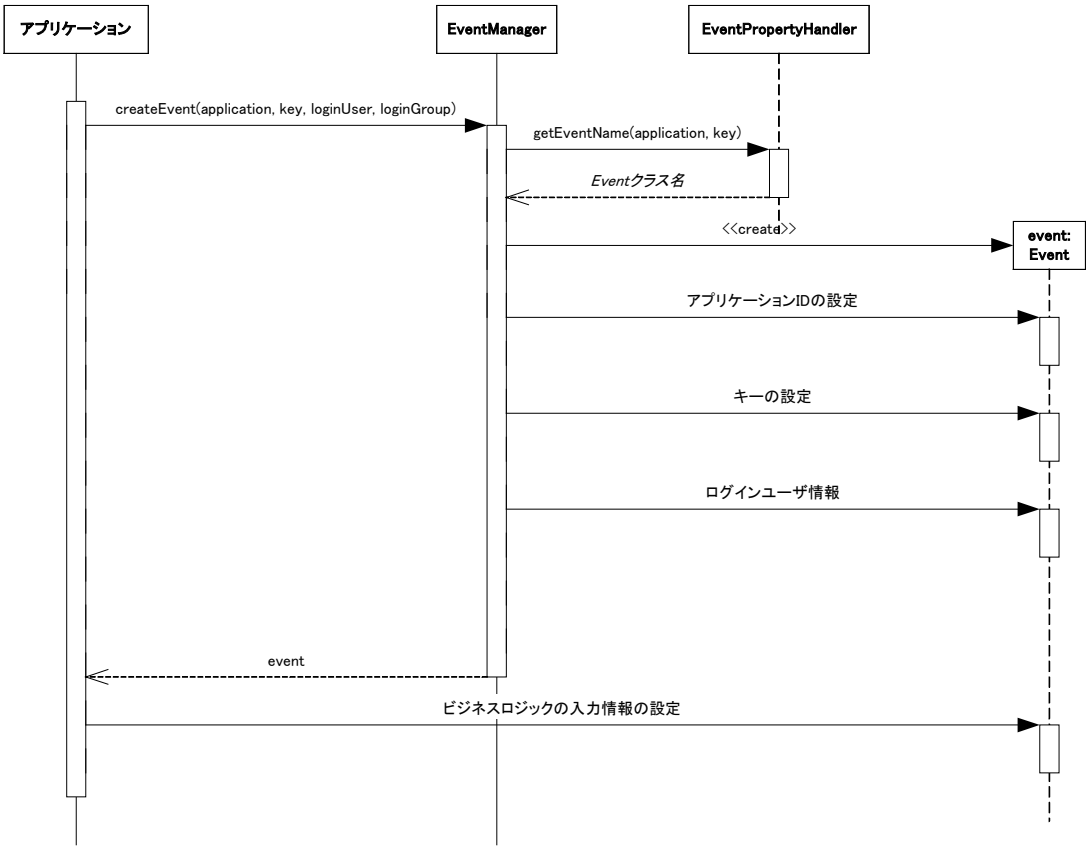


Figure 4-3 Event Generation

If business logic (EventListener) does not need to input information other than information set by the system (application ID, event key, and log-in information), you do not have to set Event that corresponds to the application ID and event key in the property. In this case, getEventName method of EventPropertyHandler should be designed to return null. (refer to [エラー！参照元が見つかりません。 エラー！参照元が見つかりません。]). If

getEventName method of EventPropertyHandler returns null, EventManager generates jp.co.intra_mart.framework.base.event.EmptyEvent (refer to [Figure 4-4 Event Generation (no event setting)].

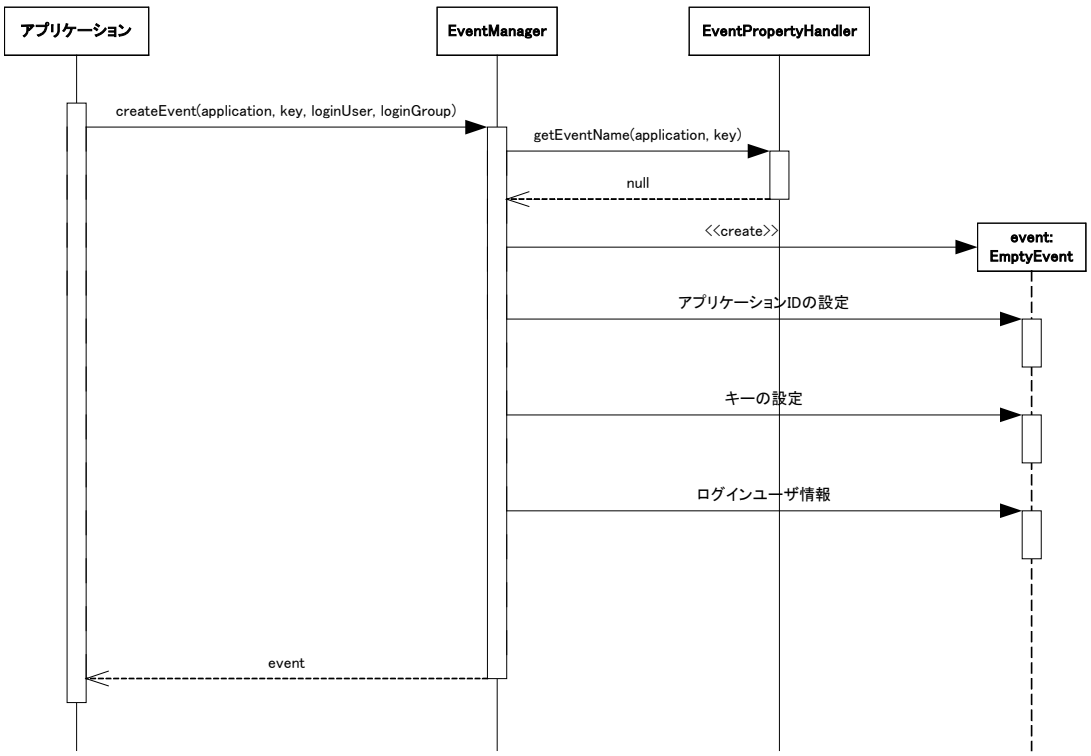


Figure 4-4 Event Generation (no event setting)

4.3.2 EventListenerFactory

The main role of EventListenerFactory is to generate EventListener. EventListenerFactory itself is not related to the business logic execution. Business logic execution is closely related to EventListener. Importance of existence of EventListenerFactory is described here.

Normally, some preparation needs to be done before you execute business logic. However, this preparation is vary greatly depending on the execution configuration of business logic. For example, if the business logic is implemented by Java classes on Java Virtual Machine (VM), necessary tasks before executing business logic would only be the generation of business logic and subsequent tasks are performed by simple method calls. On the other hand, if business logic is executed in the remote environment as represented by EJB[6], it is necessary to obtain Home object or Remote object as the necessary tasks before executing business logic, and the procedures for the execution would be different from the case in which method of Java class is called directly as described before (refer to [Figure 4-5 Difference depending on the Implementation of Business Logic]).

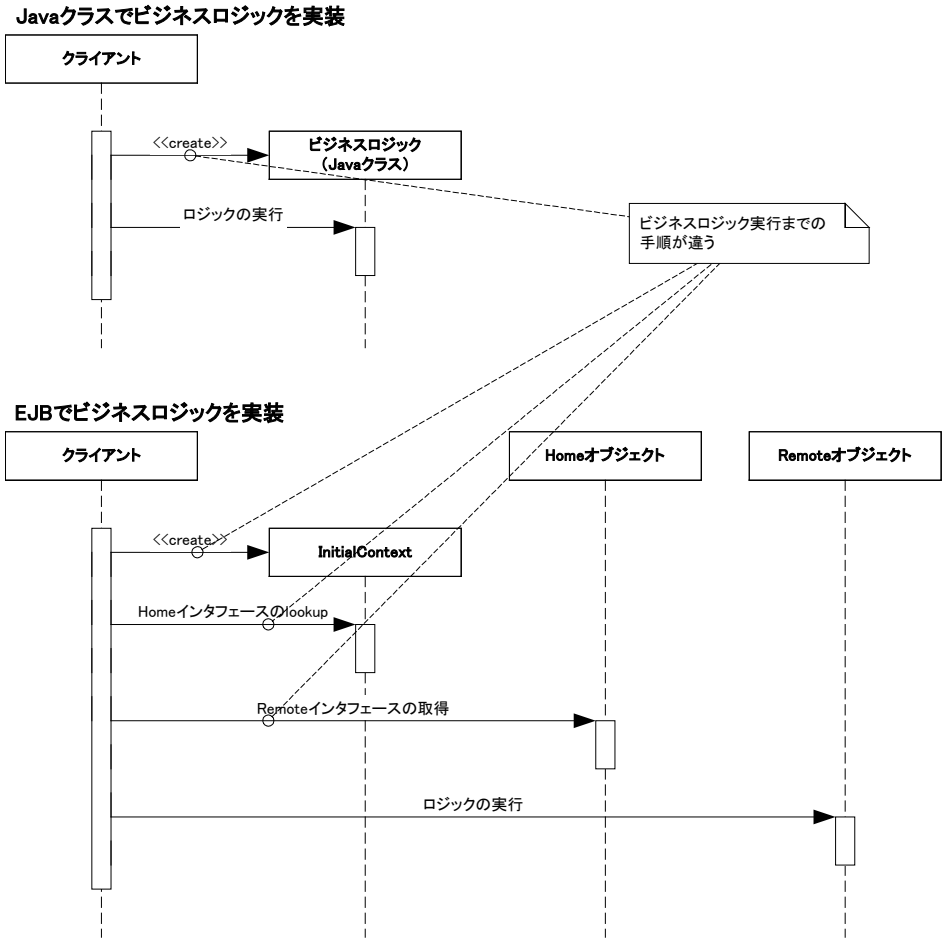


Figure 4-5 Difference depending on the Implementation of Business Logic

On IM-JavaEE Framework, above problem is solved by dividing it into EventListenerFactory and EventListener. EventListenerFactoryprovides abstraction for the generation and preparation of connection part to the business logic, while EventListener is an execution part of business logic.

One EventListenerFactory can be associated with the combination of application ID and event key. If dynamic loading is not set (if the return value of isDynamic method of EventPropertyHandler is false), EventListenerFactory is cacheed internally (refer to [Figure 4-6 EventListenerFactory Generation (with cache, already generated)]. If not, EventListenerFactory will be generated everytime (refer to [Figure 4-7 EventListenerFactory Generation (no cache or not generated].)

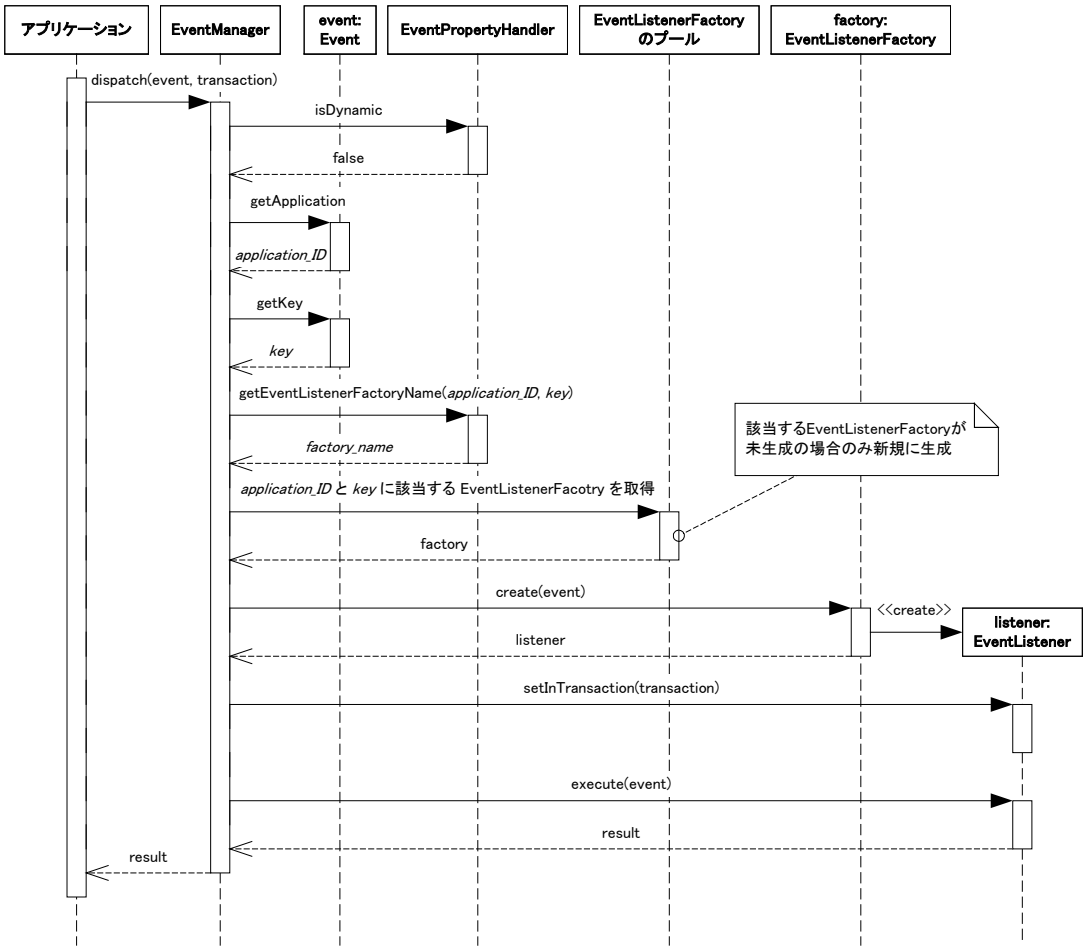


Figure 4-6 EventListenerFactory Generation (with cache, already generated)

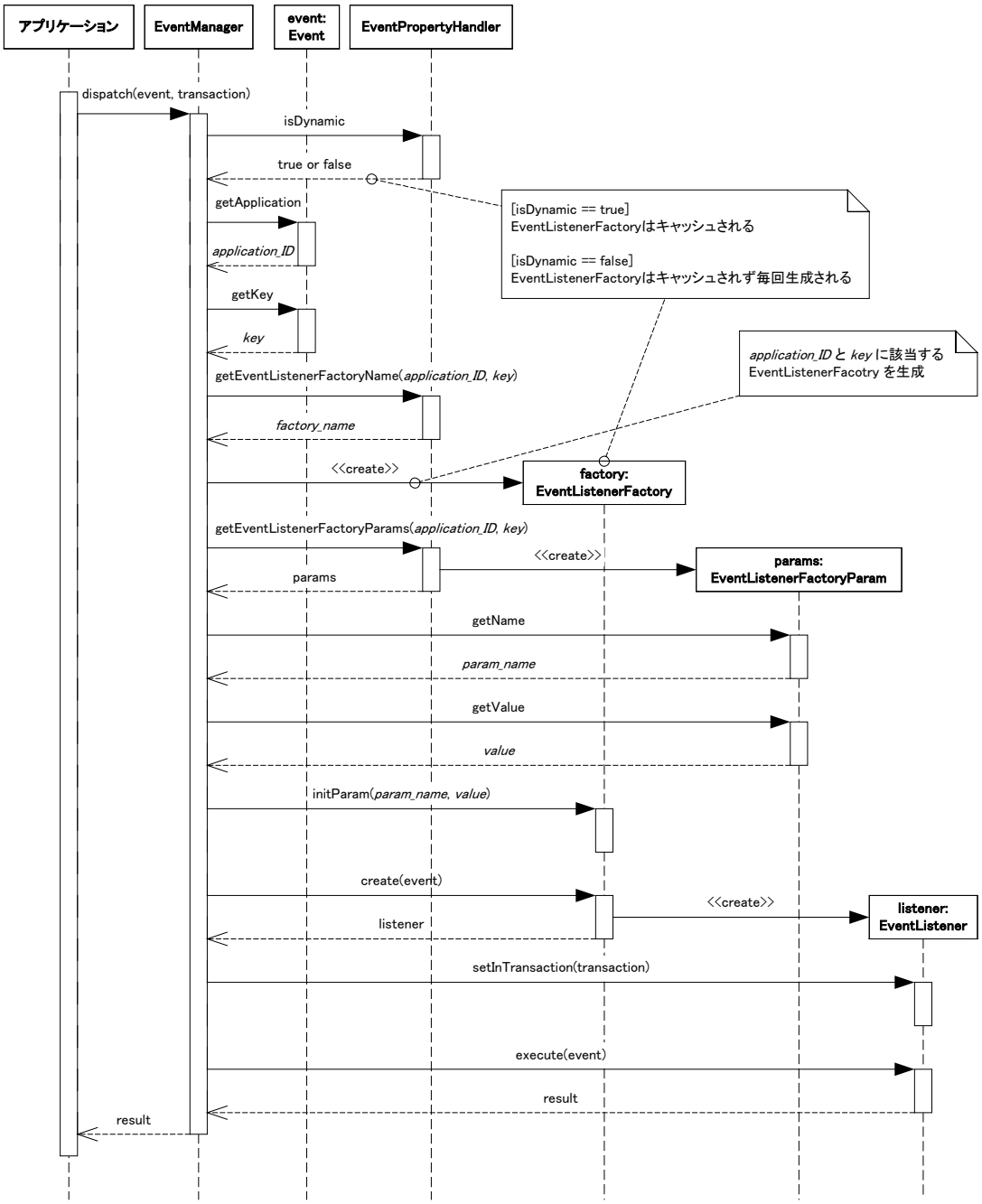


Figure 4-7 EventListenerFactory Generation (no cache or not generated)

4.3.2.1 EventListenerFactory provided as Standard

EventListenerFactory can also be created by the developer him/herself, but popular ones are already provided from the beginning. Followings are the EventListenerFactory provided from the beginning.

- StandardEventListenerFactory
- GenericEventListenerFactory
- StandardEJBEventListenerFactory
- GenericEJBEventListenerFactory

All of these belong to the package jp.co.intra_mart.framework.base.event.

4.3.2.1.1 StandardEventListenerFactory

StandardEventListenerFactory generates StandardEventListener. create method of StandardEventListenerFactory generates instance of StandardEventListener everytime it is called. Instance of StandardEventListener will not be cacheed. Please refer to [Figure 4-8 StandardEventListener Generation].

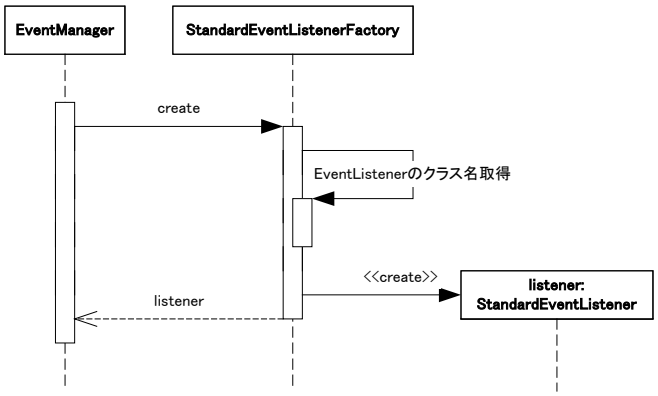


Figure 4-8 StandardEventListener Generation

If this EventListenerFactory is used, initialization parameter shown in [Table 4-1 Initialization Parameter of StandardEventListenerFactory] should be obtained by getEventListenerFactoryParams method of EventPropertyHandler.

Table 4-1 Initialization Parameter of StandardEventListenerFactory

Parameter Name	Parameter Contents
listener	Complete class name that includes the package of StandardEventListener to be generated.

4.3.2.1.2 GenericEventListenerFactory

GenericEventListenerFactory generates GenericEventListener. create method of GenericEventListenerFactory generates instance of GenericEventListener everytime it is called. Instance of StandardEventListener will not be cacheed. Please refer to [Figure 4-9 GenericEventListener Generation].

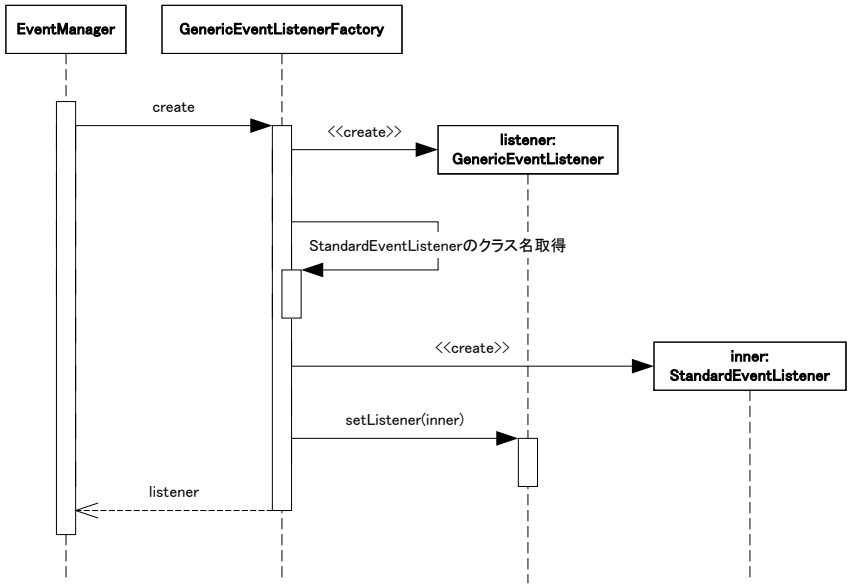


Figure 4-9 GenericEventListener Generation

If this EventListenerFactory is used, initialization parameter shown in [Table 4-2 Initialization Parameter of GenericEventListenerFactory] should be obtained by getEventListenerFactoryParams of EventPropertyHandler.

Table 4-2 Initialization Parameter of GenericEventListenerFactory

Parameter Name	Parameter Contents
listener	Complete class name that include the package of StandardEvent Listener to be included.

4.3.2.1.3 StandardEJBEventListenerFactory

StandardEJBEventListenerFactory generates StandardEJBEventListener. Once it generates StandardEJBEventListener by create method, the instance will be cacheed to instance of StandardEJBEventListenerFactory. Please refer to [「Figure 4-10 StandardEJBEventListener Generation].

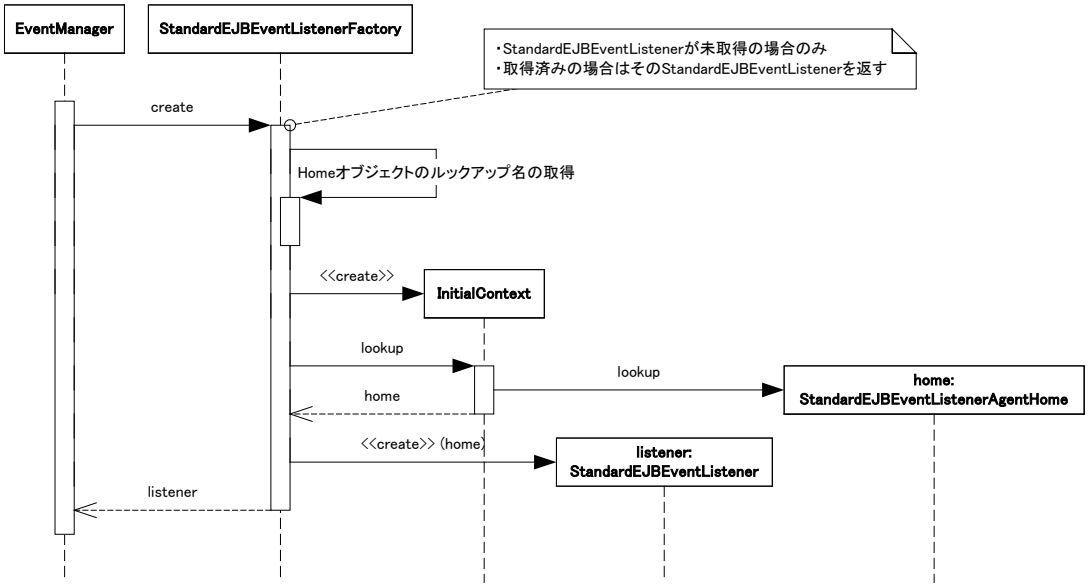


Figure 4-10 StandardEJBEventListener Generation

Initialization parameter shown in [Table 4-3 Initialization Parameter of StandardEJBEventListenerFactory] should be obtained by getEventListenerFactoryParams method of EventPropertyHandler.

Table 4-3 Initialization Parameter of StandardEJBEventListenerFactory

Parameter Name	Parameter Contents
home	Name when look up the Home object of EJB that is required in StandardEJBEventListener which is to be generated.

4.3.2.1.4 GenericEJBEventListenerFactory

GenericEJBEventListenerFactory generates GenericEJBEventListener to be used when StandardEventListener is called through EJB. Once GenericEJBEventListener is generated by create method, the instance will be cacheed to instance of GenericEJBEventListenerFactory. Please refer to [Figure 4-11 GenericEJBEventListener Generation].

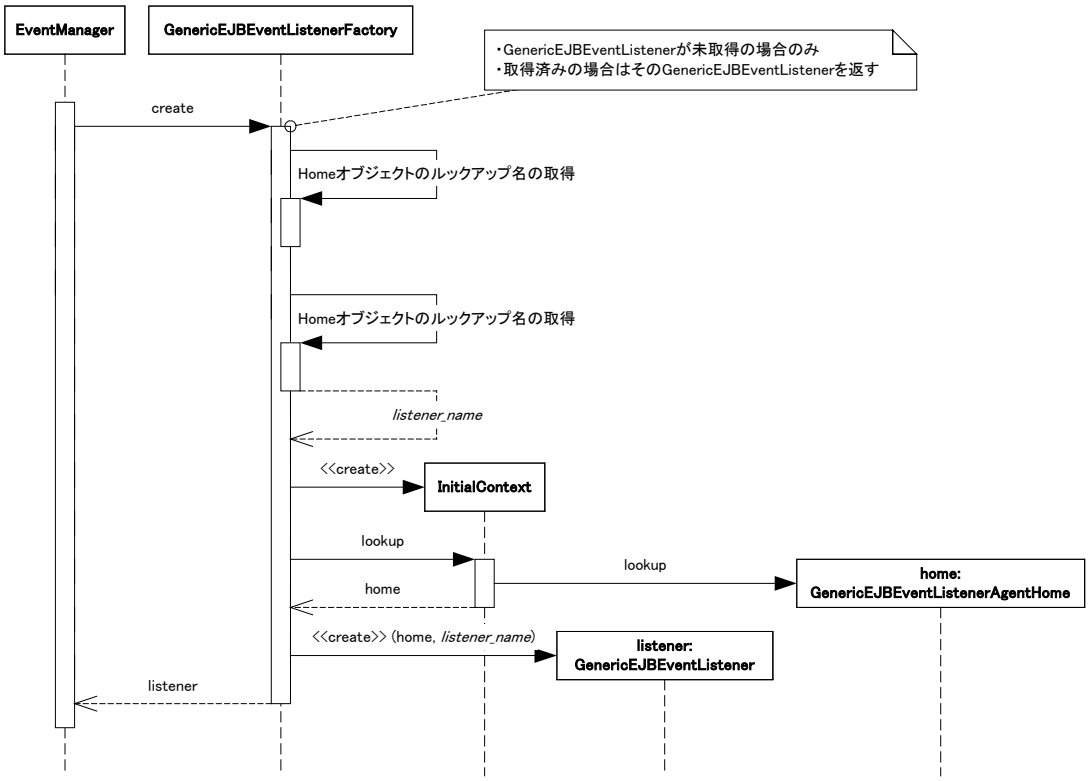


Figure 4-11 GenericEJBEventListener Generation

Initialization parameter shown in [Table 4-4 Initialization Parameter of GenericEJBEventListenerFactory] should be obtained by getEventListenerFactoryParams of EventPropertyHandler.

Table 4-4 Initialization Parameter of GenericEJBEventListenerFactory

Parameter Name	Parameter Contents
home	Name when look up the Home object of EJB that is required in GenericEJBEventListener to be generated.
listener	Complete class name that includes the package of StandardEventListener to be generated.

4.3.2.2 Original EventListenerFactory

When the developer creates EventListenerFactory originally, following requirements shoud be met.

- jp.co.intra_mart.framework.base.event.EventListenerFactory interface is implemented.
- public default constructor (constructor with no argument) is defined.
- create method returns appropriate class instance that implements EventListener interface.

If EventListenerFactory that is originally developed meets above conditions, event framework of IM-JavaEE Framework treats the EventListenerFactory as shown in [Figure 4-6 EventListenerFactory Generation (with cache, already generated))] or [Figure 4-7 EventListenerFactory Generation (no cache or not generated].

4.3.3 EventListener

The main role of EventListeneris to execute the business logic. Business logic is performed in execute method. In the execute method, it is preferrable to execute EventTrigger (refer to [4.3.4 EventTrigger]) before executing the business logic. Process overview of EventListener is shown in [Figure 4-12 EventListener Overview].

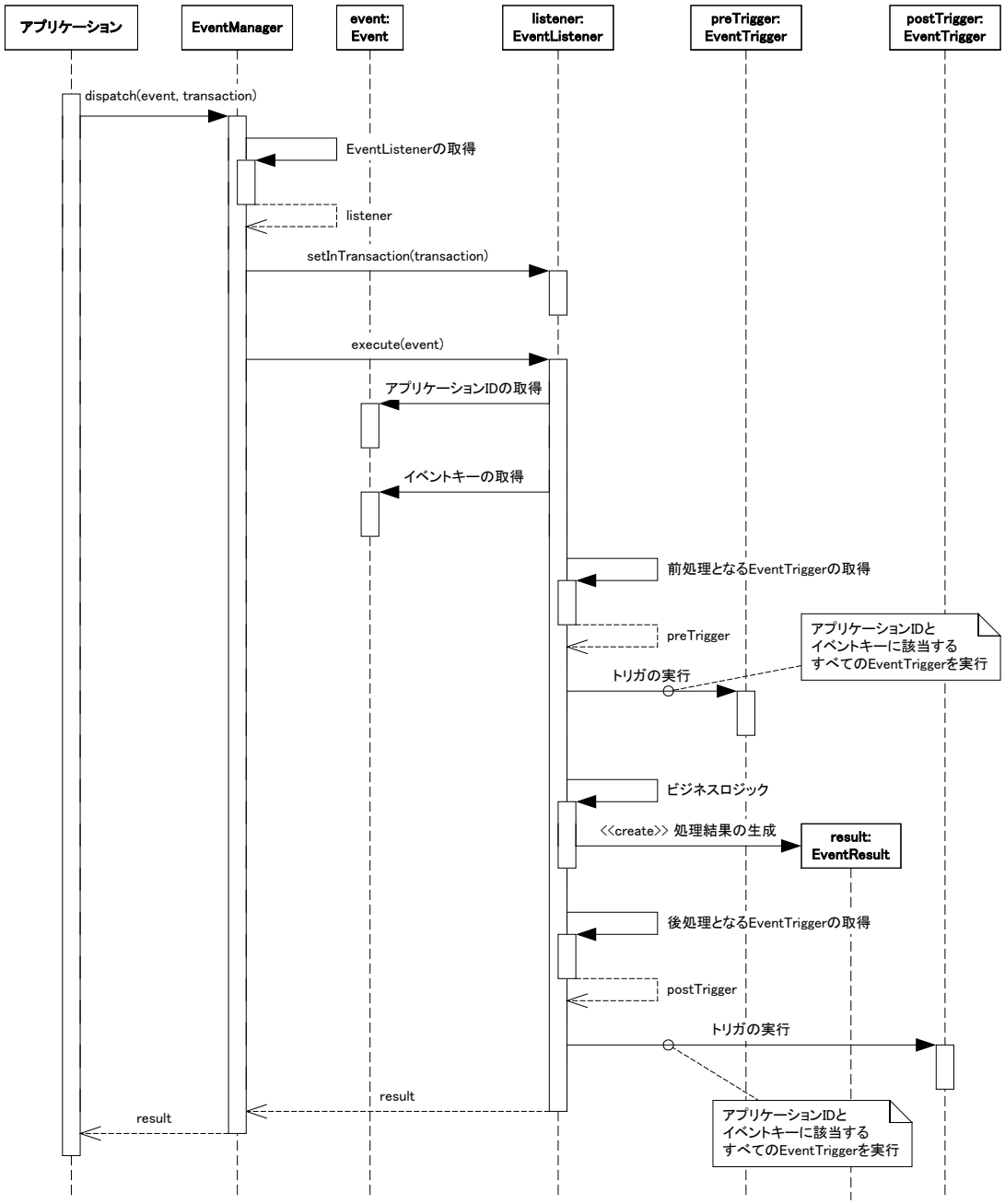


Figure 4-12 EventListener Overview

4.3.3.1 Abstract EventListener provided as Standard

EventListener can also be developed by the developer him/herself, but several popular ones are provided as abstract classes from the beginning. Followings are the EventListener provided from the beginning.

- StandardEventListener
- GenericEventListener
- StandardEJBEventListener
- GenericEJBEventListener

All of these are belong to the package jp.co.intra_mart.framework.base.event.

4.3.3.1.1 StandardEventListener

StandardEventListener is the EventListener that has the most basic function. The developer can implement business

logic by extending this class. This class has following features.

- No need to consider about EventTrigger execution.
- You can implement business logic just by overriding fire method.
- Transaction management is done automatically. Please refer to [4.5.1 StandardEventListener].
- If you combine with GenericEJBEventListener, it can be executed in remote environment through EJB. Please refer to [4.3.3.1.4 GenericEJBEventListener].

Operation of StandardEventListener is shown in [Figure 4-13 StandardEventListener Operation].

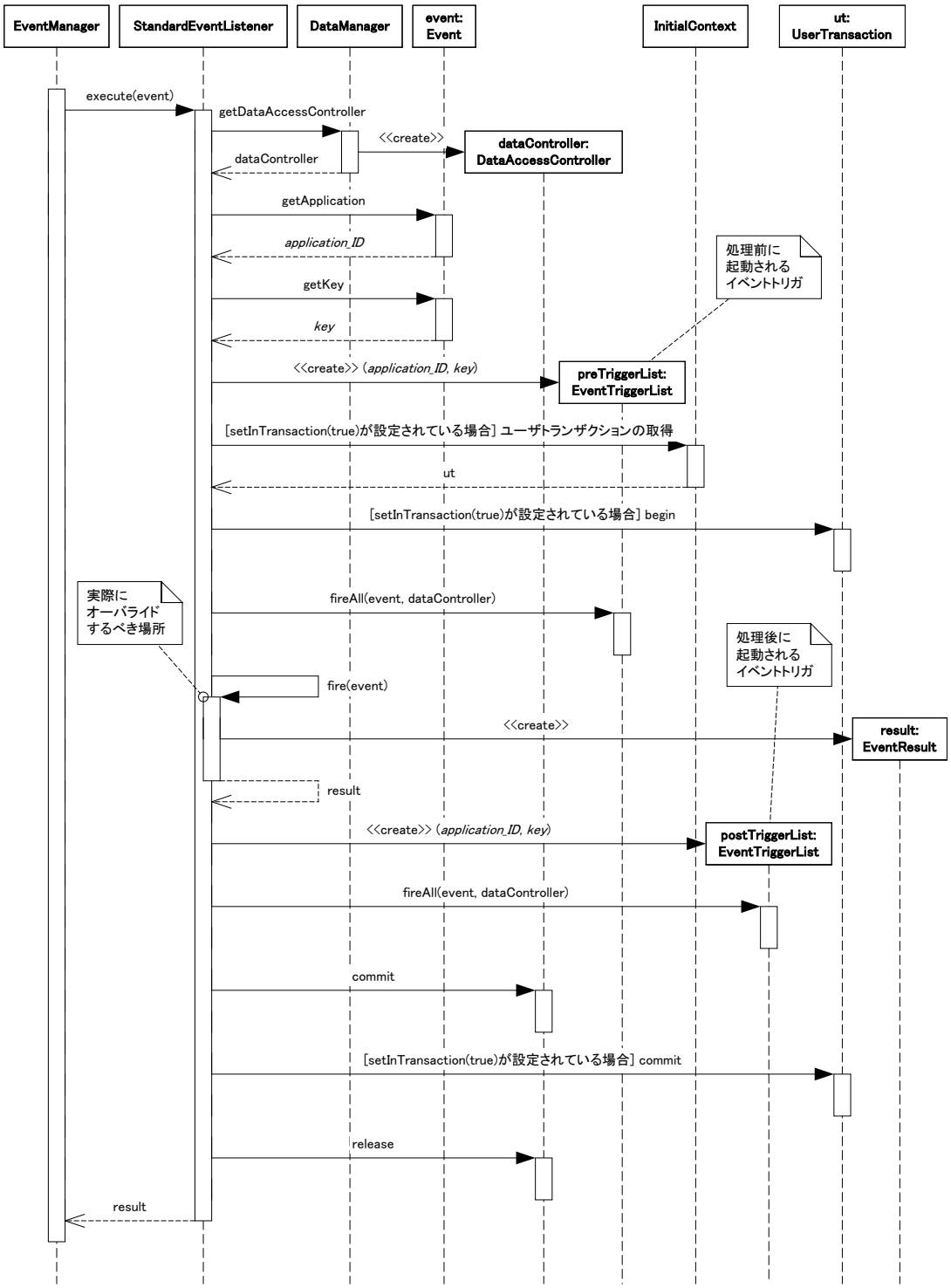


Figure 4-13 StandardEventListener Operation

4.3.3.1.2 GenericEventListener

GenericEventListener just laps StandardEventListener, and execute method just calls the execute method of StandardEventListener.

Operation of StandardEventListener is shown in [Figure 4-14 GenericEventListener Operation].

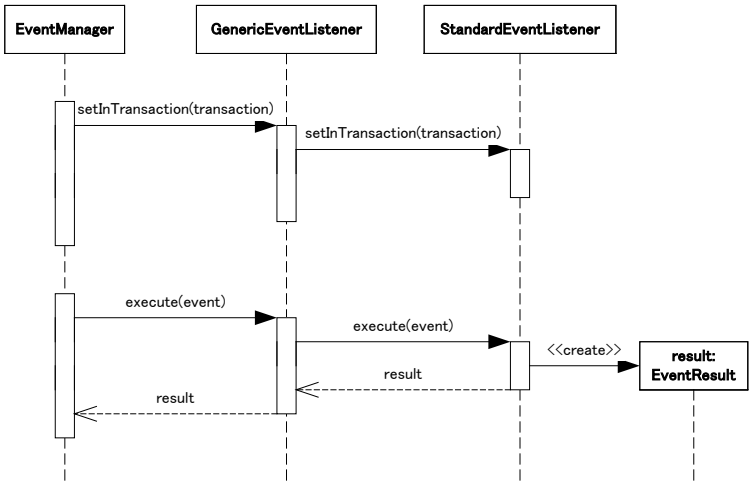


Figure 4-14 GenericEventListener Operation

4.3.3.1.3 StandardEJBEventListener

StandardEJBEventListener is the EventListener that uses EJB. Structure and the sequence figure are shown in [Figure 4-15 StandardEJBEventListener Structure], [Figure 4-16 StandardEJBEventListener Operation (client)] and [Figure 4-17 StandardEJBEventListener Operation (EJB server)].

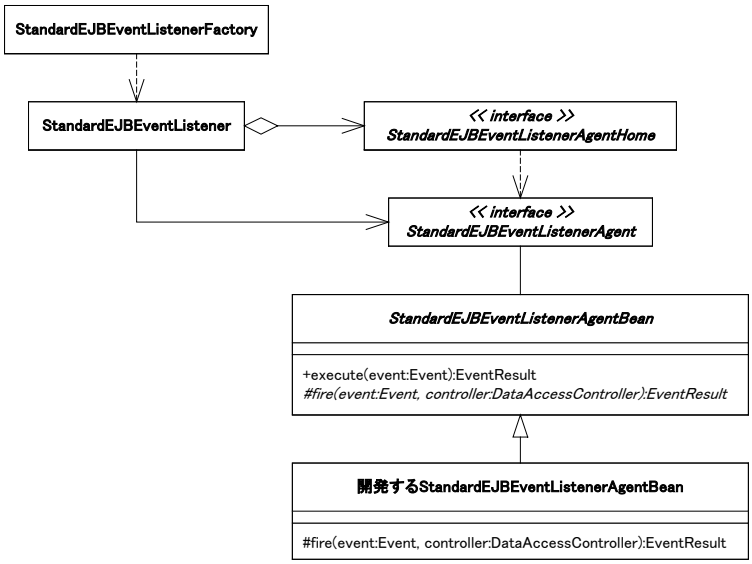


Figure 4-15 StandardEJBEventListener Structure

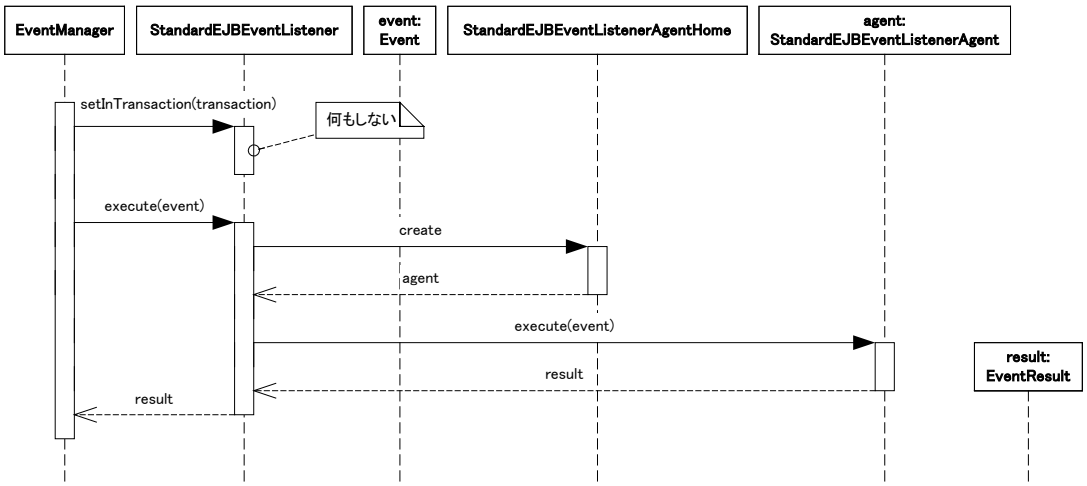


Figure 4-16 StandardEJBEventListener Operation (client)

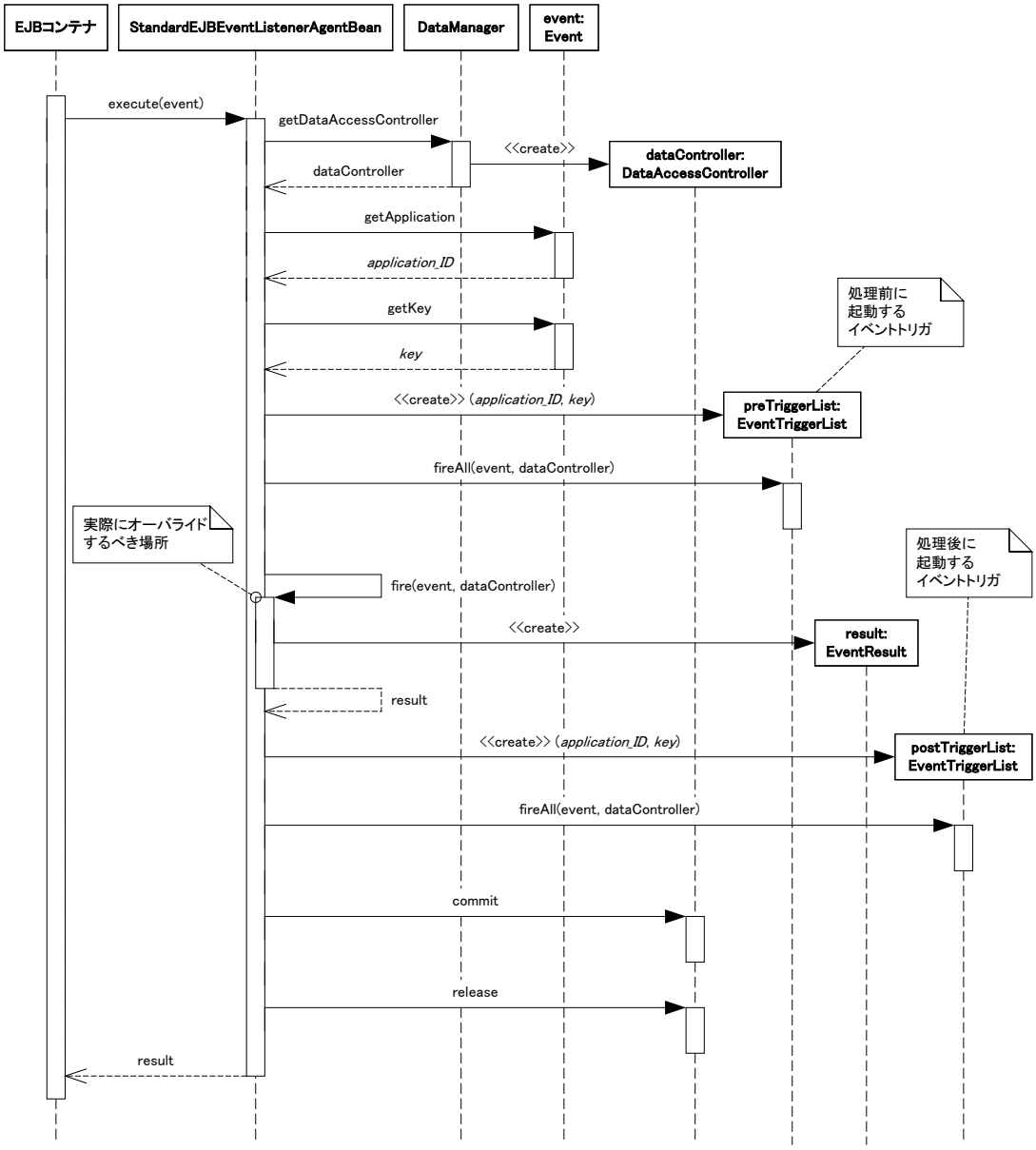


Figure 4-17 StandardEJBEventListener Operation (EJB server)

4.3.3.1.4 GenericEJBEventListener

GenericEJBEventListener is the EventListener that executes StandardEventListener in remote environment through EJB. Structure and the sequence figure is shown in [Figure 4-18 GenericEJBEventListener Structure]. [Figure 4-19 GenericEJBEventListener Operation (client)] and [Figure 4-20 GenericEJBEventListener Operation (EJB server))].

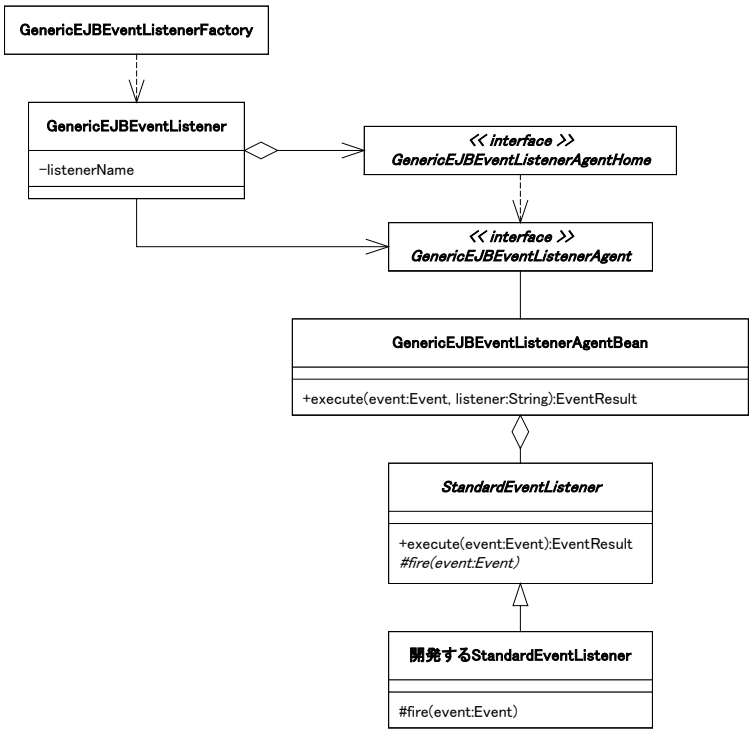


Figure 4-18 GenericEJBEventListener Structure

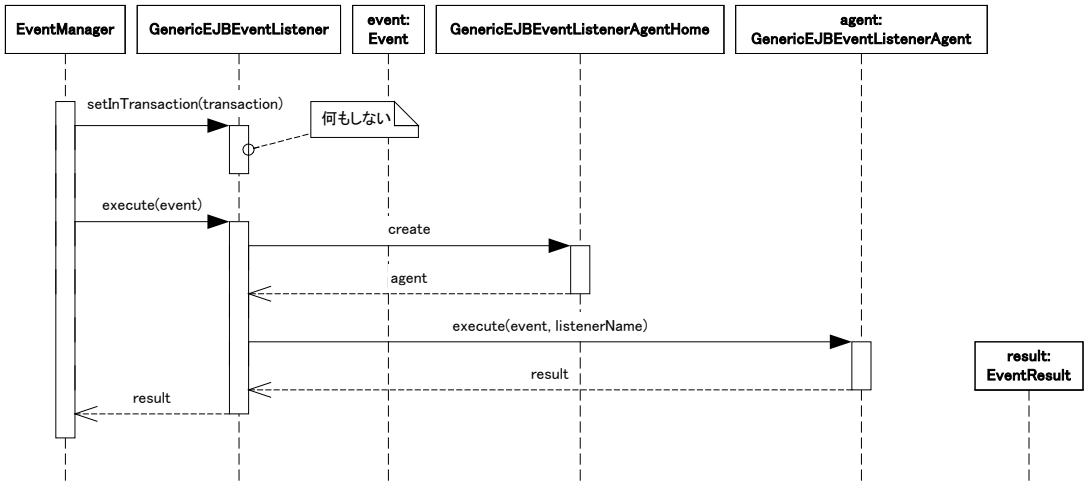


Figure 4-19 GenericEJBEventListener Operation (client)

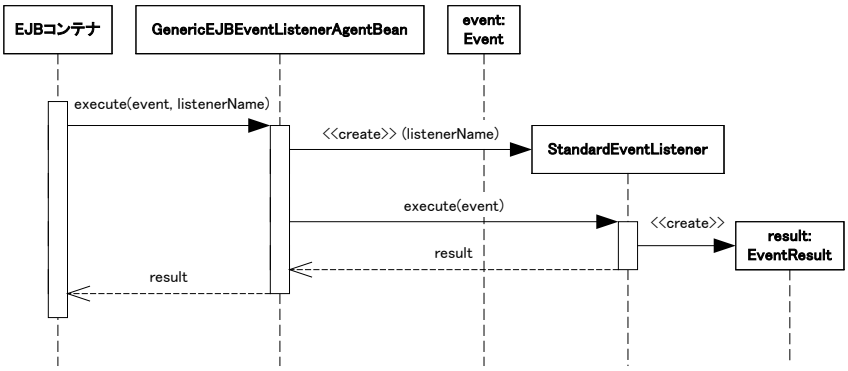


Figure 4-20 GenericEJBEventListener Operation (EJB server)

4.3.3.2 Original EventListener

EventListener processes received event. EventListener should meet the following requirements.

- Interface jp.co.intra_mart.framework.base.event.EventListener is implemented.
- In setInTransaction method, if the argument value is true, preparation in case transaction is already started is implemented and the preparation in case transaction has not yet started is implemented if it is false.
- In execute method (if it exists), every EventTrigger that corresponds to application ID and event key should be executed before and after the actual process. In this case, EventTrigger list will be obtained as below.
  - For application ID and event key, the value to be obtained by getApplication method and getKey method of the event which was passed as the argument.
  - EventTrigger list that is performed before actual process is obtained by getEventTriggerInfos method of EventPropertyHandler.
  - EventTrigger list that is performed after actual process is obtained by getPostEventTriggerInfos method of EventPropertyHandler.
  - Process related to the transaction (start transaction, commit, roll back etc.) shoud be performed if necessary. In this case, transaction process that reflects the preparation performed in setInTransection method is preferred.

4.3.4 EventTrigger

EventTrigger is the process contents that are performed before and after the execution of EventListener. EventTrigger is different from EventListener on following points.

- It can be set more than one for the same application ID and event key.
- It does not return the process results (EventResult).

EventTrigger should meet the following conditions.

- Interface jp.co.intra_mart.framework.base.event.EventTrigger is implemented.
- fire method is stated to perform appropriate process.

If you use jp.co.intra_mart.framework.base.event.EventTriggerAdapter abstract class that implements EventTrigger interface, you will be able to use getDAO method in fire (Event) method as StandardEventListener. DAO to be obtained by this method is the same with the return value of getDAO method of DataAccessController passed in fire(Event, DataAccessController) method.

4.3.5 EventResult

EventResult is the process result of EventListener. EventResult should meet the following conditions.

- jp.co.intra_mart.framework.base.event.EventResult interface is implemented.
- It can be serialized. If EJB is used, EventResult will be passed from remote environment to the original environment. At this point, if it cannot be serialized, EventResult contents may be corrupted.

4.4 Property related to the Event

In event framework of IM-JavaEE Framework, various properties can be set at outside. Event property will be obtained from the class that implements jp.co.intra_mart.framework.base.event.EventPropertyHandler interface. Multiple implementation class that implements this interface is provided as standard in IM-JavaEE Framework (refer to [Figure 4-21 EventPropertyHandler]). How to set event property is not specially defined in IM-JavaEE Framework, it depends on the class that implements the interface described above.

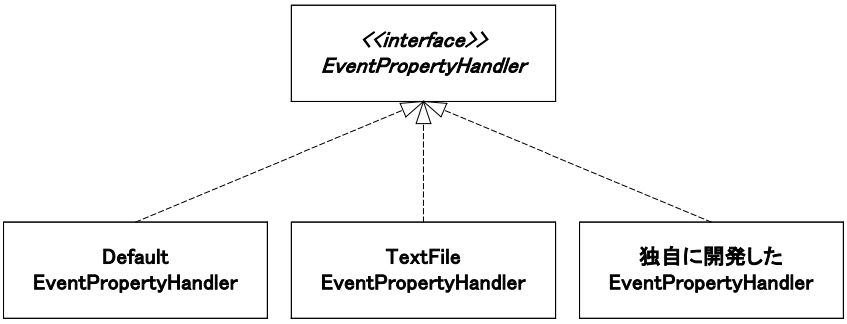


Figure 4-21 EventPropertyHandler

4.4.1 Obtaining Property related to the Event

Property related to the Event will be obtained from EventPropertyHandler. EventPropertyHandler can be obtained by getEventPropertyHandler of jp.co.intra_mart.framework.base.event.EventManager. EventPropertyHandler must be obtained through this method and the developer cannot generate implementation class of this EventPropertyHandler explicitly (generation by new or instance generation using newInstance method of java.lang.Class or reflection) by him/herself.

Procedures for obtaining EventPropertyHandler and property are shown in [Figure 4-22 Obtaining EventPropertyHandler].

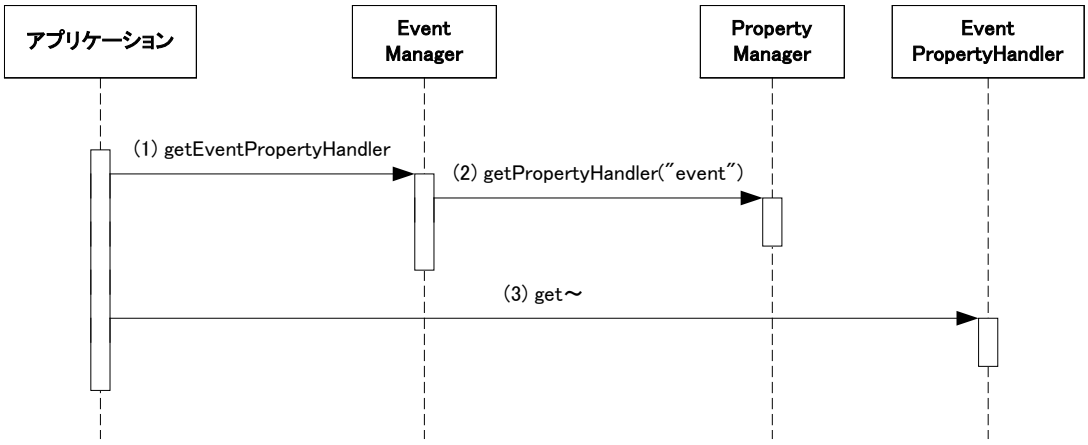


Figure 4-22 Obtaining EventPropertyHandler

1. Obtain EventPropertyHandler from EventManager.
2. Inside the EventManager, EventPropertyHandler is obtained from PropertyManager and returns to the application. This part is done in the EventManager so the developer doesn't have to consider about this.
3. Obtain each property by using EventPropertyHandler.

4.4.2 EventPropertyHandler provided as Standard

In IM-JavaEE Framework, several classes that implement jp.co.intra_mart.framework.base.event.EventPropertyHandler are provided. Setting method or its features are depending on each, so the operator should switch these accordingly.

4.4.2.1 DefaultEventPropertyHandler

It is provided as jp.co.intra_mart.framework.base.event.DefaultEventPropertyHandler.

Property setting is done by resource file. Resource file contents are set in [property name=property value] format. Characters can be used are according to java.util.ResourceBundle. This resource file should be placed in the class path that can be obtained from application to be used. Please refer to API list for the detail of file name of resource file or property name to be set.

4.4.2.2 TextFileEventPropertyHandler

It is provided as jp.co.intra_mart.framework.base.event.TextFileEventPropertyHandler.

Resource file that has the same format with DefaultEventPropertyHandler is used but following points are different.

- No need to pass a class path.
- If the location accessed from the application, it can be placed at any location in the file system.
- You can reload the resource file without stopping the application according to the setting.

Please refer to API list for the detail of file name of resource file or the property to be set.

4.4.2.3 XmlEventPropertyHandler

It is provided as jp.co.intra_mart.framework.base.event.XmlEventPropertyHandler.

Property setting is done in XML format. It should be placed in the class path which can be obtained by the application. Unique ID and Java package path are recognized as application. For example, if application ID is "foo.bar.example", it is placed as "foo/bar/event-config-example.xml" as class path.

XmlEventPropertyHandler supports dynamic loading.

Please refer to API list for details.

4.4.3 Original EventPropertyHandler

If the developer creates EventPropertyHandler originally, follwing requirements should be met.

- jp.co.intra_mart.framework.base.event.EventPropertyHandler interface is implemented.
- public default constructor (with no argument) is defined.
- Appropriate value must be returned to all methods (refer to [4.4.4 Property Contents]).
- If isDynamic() method returns false, the value of method that obtains property will not change unless application server is restarted.

4.4.4 Property Contents

Setting method of property related to the Event is depending on the type of EventPropertyHandler that is to be used when operation, but the concept is same.

Contents of property related to the event are described below.

4.4.4.1 Common

4.4.4.1.1 Dynamic Loading

It can be obtained by isDynamic() method.

If the return value of this method is true, each property obtaining method (get~method) defined in this interface should load setting information initially everytime as the implementation. If it is false, each property obtaining method can cache the value to be obtained internally considering the performance.

4.4.4.2 Application Individual

4.4.4.2.1 Event

It can be obtained by getEventName(String application, String key) method.

It sets the complete class name of the event that correnspponds to the application ID and the event key. If it is not set, it returns null. The class to be specified here needs to extend jp.co.intra_mart.framework.base.event.Event class.

4.4.4.2.2 EventListenerFactory

It can be obtained by getEventListenerFactoryName(String application, String key) method.

It sets the complete class name of EventListenerFactory that corresponds to the application ID and the event key. If it is not sest, it throws jp.co.intra_mart.framework.base.event.EventPropertyException. the class to be specified here needs to implement jp.co.intra_mart.framework.base.event.EventListenerFactory interface.

4.4.4.2.3 **Initial Parameter of EventListenerFactory**

It can be obtained by getEventListenerFactoryParams(String application, String key) method.

It sets the initial parameter of EventListenerFactory that corresponds to the application ID and the event key. If it is not set, it returns the array with the size 0.

4.4.4.2.4 **EventTrigger Information (start before the event process)**

It can be obtained by getEventTriggerInfos(String application, String key) method.

It sets EventTrigger information that corresponds to the application ID and the event key. EventTrigger to be obtained here is started before the event process. The information to be returned should meet the following conditions.

- Collections of jp.co.intra_mart.framework.base.event.EventTriggerInfo.
- In the iterator method of collections to be returned, it can be obtained in the sequence that EventTriggerInfo is set.
- EventTriggerInfo contains following information.
  - Unique sequence in the combination of application ID and event key (can be obtained by getNumber method)
  - The complete class name that includes the package name of EventTrigger (can be obtained by getName method).

If it is not set, Collection with null contents will be returned.

4.4.4.2.5 **EventTrigger Information (start after the event process)**

It can be obtained by getPostEventTriggerInfos(String application, String key) method.

It sets the EventTrigger Information that corresponds to the application ID and the event key. EventTrigger to be obtained here is started after the event process. The information to be returned should meet the following conditions.

- Collections of jp.co.intra_mart.framework.base.event.EventTriggerInfo.
- In the iterator method of collections to be returned, it can be obtained in the sequence that EventTriggerInfo is set.
- EventTriggerInfo contains following information.
  - Unique sequence in the combination of application ID and event key (can be obtaied by getNumber method).
  - The complete class name that includes the package name of EventTrigger (can be obtained by getName method).

If it is not set, Collection with null contents will be returned.

4.5 Transaction

StandardEventListener and StandardEJBEventListener that are provided as standard in event framework of IM-JavaEE Framework have transaction function. Basically, these start/end transaction with the unit of execute(Event event) method. If you manufacture the component of EventListener by using these, fire method should be implemented as follows :

- If you want to commit the transaction, end it by the normal process.
- If you want to roll back the transaction, throw the exceptions.

4.5.1 StandardEventListener

How the transaction is managed in the StandardEventListener is shown in [Figure 4-23 Transaction of StandardEventListener (started inside)].

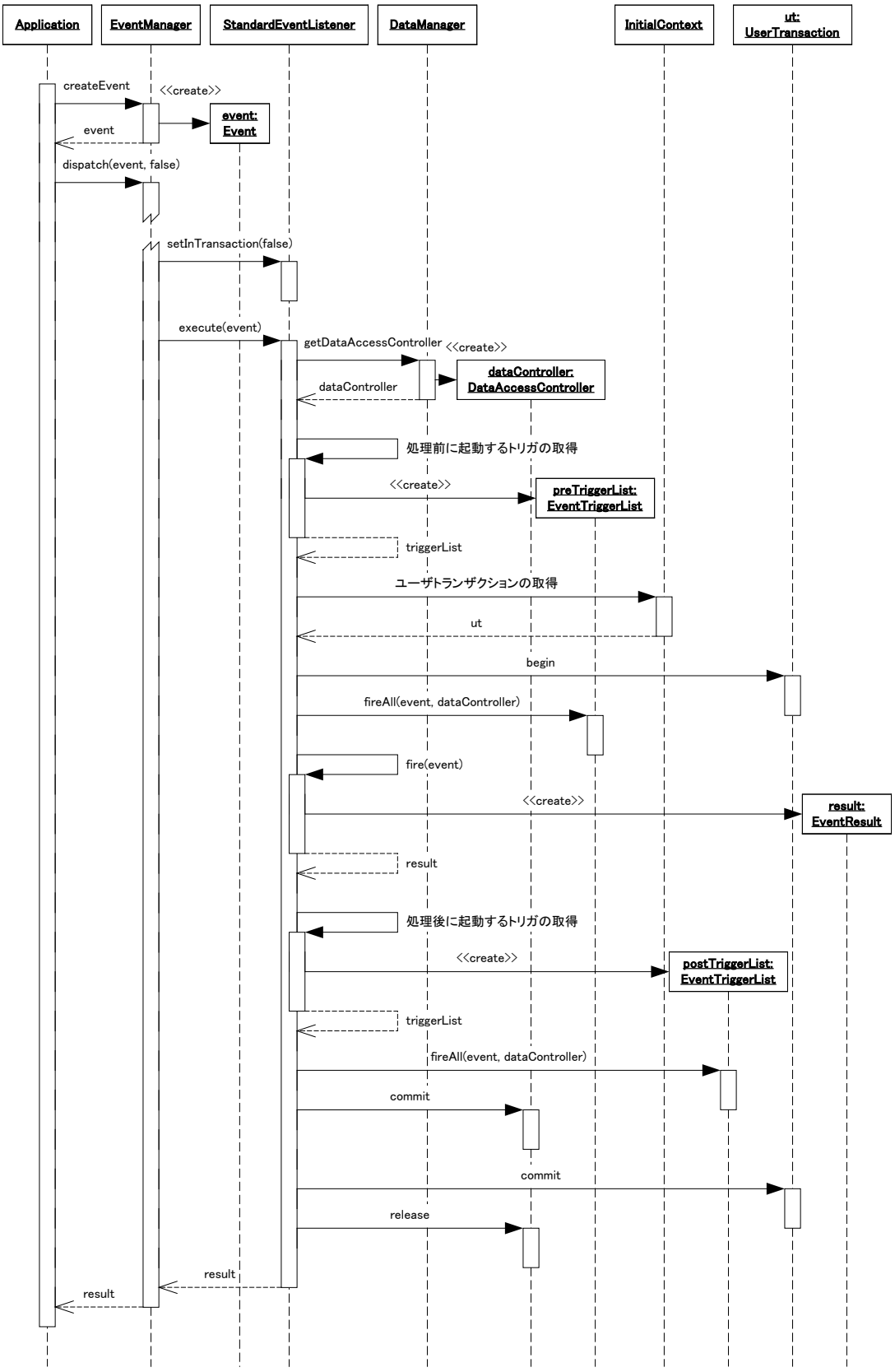


Figure 4-23 Transaction of StandardEventListener (started inside)

How the transaction is managed outside the StandardEventListener is shown in [Figure 4-24 Transaction of StandardEventListener (started outside)].

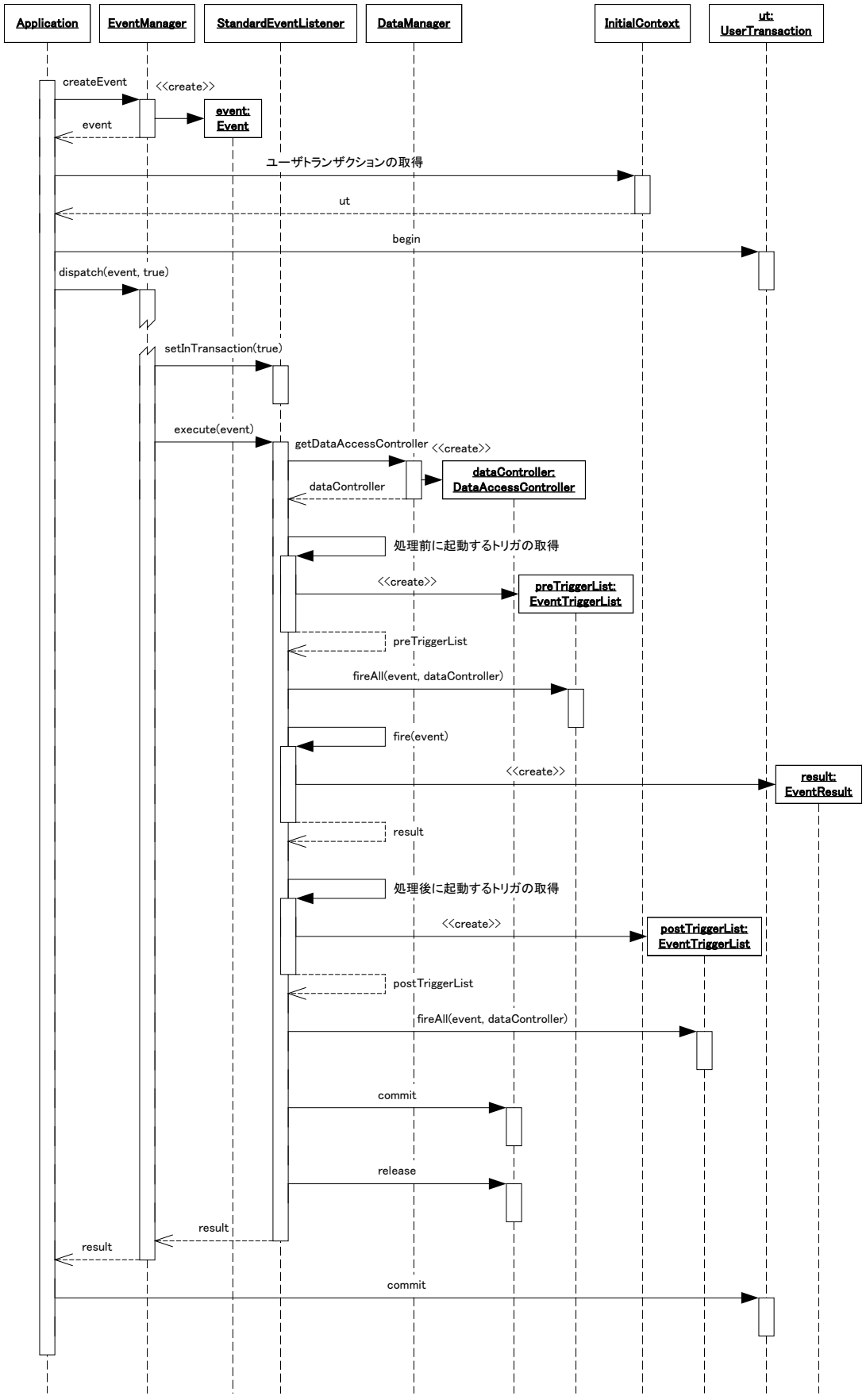


Figure 4-24 Transaction of StandardEventListener (started outside)

If you take a look at [Figure 4-23 Transaction of StandardEventListener (started inside)] or [Figure 4-24 Transaction of StandardEventListener (started outside)], you can see that commit is done in 2 locations, DataAccessController and UserTransaction (refer to [5.5 Transaction] for the transaction management done in DataAccessController). This is because both of them will not be the same transaction since the management system of DataAccessController and UserTransaction is different. Thus, if you handle the database by data framework, it is recommended to access by DataSource that is handled in UserTransaction, not by DataAccessController.

StandardEventListener can call other EventListener from inside the fire method in a nested way. In this case, dispatchEvent method of StandardEventListener is used. This is depicted in [Figure 4-25 StandardEventListener calling Other EventListener].

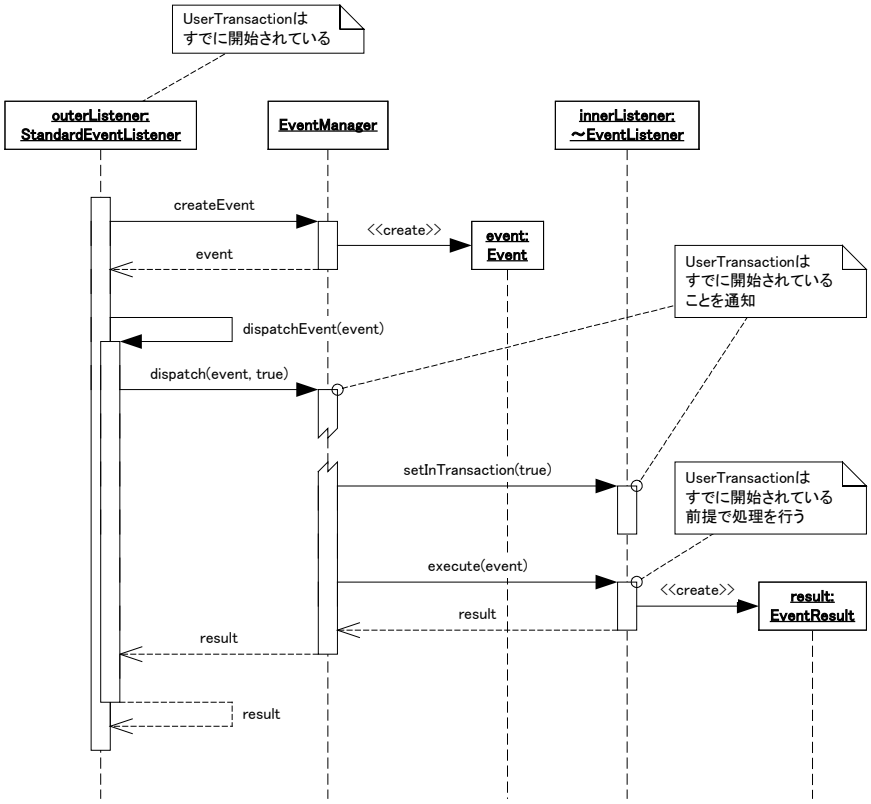


Figure 4-25 StandardEventListener calling Other EventListener

When you call StandardEventListener or StandardEJBEventListener from StandardEventListener, you should be awared of one thing. If you take a look at [Figure 4-23 Transaction of StandardEventListener (started inside)],[Figure 4-24 Transaction of StandardEventListener (started outside)], and [Figure 4-25 StandardEventListener calling Other EventListener] in connected format, you can see that DataAccessController(refer to [5.3.2 DataAccessController] for the detail of DataAccessController) is generated in both external StandardEventListener and internal StandardEventListener. This means that the transaction managed in DataAccessController is generated separatedly. In other words, if StandardEventListener is nested, it may not be gathered in one transaction if the data connector that is managed by DataAccessController (such as JDBCConnector. Refer to [5.3.3 DataConnector] for the detail of deta connector) is used. In order to avoid this issue, it is recommended to access by DataSource that is handled in UserTransaction, not by DataAccessController, if the database is handled by data framwrork.

4.5.2 GenericEventListener

Transaction is not specially managed in GenericEventListener. How to manage the transaction is up to the inclusive StandardEventListener.

4.5.3 StandardEJBEventListener

How the transaction is managed in StandardEJBEventListener is shown in [Figure 4-26 Transaction of StandardEJBEventListener].

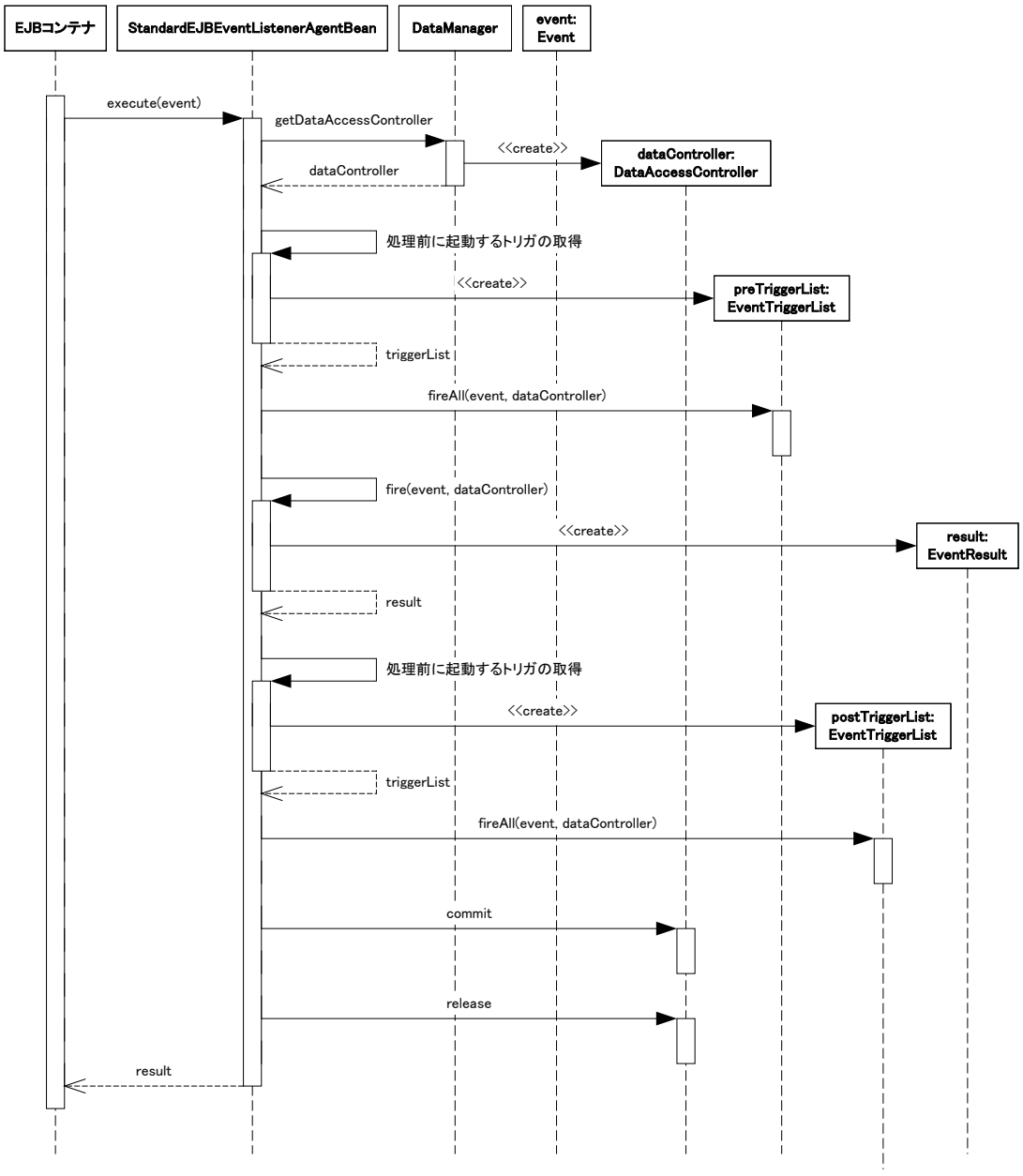


Figure 4-26 Transaction of StandardEJBEventListener

If you take a look at [Figure 4-26 Transaction of StandardEJBEventListener], you can see that starting/ending of UserTransaction is not performed. Thus, if you use StandardEJBEventListener, you should control the transaction by CMT (container-managed transaction). Access for the databas is recommended to be performed by DataSource due to the same reason described in [4.5.1 StandardEventListener].

StandardEJBEventListener can call other EventListener from inside the fire method in a nested way. In this case, dispatchEvent of StandardEJBEventListener is used. This is depicted in [Figure 4-27 StandardEJBEventListener calling Other EventListener].

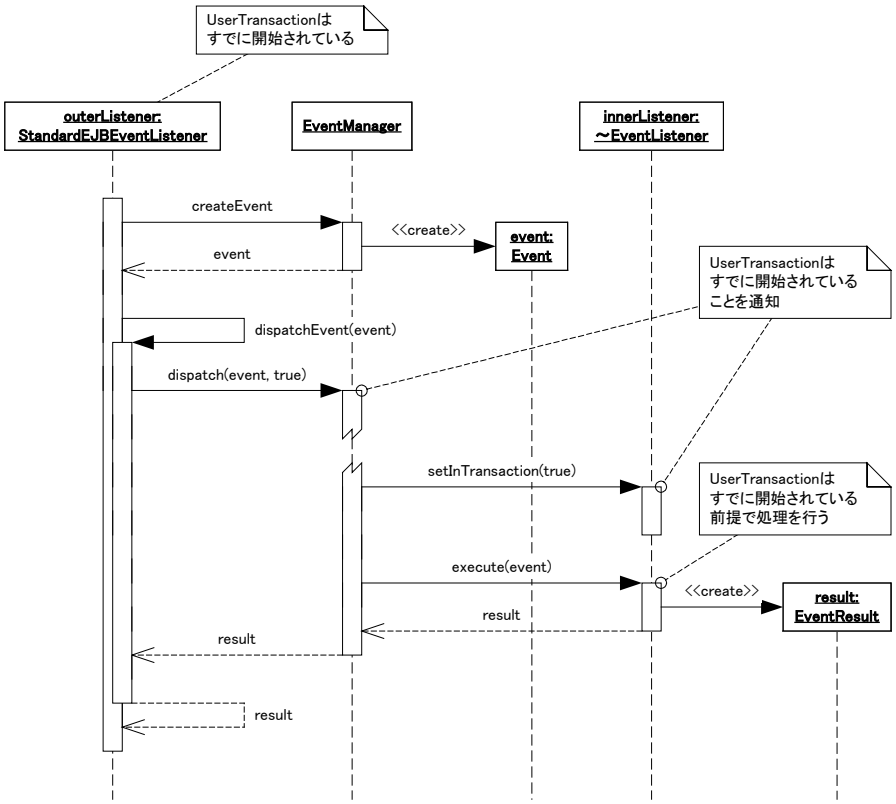


Figure 4-27 StandardEJBEventListener calling Other EventListener

When you call StandardEventListener or StandardEJBEventListener from StandardEJBEventListener, you shouuld be aware of the points that are described in [4.5.1 StandardEventListener] for the data framework (refer to [5 Data Framework] for the detail). In order to avoid this issue, it is recommended to access by DataSource that is handled in UserTransaction, not by DataAccessController, if the database is handled by data framwrork.

4.5.4 GenericEJBEventListener

How the transaction is managed in GenericEJBEventListener is shown in [Figure 4-28 Transaction of GenericEJBEventListener].

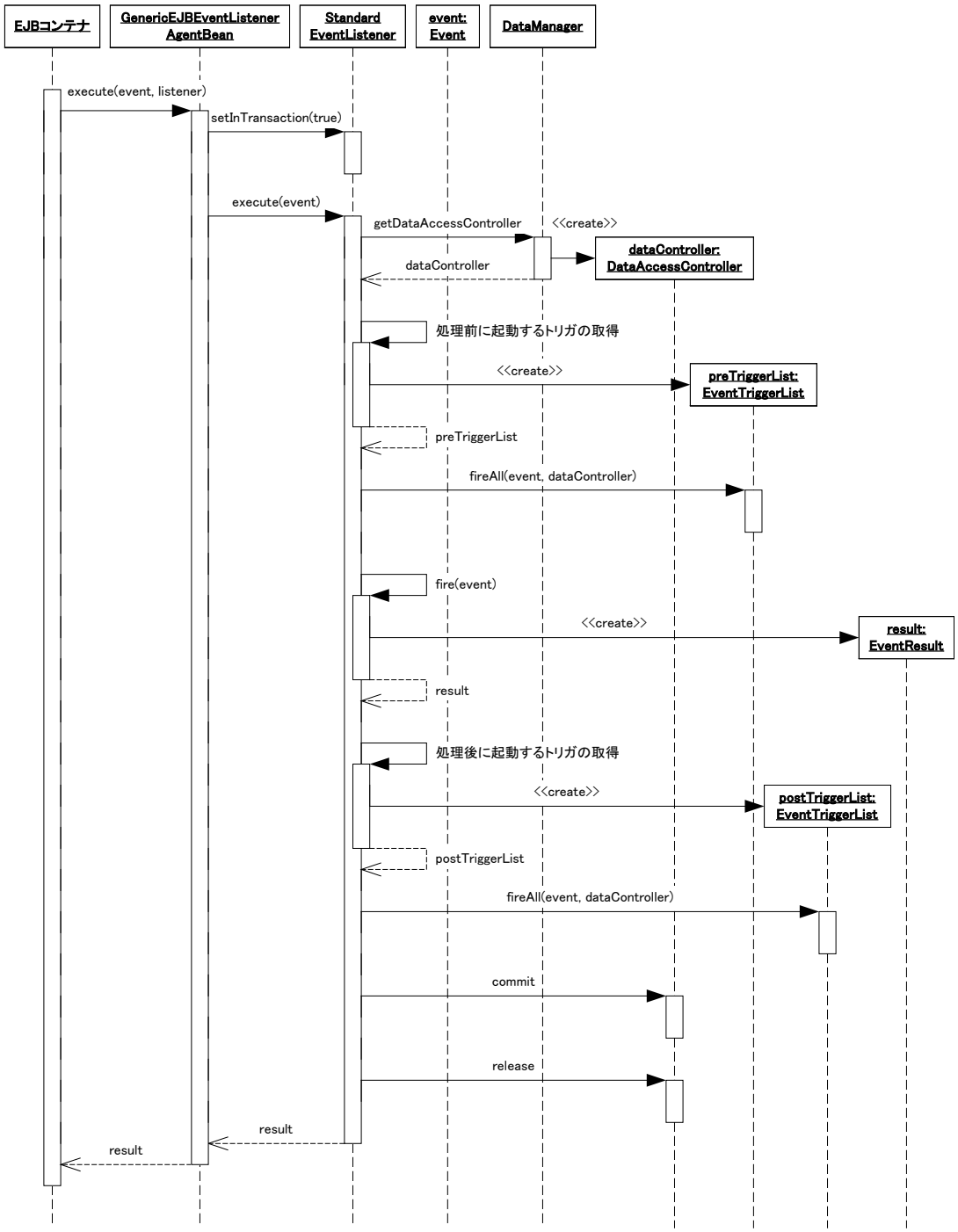


Figure 4-28 Transaction of GenericEJBEventListener

If you take a look at [Figure 4-28 Transaction of GenericEJBEventListener], you can see that starting/ending of UserTransaction is not performed. Thus, if you use GenericEJBEventListener, you should control the transaction by CMT (container-managed transaction). Access for the databas is recommended to be performed by DataSource due to the same reason described in [4.5.1 StandardEventListener].

GenericEJBEventListener can call other EventListener from inside the fire method in a nested way. In this case, dispatchEvent method of GenericEJBEventListener is used. This is depicted in [Figure 4-29 GenericEJBEventListener calling Other EventListener].

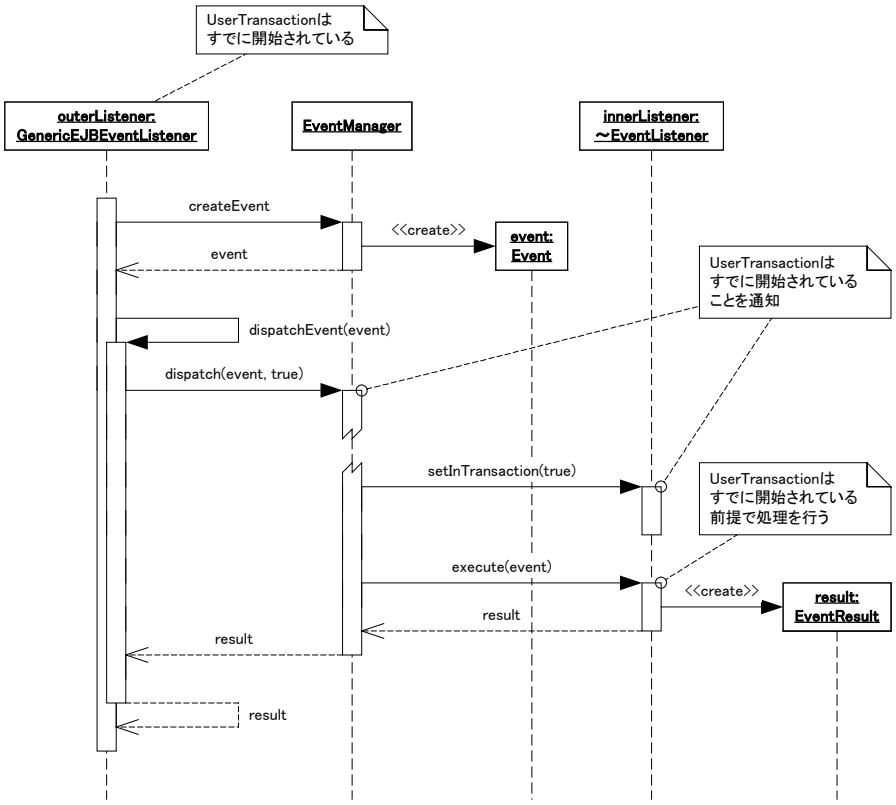


Figure 4-29 GenericEJBEventListener calling Other EventListener

When you call StandardEventListener or StandardEJBEventListener from StandardEJBEventListener, you should be aware of the points that are described in [4.5.1 StandardEventListener] for the data framework (refer to [5 Data Framework] for the detail). In order to avoid this issue, it is recommended to access by DataSource that is handled in UserTransaction, not by DataAccessController, if the database is handled by data framwrork.

5 Data Framework

5.1 Overview

Data persistence or linkage with other system can be the most important element in a business logic. Some procedures are required in order to connect with database or other systems, but most of them are in a fixed format. Connection destination may be changed, in this case, information of connection destination should be corrected but the procedures of connection is the same in most of cases. (if you change the type of database only, it is rare to correct the connection obtaining or SQL issuance).

In data framework, these fixed parts are made into framework.

5.2 Structure

5.2.1 Structure Elements

Data framework is structured by the followings.

- DAO
- DataConnector
- DataAccessController
- Resource

The relationship between these are shown in [Figure 5-1 Data Framework Structure].

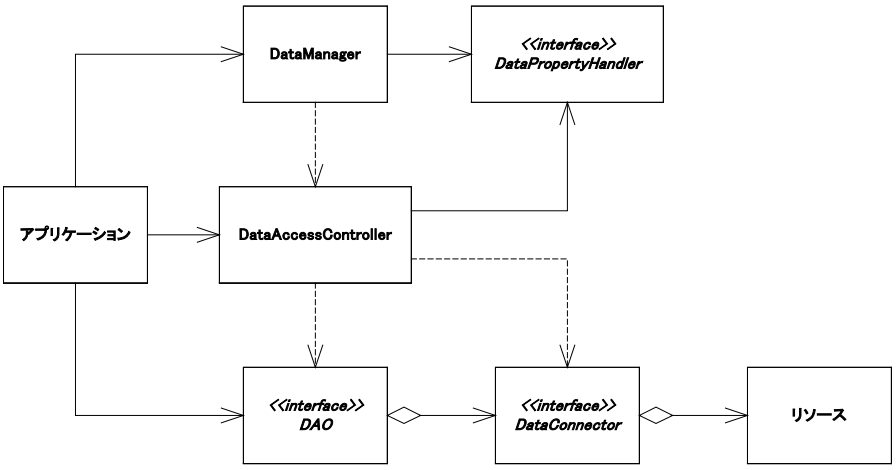


Figure 5-1 Data Framework Structure

5.2.1.1 DAO

DAO (Data Access Object) supports the access for resources in EIS layer (database or backbone system etc.). The developer of business logic should not operate database or files directly, better to access the resource through DAO.

DAO should know the types of resources in connection destination but does not need to know the connection destination. DAO does not access to the resource directly, but through the DataConnector.

This component should be implemented by the depeloper who has a good knowledge of resources.

5.2.1.2 DataConnector

DataConnector has role to connect DAO and the resource in EIS layer. Some of these components are provided as standard by intra-mart but it can be originally implemented by the developer.

DataConnector should be specialized for the resource. DataConnector needs to know the connection destination explicitly.

5.2.1.3 DataAccessController

DataAccessController does obtaining, generation or end process, and management of simple transaction of DAO.

The relationship of DAO and DataConnector is defined by the property. DataAccessController generates the DAO that appropriate DataConnector is set and provides according to this property setting contents.

Since this component is provided by intra-mart, the developer only should know how to use.

5.2.1.4 Resource

Resource is the current status of the system in EIS layer (database or backbone system etc.) or its connection method. Such as Storage server, database, or backbone system in intra-mart are listed as examples of the resource.

Resource itself is not provided in IM-JavaEE Framework.

5.2.2 Data Access

How to data access to the resource in EIS layer by using data framework of IM-JavaEE Framework is shown in [Figure 5-2 Data Access Overview].

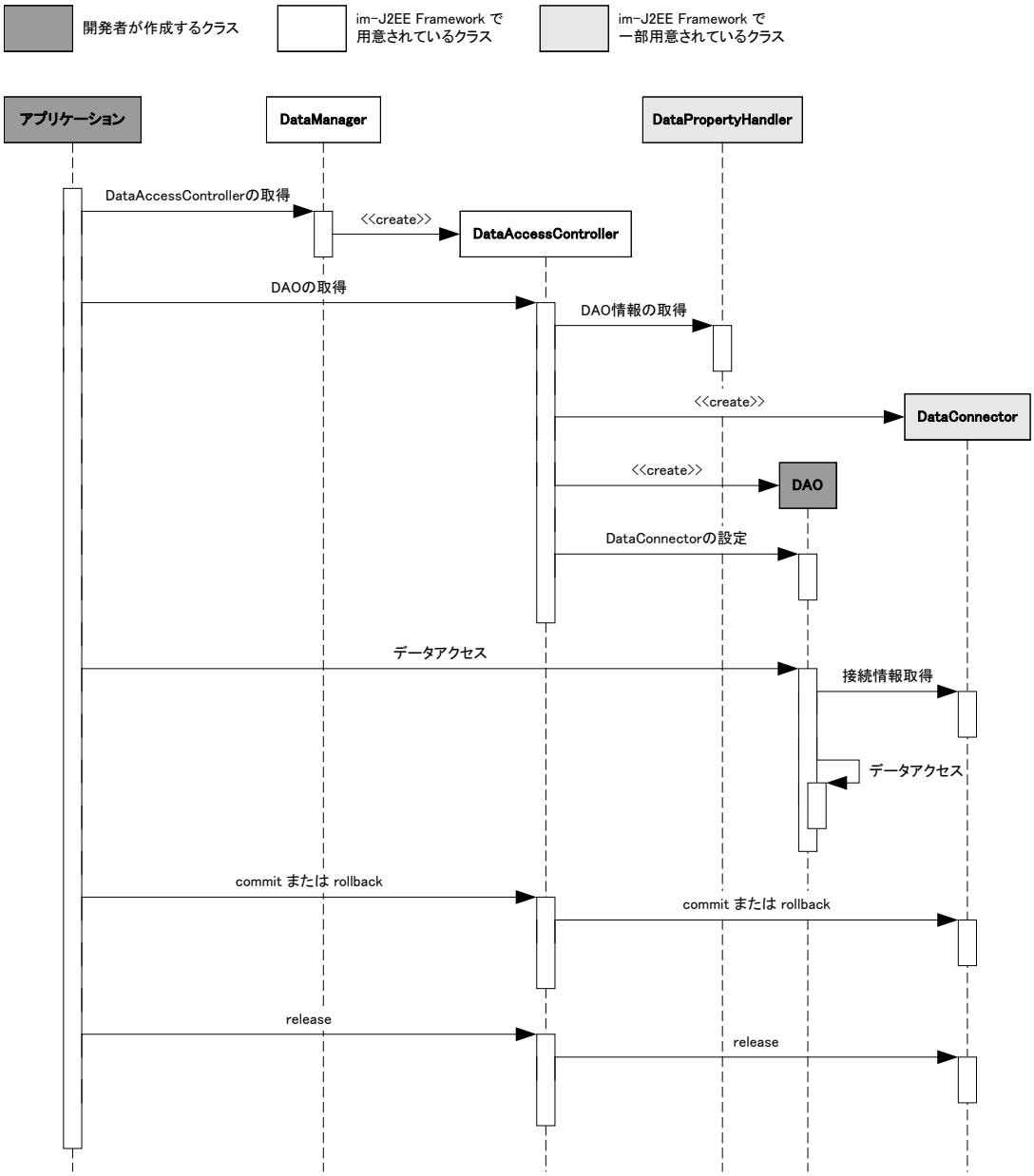


Figure 5-2 Data Access Overview

1. It calls getDataAccessController method of DataManager and obtains DataAccessController.
2. It calls getDAO method of DataAccessController and obtains DAO that accesses to the data.
3. It calls DAO method and accesses to the data.
4. It calls commit or rollback method of DataAccessController and does commit or rollback of the process contents.
5. It calls release method of DataAccessController and releases the connection for the resource.

5.3 Structure Element Details

5.3.1 DAO

DAO（Data Access Object） has a role to access for the resource in EIS layer. DAO is mainly used from business logic, so the interface should be released to the developer but he/she does not need to know the business logic detail.

DAO is dedicated for accessing the resource only, and the connection method does not have to be considered.

Connecting to the resource is depending on DataConnector.

DAO is obtained by using getDAO method of DataAccessController. How the DAO is obtained is shown in [Figure 5-3 Obtaining DAO].

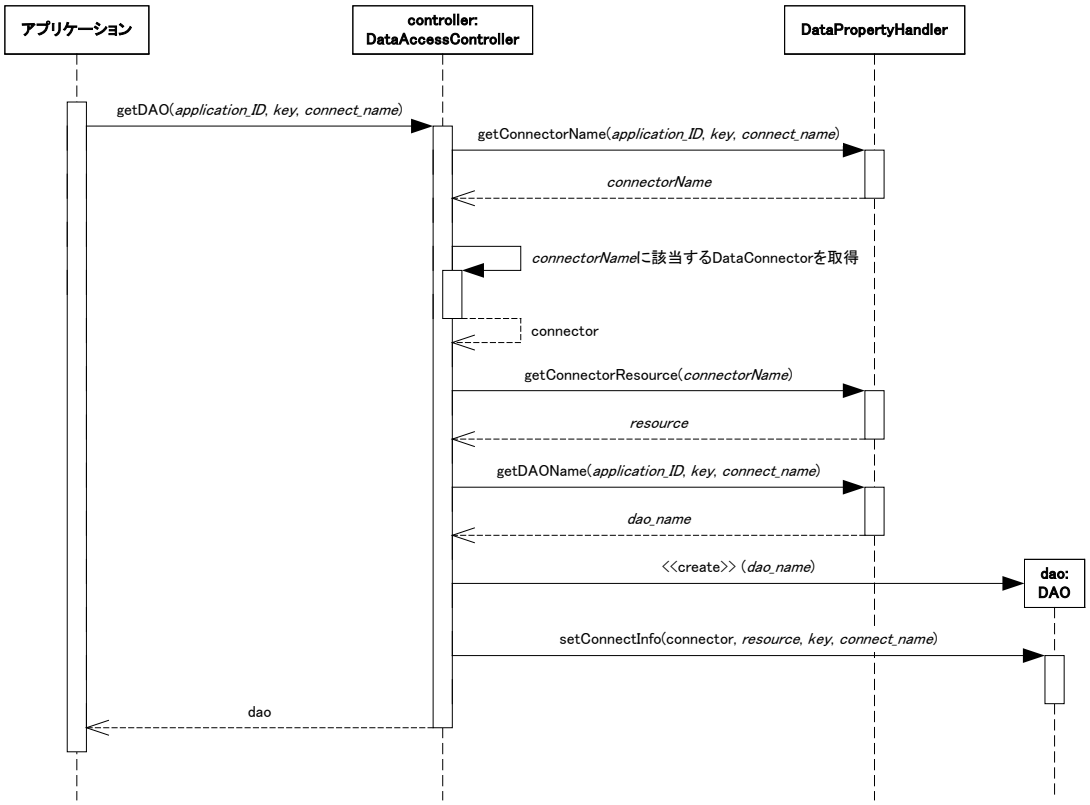


Figure 5-3 Obtaining DAO

You could obtain and directly handle DAO. However, it is strongly recommended to prepare DAO interface (DAOIF) and create the application that uses DAO which implements DAOIF. By doing this, you only have to change DAO setting when the resource type is changed and you do not have to make a correction on application that uses DAO. Please refer to [5.3.1.3 DAOIF] for details.

5.3.1.1 DAO provided as Standard

In IM-JavaEE Framework, some DAOs that are specialized for the resource considered to be used often are provided. These DAO are all abstract class so it cannot generate instance derectly, but the developer can reduce the coding amount by creating subclass of this DAO.

5.3.1.1.1 DBDAO

DBDAO is DAO that is specialized for the database. If you use DBDAO, you need to specify the subclass of jp.co.intra_mart.framework.base.data.DBConnector as DataConnector (refer to [Figure 5-4 DBDAO Structure].).

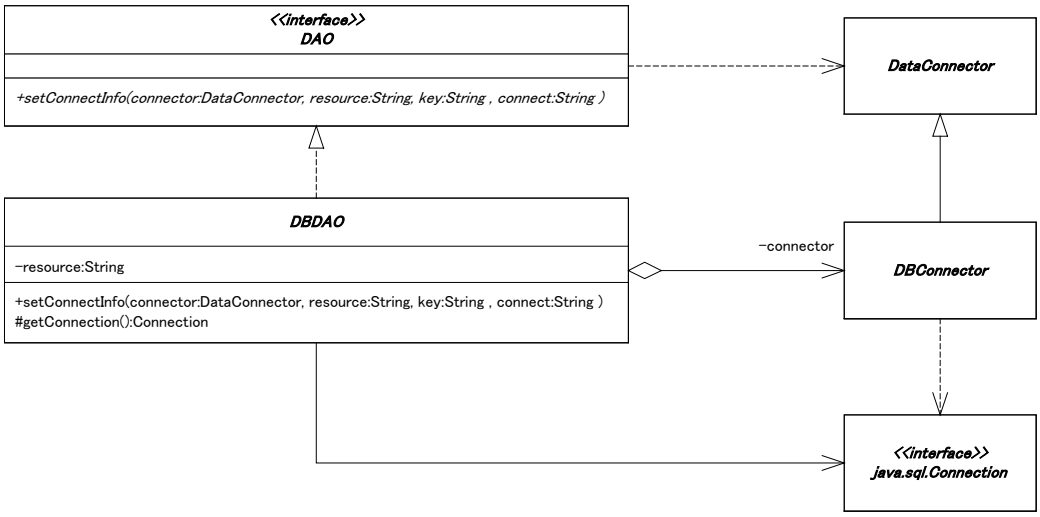


Figure 5-4 DBDAO Structure

In order to access to the database, you have to go through java.sql.Connection interface. In DBDAO, if the appropriate DataConnector is set, you only have to call getConnection method to obtain java.sql.Connection (refer to [Figure 5-5 Obtaining Connection (DBDAO)].

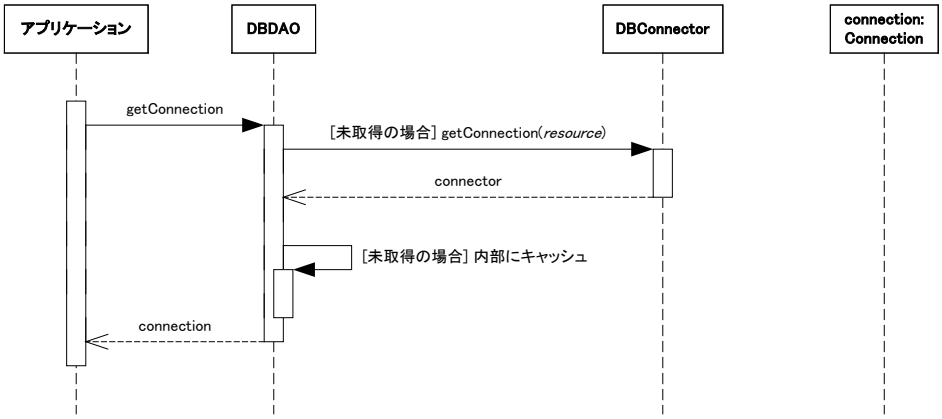


Figure 5-5 Obtaining Connection (DBDAO)

5.3.1.1.2 **TenantDBDAO**

It is DAO that is specialized for the access to tenant database[6] that is set in TenantDBDAO intra-mart.

If you use TenantDBDAO, jp.co.intra_mart.framework.base.data.TenantDBConnector or its subclass should be specified as DataConnector (refer to [Figure 5-6 TenantDBDAO Structure]).

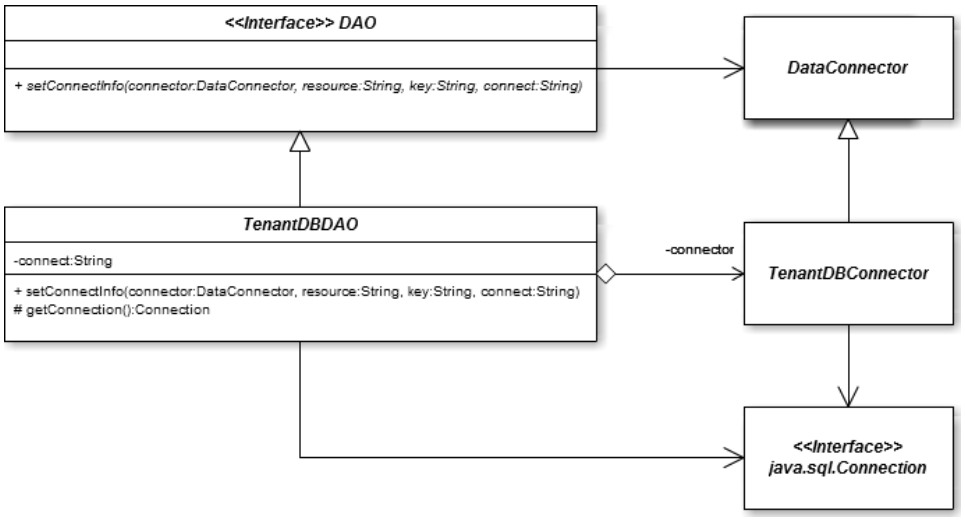


Figure 5-6 TenantDBDAO Structure

In order to access to the database, you should go through java.sql.Connection interface. In TenantDBDAO, if the appropriate DataConnector is set, you only have to call getConnection method to obtain java.sql.Connection (refer to [Figure 5-7 Obtaining Connection (TenantDBDAO].

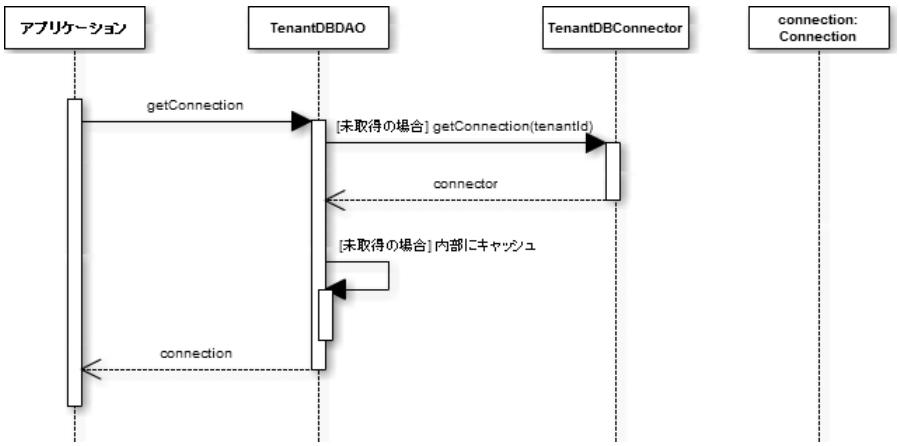


Figure 5-7 Obtaining Connection (TenantDBDAO)

5.3.1.1.3 **SharedDBDAO**

SharedDBDAO is DAO that is specialized for accessing shared database[7] which was set in intra-mart. If you use SharedDBDAO, you need to specify jp.co.intra_mart.framework.base.data. SharedDBConnector or its subclass as DataConnector (refer to [Figure 5-8 SharedDBDAO Structure].

6 Database that is set by WEB-INF/conf/data-source-mapping-config.xml of intra-mart. It is defined as tenant-data-source.

7 Database that is set by WEB-INF/conf/data-source-mapping-config.xml of intra-mart. It is defined as shared-data-source.

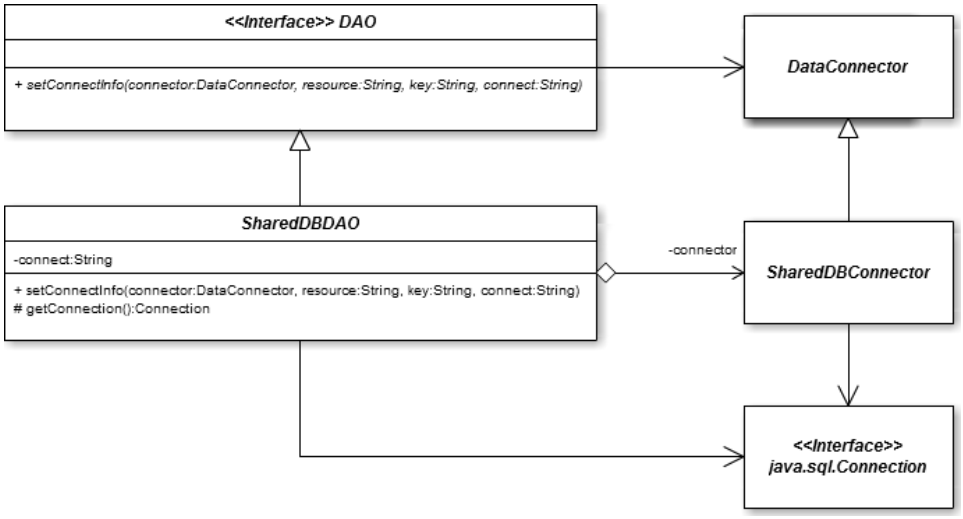


Figure 5-8 SharedDBDAO Structure

In order to access to the database, you have to go through java.sql.Connection interface. In SharedDBDAO, if the appropriate DataConnector, you only have to call getConnection method to obtain java.sql.Connection (refer to [Figure 5-9 Obtaining Connection (SharedDBDAO]).

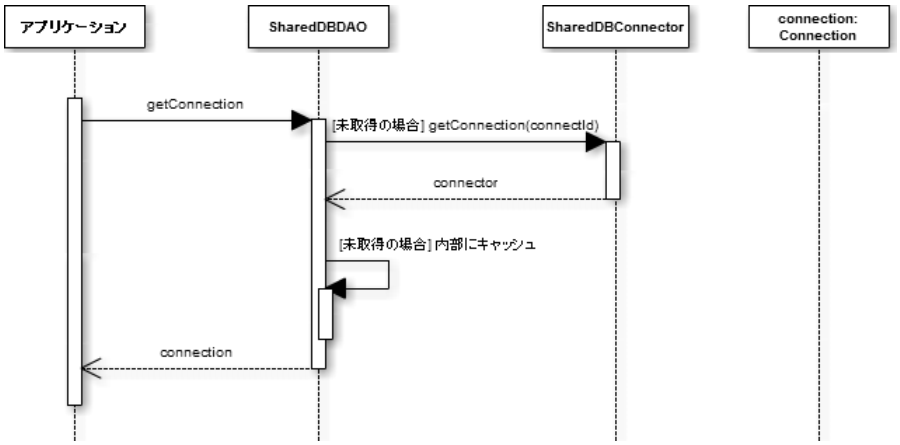


Figure 5-9 Obtaining Connection (SharedDBDAO)

5.3.1.1.4 **IntramartDBDAO**

IntramartDBDAO is DAO that is specialized for accessing database[8] which was set in intra-mart. If you use IntramartDBDAO, jp.co.intra_mart.framework.base.data.IntramartDBConnector or its subclass should be specified as DataConnector (refer to [Figure 5-10 IntramartDBDAO Structure]).

DbsConnection is not recommended method and it will not be operated unless you install it as compatible mode. Therefore, IntramartDBDAO is operated when compatible module is installed.

Using TenantDBDAO or SharedDBDAO is preferred.

[8] Database that is set by WEB-INF/conf/data-source-mapping-config.xml of intra-mart.
Same ID should be set for tenant-data-source and for shared-data-source.

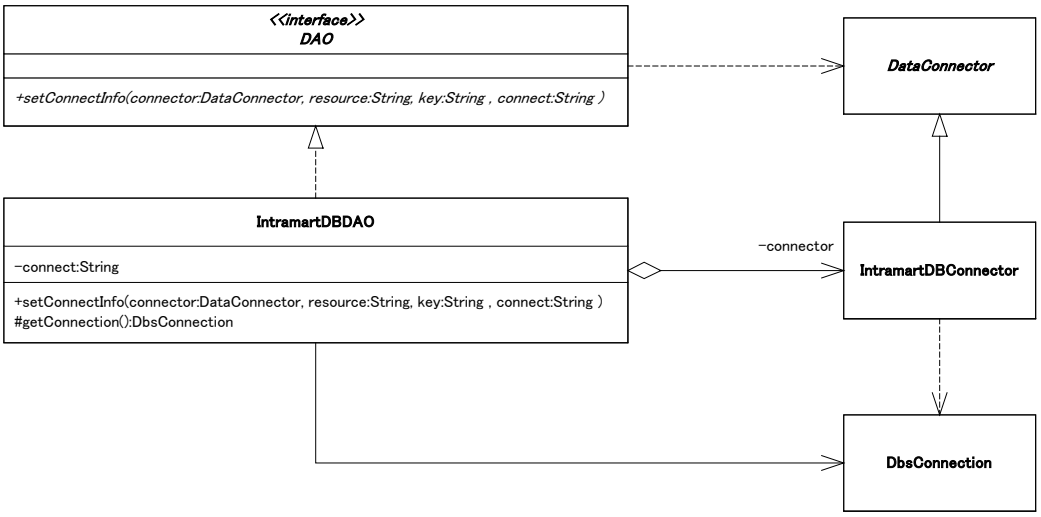


Figure 5-10 IntramartDBDAO Structure

In order to access to the database specified in intra-mart, you should access to database through jp.co.intra_mart.foundation.database.DbsConnection. In IntramartDBDAO, you only have to call getConnection method to obtain DbsConnection (refer to [Figure 5-11 Obtaining Connection (IntramartDBDAO].

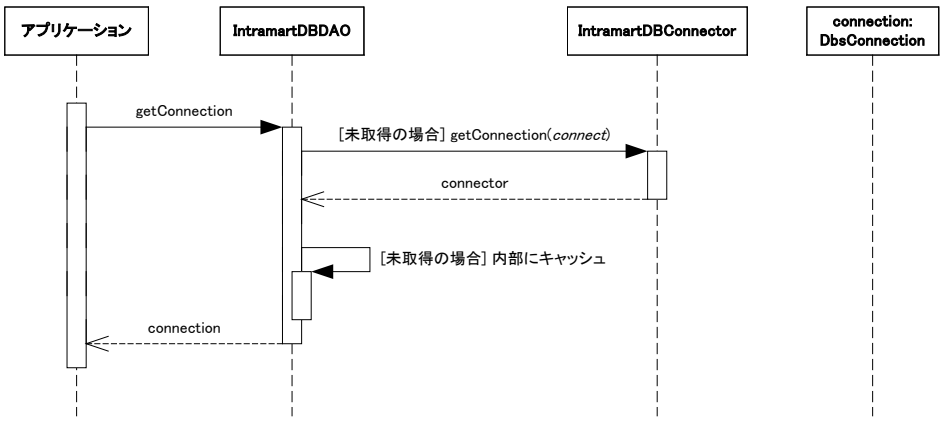


Figure 5-11 Obtaining Connection (IntramartDBDAO)

5.3.1.1.5 **IntramartStorageDAO**

IntramartStorageDAO is DAO that is specialized for accessing file that is operating in Public Storage of intra-mart. If you use IntramartStorageDAO, you should specify jp.co.intra_mart.framework.base.data.IntramartStorageConnector or its subclass as DataConnector (refer to [Figure 5-12 IntramartStorageDAO Structure]).

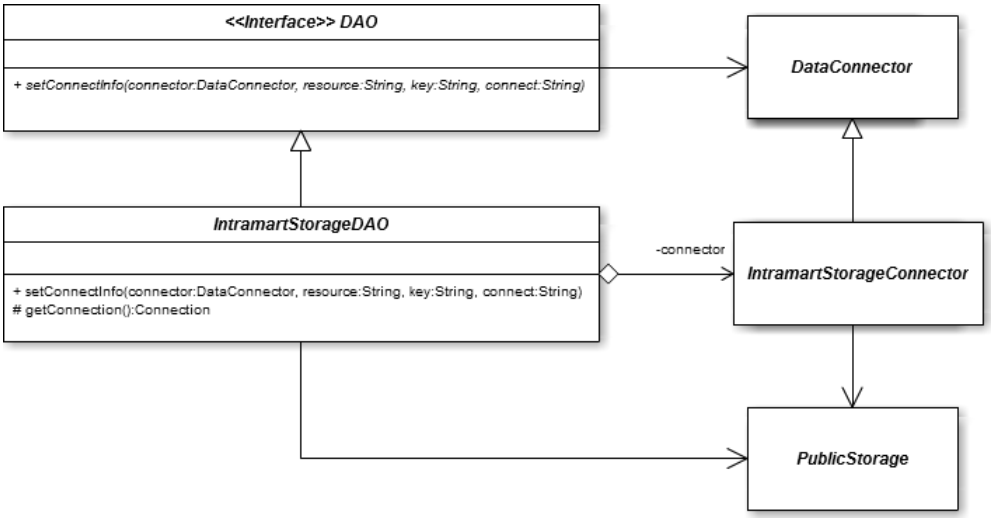


Figure 5-12 IntramartStorageDAO Structure

In order to use Storage Service, you should access to the file through jp.co.intra_mart.foundation.service.client.file.PublicStorage. in IntramartStorageDAO, you only have to call getStorageFile method to obtain PublicStorage (refer to [Figure 5-13 Obtaining Connection (IntramartStorageDAO]).

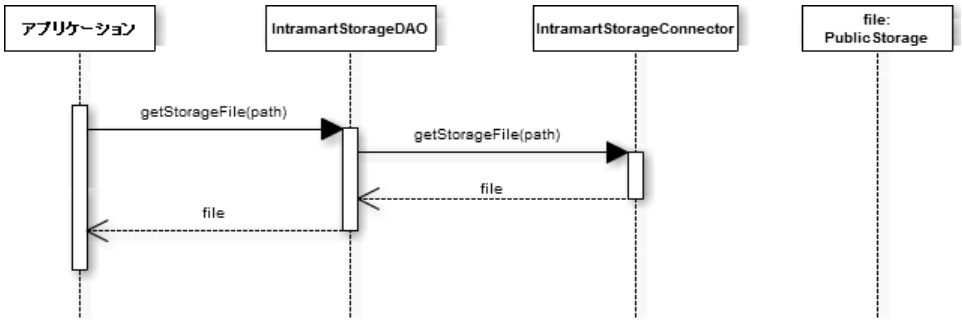


Figure 5-13 Obtaining Connection (IntramartStorageDAO)

5.3.1.2 Original DAO

If the developer create DAO originally, the developer has to meet the following requirements.

- jp.co.intra_mart.framework.base.data.DAO interface is implemented.
- public default constructor (constructor with no argument) is defined.
- Appropriate connection information should be set in setConnectInfo method.

Followings are preferred to be implemented, but not required.

- It is abstract class.
- Simple method to access to the specific resource through specific DataConnector is implemented. For example, DBDAO has getConnection method to access to the relational database through subclass of DBConnector.

5.3.1.3 DAOIF

Application accesses to the resource through DAO. DAO can be obtained by getDAO method of DataAccessController. Since the return value type of this method is java.lang.Object, the application that uses DAO should be down-casted for the appropriate class. In this case, it is recommended to downcast to DAOIF (interface that corresponding DAO method is defined) instead of downcasting to the DAO class to be used directly.

If DAOIF is used, you only have to switch the DAO when the resource type is changed and no need to correct the application that uses DAO. And in order to utilize the merit of DAOIF, application shoud do coding to DAOIF, not to DAO.

As an example, suppose there is DAO as shown in [Figure 5-14 Use of DAOIF].

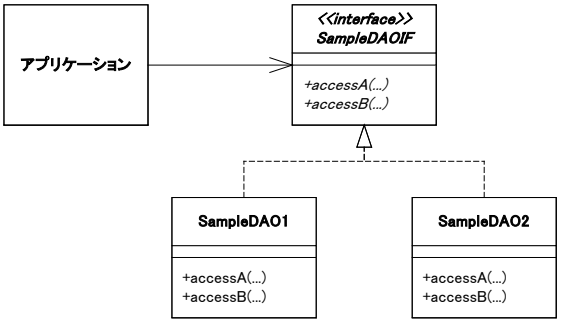


Figure 5-14 Use of DAOIF

Application code that accesses to the resource without using DAOIF is shown in [List 5-1 When DAOIF is not used]. It operates fine with this code too, but if the resource type that you use has been changed, you have to correct the application unless you don't change the implementation of SampleDAO1.

List 5-1 When DAOIF is not used

```
・・・
// DataAccessController has been obtained as controller
SampleDAO1 dao = (SampleDAO1)controller.getDAO("app1", "key1", "con1");
dao.access(...);
・・・
```

Application code to access the resource by using DAOIF is shown in [List 5-2 When DAOIF is used (recommended)]. In this code, please note that the coding is for SampleDAOIF. And in this code, if the resource type to be used is changed, SampleDAO1 should be switched to SampleDAO2 in property setting, but no need to make a correction for the application.

List 5-2 When DAOIF is used (recommended)

```
・・・
// DataAccessController has been obtained as controller
SampleDAOIF dao = (SampleDAOIF)controller.getDAO("app1", "key1", "con1");
dao.access(...);
・・・
```

5.3.2 DataAccessController

DataAccessController has 2 roles.

- Obtaining DAO
- Transaction management

5.3.2.1 Obtaining DAO

DAO can be obtained through DataAccessController. This is depicted in [Figure 5-15 Obtaining DAO].

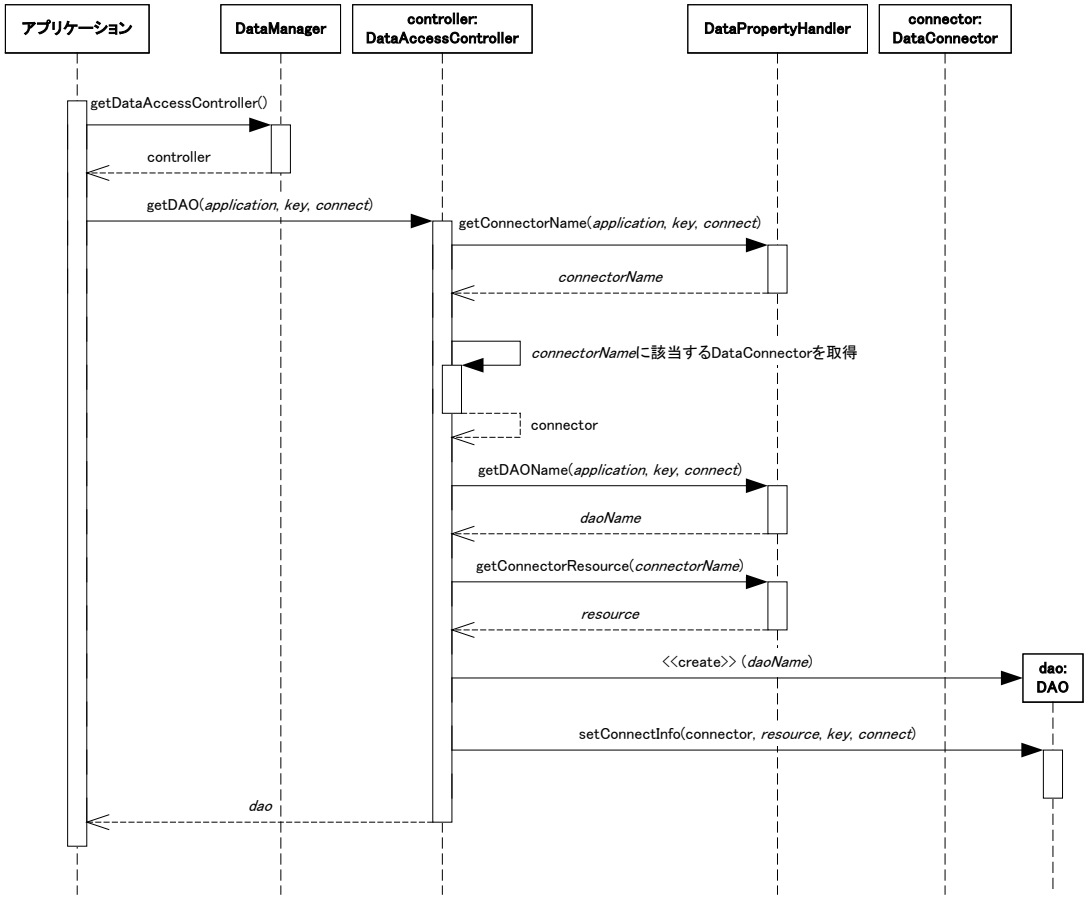


Figure 5-15 Obtaining DAO

Following processes are performed in [Figure 5-15 Obtaining DAO].

1. Application obtains DataAccessController from DataManager (getDataAccessController).
2. Application passes application ID, key, and connection name and requests to obtain DAO for the obtained DataAccessController (getDAO).
3. DataAccessController obtains deta connector name that corresponds to the application ID, the key, and the connection name (getConnectorName).
4. DataAccessController obtains DataConnector that corresponds to the obtained data connector name.
5. DataAccessController obtains DAO class name that corresponds to the application ID, the key, and the connection name (getDAOName).
6. DataAccessController obtains the resource name that corresponds to the application ID, the key, and the connection name (getConnectorResource).
7. DataAccessController generates DAO based on the obtained DAO class name.
8. DataAccessController sets DataConnector to the generated DAO (setConnectorInfo)
9. DataAccessController returns DAO to the application.

DataAccessController puts DataConnector to cache with data connector name as a key. When DataAccessController obtains DataConnector, it first searches its own cache. If it cannot be found in the cache, it obtains class name of DataConnector corresponding to data connector name from DataPropertyHanler (getConnectorClassName), and generates DataConnector newly. This DataConnector will be added to the cache and re-used in the same DataAccessController.

Behavior of DataConnector cache is shown in [Figure 5-16 Obtaining DataConnector (when it is cacheed)] and [Figure 5-17 Obtaining DataConnector (when it is not cacheed)）].

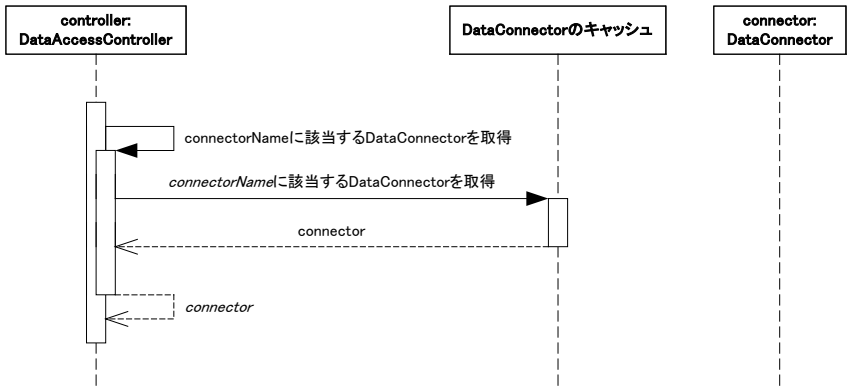


Figure 5-16 Obtaining DataConnector (when it is cacheed)

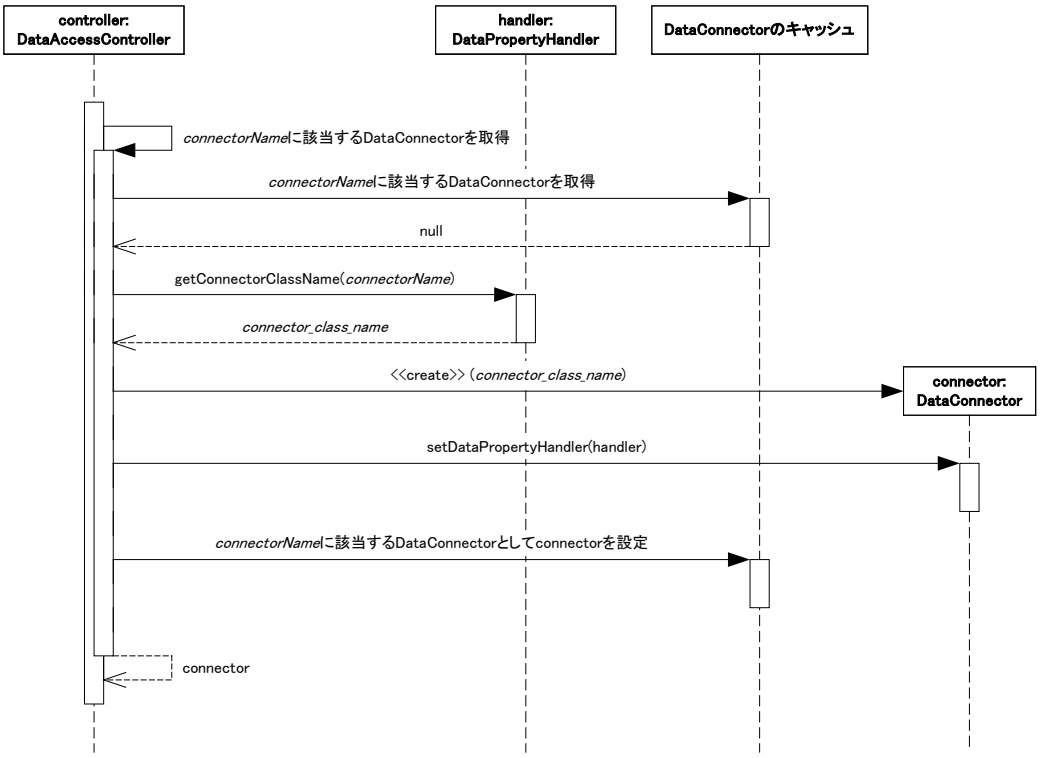


Figure 5-17 Obtaining DataConnector (when it is not cacheed))

5.3.2.2 Transaction Management

Simple transaction can be actualized by using DataAccessController. Transaction management by this method is not recommended. Please refer to [5.5.1.2 Transaction Management by DataAccessController].

5.3.3 DataConnector

DataConnector has a role to connect DAO and the resource. DataConnector that can be used is limited depending on DAO, so the DataConnector to be used should be considered for each DAO (refer to [5.3.1 DAO].)

5.3.3.1 DataConnector provided as Standard

In IM-JavaEE Framework, following DataConnector considered to be used often are provided in advance.

- DBConnector

  DataConnector to connect to the relational database. This is abstract class.

- JDBCConnector
  DataConnector to connect by JDBC. This is defined as subclass of DBConnector.
- DataSourceConnector
  DataConnector to connect through data source. This is defined as the subclass of DBConnector.
- IntramartDBConnector
  DataConnector to connect to the database that is managed in intra-mart.
- IntramartStorageConnector
  DataConnector to connect to Public Storage of intra-mart.

Followings are the descriptions.

5.3.3.1.1 DBConnector

The main role of DBConnector is to obtain java.sql.Connection in order to connect to RDBMS. DBConnector is abstract class and the obtaining method of Connection itself is depending on the subclass. The connector of this subclass is mainly aimed to be used in DBDAO.

DBConnector structure is shown in [Figure 5-18 DBConnector Structure].

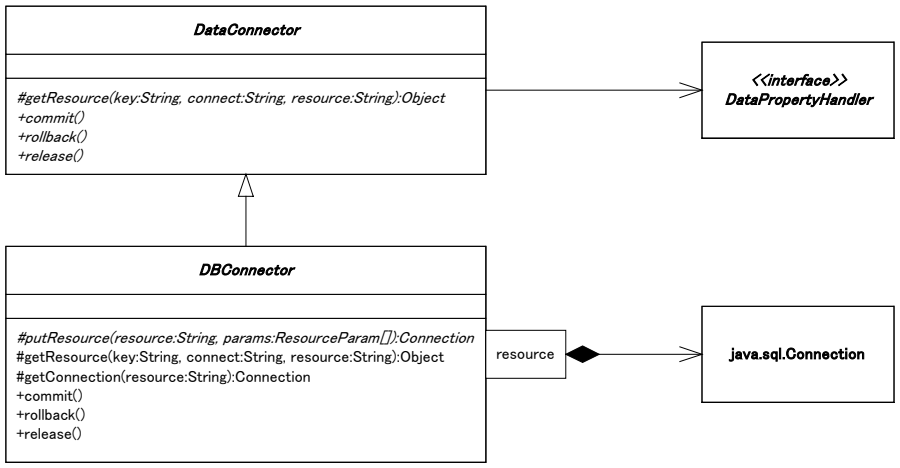


Figure 5-18 DBConnector Structure

If there is a connection request (getConnection), DBConnector searches connection that matches with the resource name from the Connection cache that is retained for itself. If Connection has not been obtained yet, it newly obtains Connection which obtains resource information corresponding to the resource name from DataPropertyHandler (getResourceParams), and add the obtained Connection to cache. The cached Connection will be re-used in the same DBConnector.

How the Connection is re-used is shown in [Figure 5-19 Obtaining Connection from DBConnector (obtained] and [Figure 5-20 Obtaining Connection from DBConnector (new))].

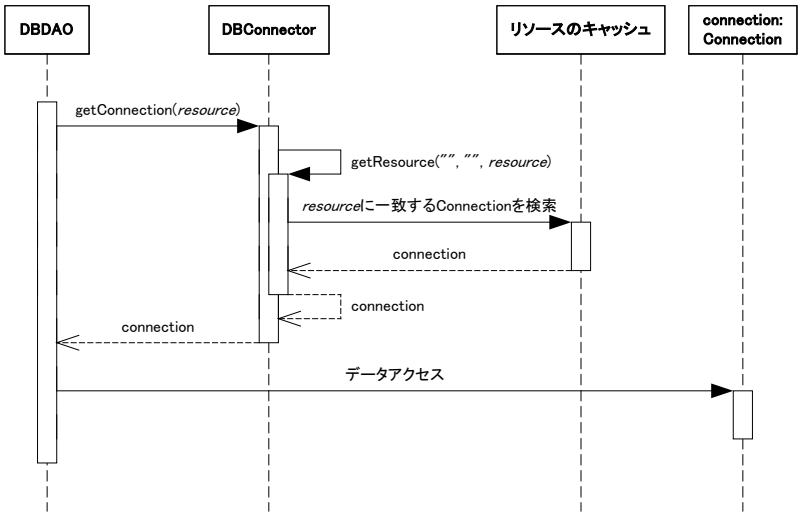


Figure 5-19 Obtaining Connection from DBConnector (obtained)

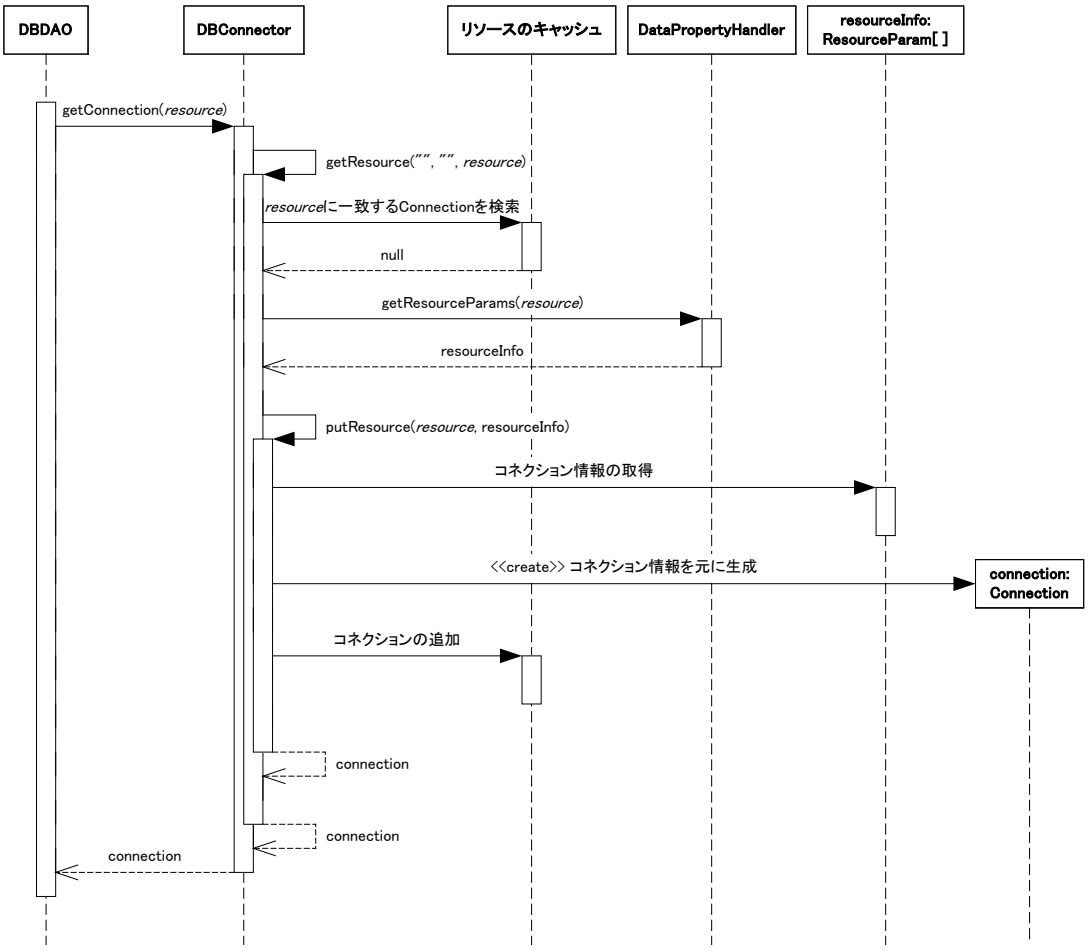


Figure 5-20 Obtaining Connection from DBConnector (new))

5.3.3.1.2 **JDBCConnector**

The use of JDBCConnector is not recommended. The use of DataSourceConnector is recommended instead.

JDBCConnector provides a simple function to connect to the RDBMS. JDBCConnector is aimed to be used in DBDAO mainly.

When JDBCConnector is used, the setting shown in [Table 5-1 Setting Contents of JDBCConnector] will be required for the data property.

Table 5-1 Setting Contents of JDBCConnector

Item	Contents	
DataConnector	jp.co.intra_mart.framework.base.data.JDBCConnector	
Data connector name	Optional	
Resource name	Optional	
Resource parameters	driver	Driver class name that connects to the database
	url	URL that connects to the database
	username	User name that connects to the database
	password	User password that connects to the database

If XmlDataPropertyHandler is specified as DataPropertyHandler, the examples of setting contents that corresponds to [Table 5-1 Setting Contents of JDBCConnector] are shown in [List 5-3 Setting Examples of data-config.xml (JDBCConnector)] amd [List 5-4 Setting Examples of data-config-application.xml (JDBCConnector)]. In this example, application ID is application, data connector name is myCon, resource name is myResource, driver class name is sample.jdbc.driber.SampleDriver, URL when connecting to the database is jdbc:sampledb:sample, connection user of the database if imart, and password is imartpass.

List 5-3 Setting Examples of data-config.xml (JDBCConnector)

```
...
  <connector>
      <connector-name>myCon</connector-name>
      <connector-class>jp.co.intra_mart.framework.base.data.JDBCConnector</connector-class>
      <resource-name>myResource</resource-name>
  </connector>

  <resource>
      <resource-name>myResource</resource-name>
      <init-param>
          <param-name>driver</param-name>
          <param-value>sample.jdbc.driber.SampleDriver</param-value>
      </init-param>
      <init-param>
          <param-name>url</param-name>
          <param-value>jdbc:sampledb:sample</param-value>
      </init-param>
      <init-param>
          <param-name>username</param-name>
          <param-value>imart</param-value>
      </init-param>
      <init-param>
          <param-name>password</param-name>
          <param-value>imartpass</param-value>
      </init-param>
  </resource>
...
```

List 5-4 Setting Examples of data-config-application.xml (JDBCConnector)

```
・・・
  <dao-group>
      <dao-key>key</dao-key>
      <dao>
        <dao-class>・・・</dao-class>
        <connector-name>myCon</connector-name>
      </dao>
 </dao-group>
・・・
```

JDBCConnector is not recommended due to following reasons.

- It cannot be managed by the same transaction with UserTransaction.
- Normally, there is no Connection pooling function other than data source on general application server.

JDBCConnector Structure is shown in [Figure 5-21 JDBCConnector Structure].

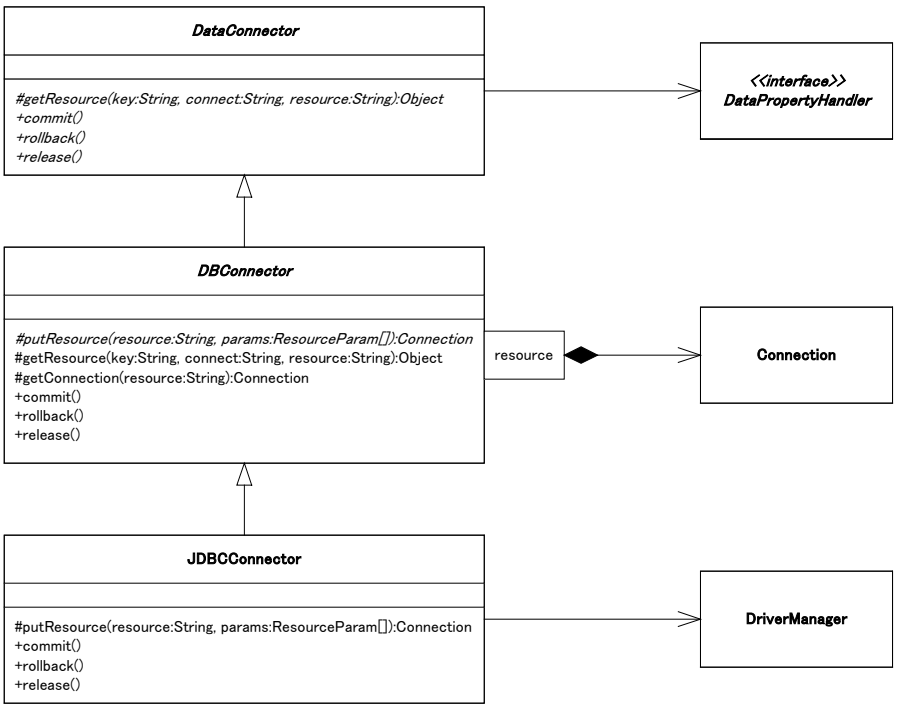


Figure 5-21 JDBCConnector Structure

Obtaining Connection by using JDBCConnector is shown in [Figure 5-22 Obtaining Connection (JDBCConnector]. The developer who does coding by using DAO nomally does not have to have deep understanding on this operation.

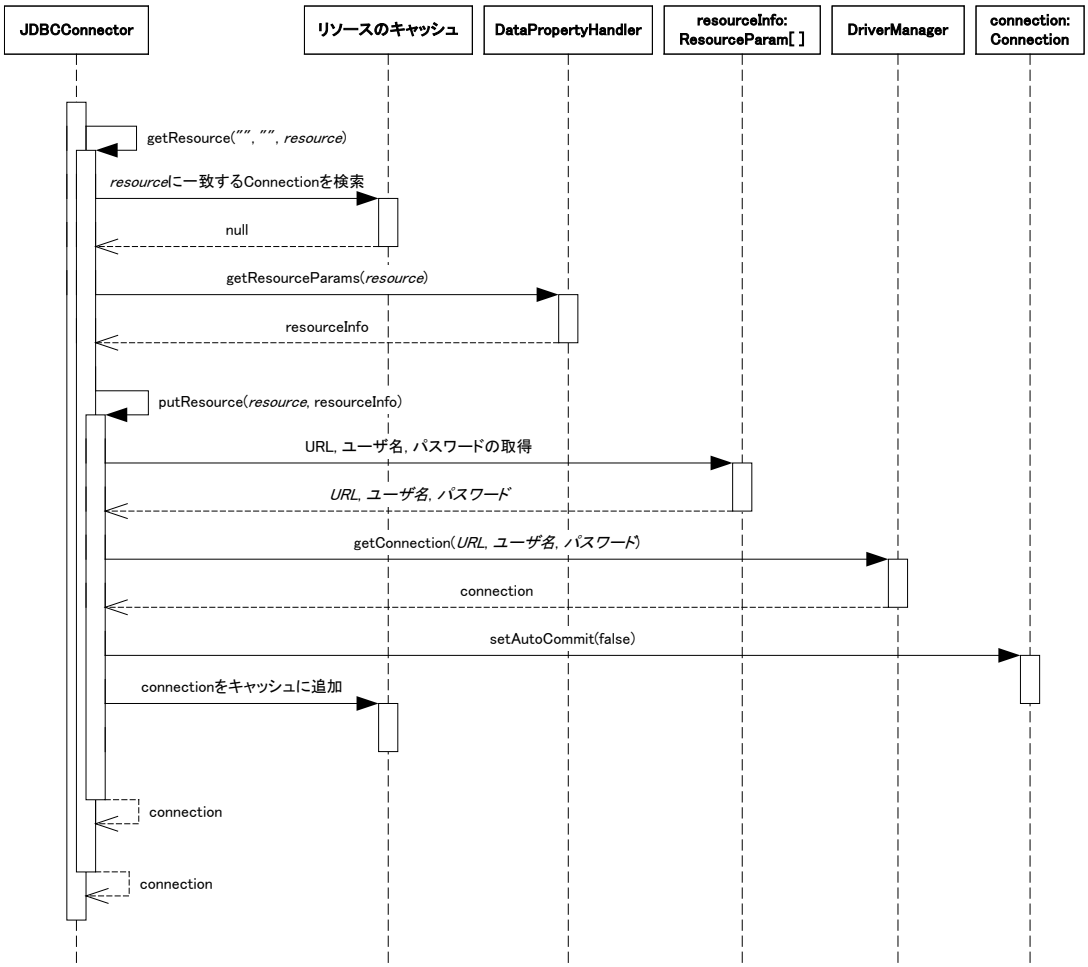


Figure 5-22 Obtaining Connection (JDBCConnector)

When commit or rollback is performed for JDBCConnector, it is depicted in [Figure 5-23 Commit or Rollback (JDBCConnector)].

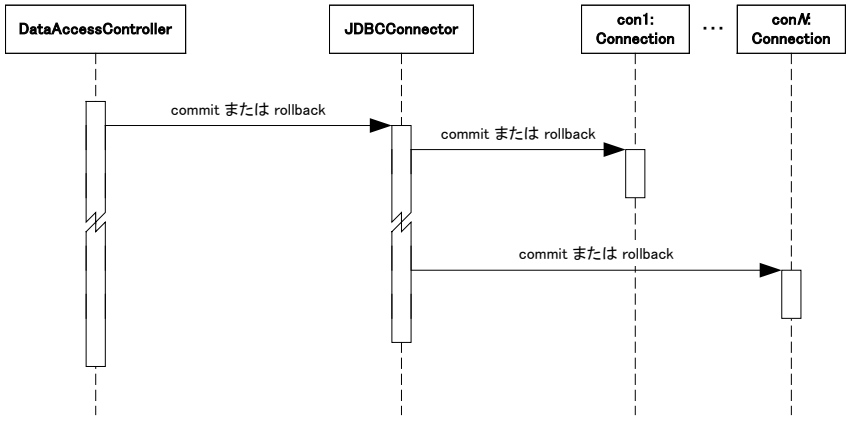


Figure 5-23 Commit or Rollback (JDBCConnector)

As you can see in [Figure 5-23 Commit or Rollback (JDBCConnector)], JDBCConnector does commit or rollback for every Connection it owns. During the commit operation, if any one of the commits fails, rollback is applied to

all the subsequent Connections but nothing will be done to the Connections already committed[9].

5.3.3.1.3 **DataSourceConnector**

DataSourceConnector provides simple function to connect to RDBMS. DataSourceConnector is mainly aimed to be used in DBDAO. The differences with JDBCConnector are described below.

- It obtains Connection for the database from DataSource that is registered in the application server.
- It is managed by the same transaction with UserTransaction.

When you use DataSourceConnector, you need to set the setting shown in [Table 5-2 Setting Contents of DataSourceConnector] to the data property.

Table 5-2 Setting Contents of DataSourceConnector

Item	Contents	
DataConnector	jp.co.intra_mart.framework.base.data.DataSourceConnector	
Data connector name	Optional	
Resource name	Optional	
Resource parameter	jndi	Lookup name of DataSource

If you specify XmlDataPropertyHandler as DataPropertyHandler, the examples of setting contents that corresponds to [Table 5-2 Setting Contents of DataSourceConnector] are shown in[List 5-5 Setting Example of data-config.xml (DataDourceConnector] and [List 6-6 Setting Example of data-config-application.xml (DataDourceConnector]. In this example, application ID is application, data connector name is, data connector name is myCon, and resource name is myResource. Application server setup so that data source can be looked up by "java:comp:env/jdbc/mydb".

List 5-5 Setting Example of data-config.xml (DataDourceConnector)

```
・・・
<connector>
      <connector-name>myCon</connector-name>
  <connector-class>jp.co.intra_mart.framework.base.data.DataSourceConnector</connector-class>
  <resource-name>myResource</resource-name>
</connector>
・・・
<resource>
  <resource-name>myResource</resource-name>
  <init-param>
      <param-name>jndi</param-name>
      <param-value>java:comp:env/jdbc/mydb</param-value>
  </init-param>
</resource>
・・・
```

[9] This indicates that it violates the atomicity of transation (Atomicity: Processes inside the transaction should either all succeed or all fail. Therefore, the use of JDBCConnector is not recommended.

List 5-6 Setting Example of data-config-application.xml (DataDourceConnector)

```
・・・
<dao-group>
  <dao-key>key</dao-key>
  <dao>
  <dao-class>・・・</dao-class>
    <connector-name>myCon</connector-name>
  </dao>
</dao-group>
・・・
```

DataSourceConnector structure is shown in [Figure 5-24 DataSourceConnector Structure].

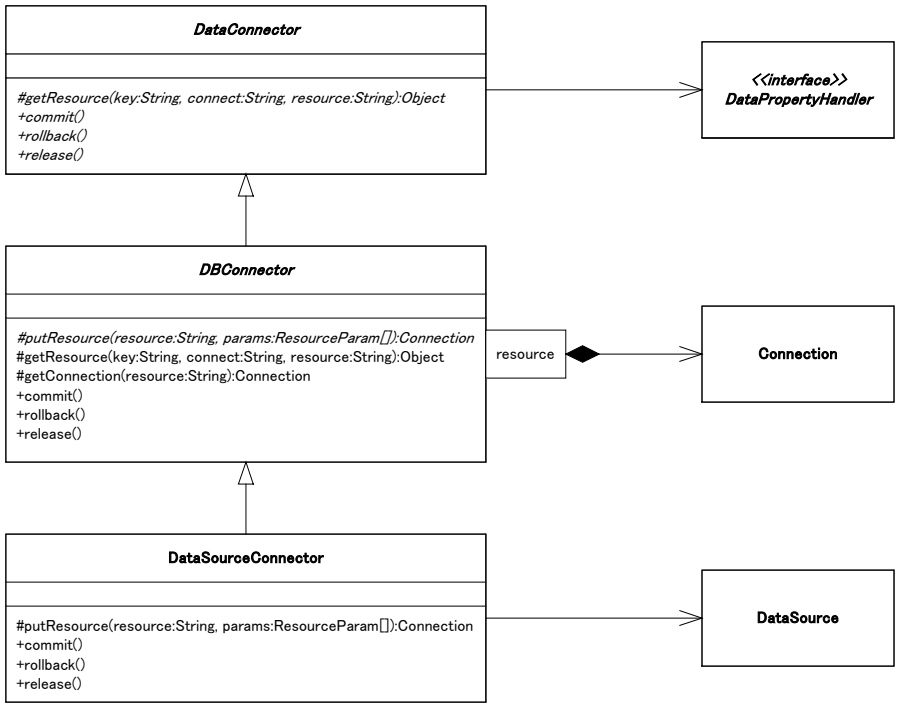


Figure 5-24 DataSourceConnector Structure

Obtaining Connection by using DataSourceConnector is shown in [Figure 5-25 Obtaining Connection (DataSourceConnector]. The developer who does coding by using DAO normally doesn't have to have deep understanding of this operation.

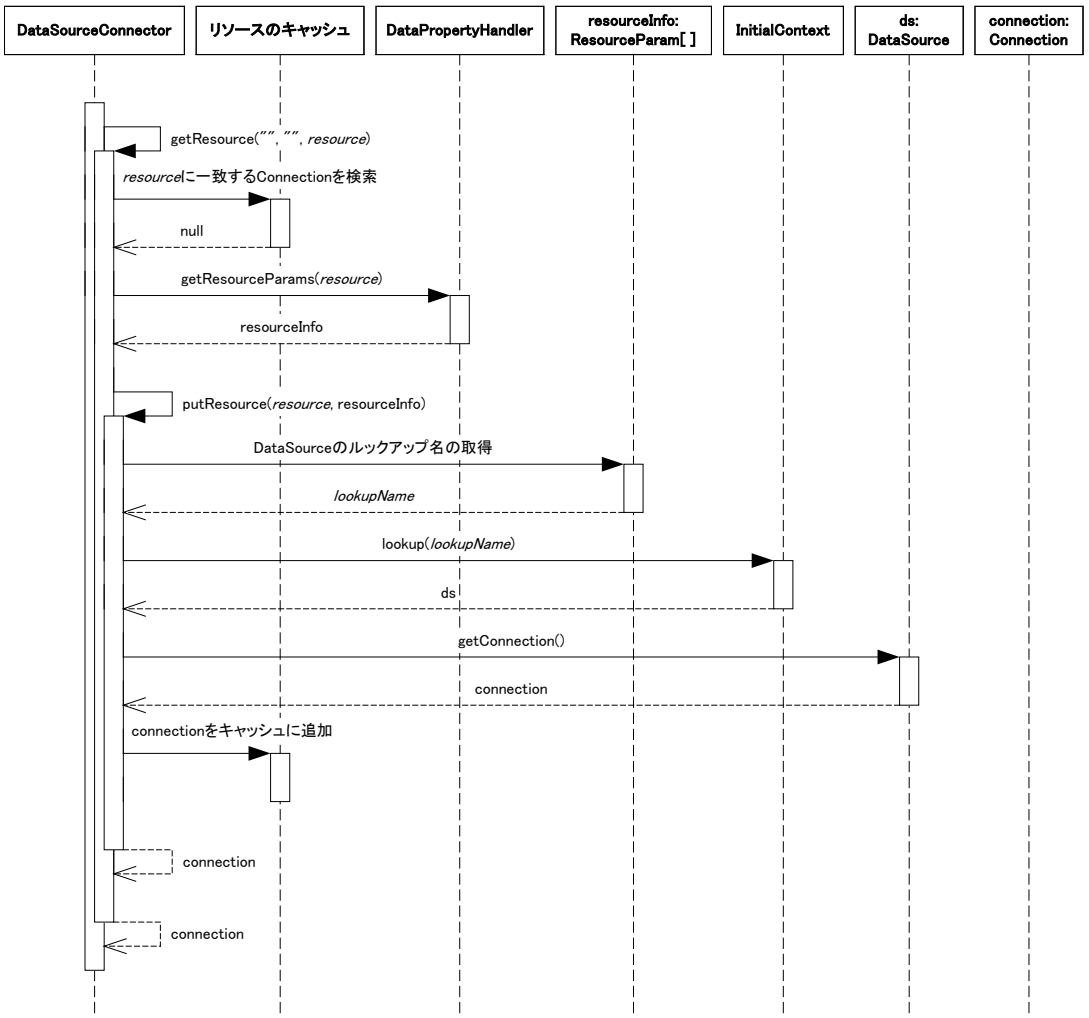


Figure 5-25 Obtaining Connection (DataSourceConnector)

When commit or rollback is performed for DataSourceConnector is shown in [Figure 5-26 Commit or Rollback (DataSourceConnector].

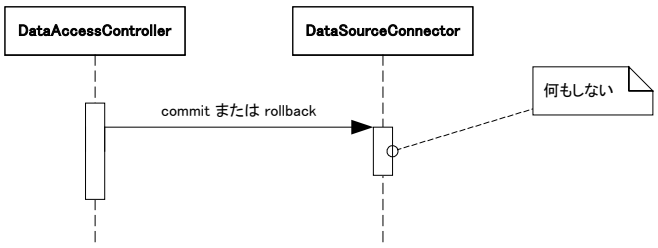


Figure 5-26 Commit or Rollback (DataSourceConnector)

5.3.3.1.4 TenantDBConnector

TenantDBConnector provides a function to connect to RDBMS that is set to the tenant database of intra-mart. TenantDBConnector is mainly aimed to be used in TenantDBDAO. RDBMS that is set in intra-mart uses DataSource to connect, so the same merit with DataSourceConnector can be obtained.

If you use TenantDBConnector, you need to set the setting shown in [Table 5-3 Setting Contents of TenantDBConnector] to data property.

Table 5-3 Setting Contents of TenantDBConnector

Item	Contents
DataConnector	jp.co.intra_mart.framework.base.data.TenantDBConnector
Data connector name	Optional
Resource name	(None)

If you specify XmlDataPropertyHandler as DataPropertyHandler, the example of setting contents that corresponds to [List 5-7 Setting Example of data-config.xml (TenantDBConnector] is shown in [List 5-8 Setting Example of data-config-application.xml (TenantDBConnector)]. In this example, application ID is application and data connector name is myCon.

List 5-7 Setting Example of data-config.xml (TenantDBConnector)

```
・・・
<connector>
   <connector-name>tenant_db</connector-name>
<connector-class>
jp.co.intra_mart.framework.base.data.TenantDBConnector
</connector-class>
  </connector>
・・・
```

List 5-8 Setting Example of data-config-application.xml (TenantDBConnector)

```
・・・
<dao-group>
  <dao-key>key</dao-key>
  <dao>
<dao-class>・・・</dao-class>
    <connector-name>tenant_db</connector-name>
  </dao>
</dao-group>
・・・
```

TenantDBConnector Structure is shown in [Figure 5-27 TenantDBConnector Structure].

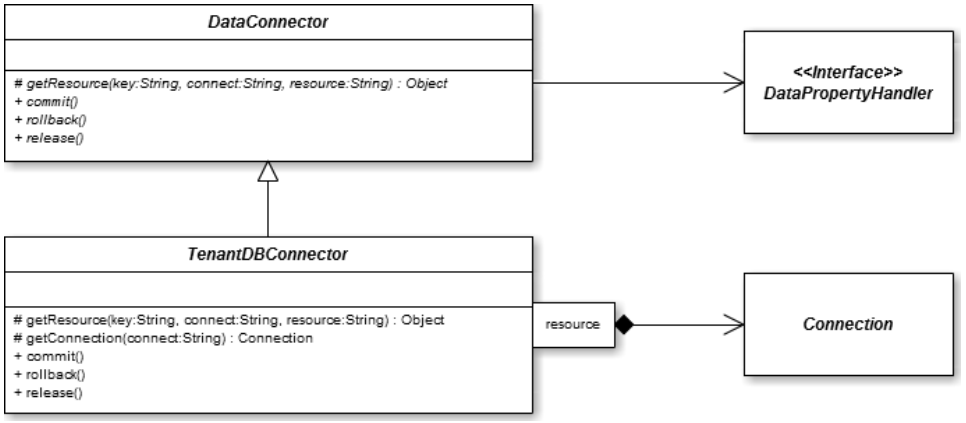


Figure 5-27 TenantDBConnector Structure

Obtaining Connection by using TenantDBConnector is shown in [Figure 5-28 Obtaining Connection (TenantDBConnector)]. The developer who does coding by using DAO normally does not have to have deep understanding of this operation.

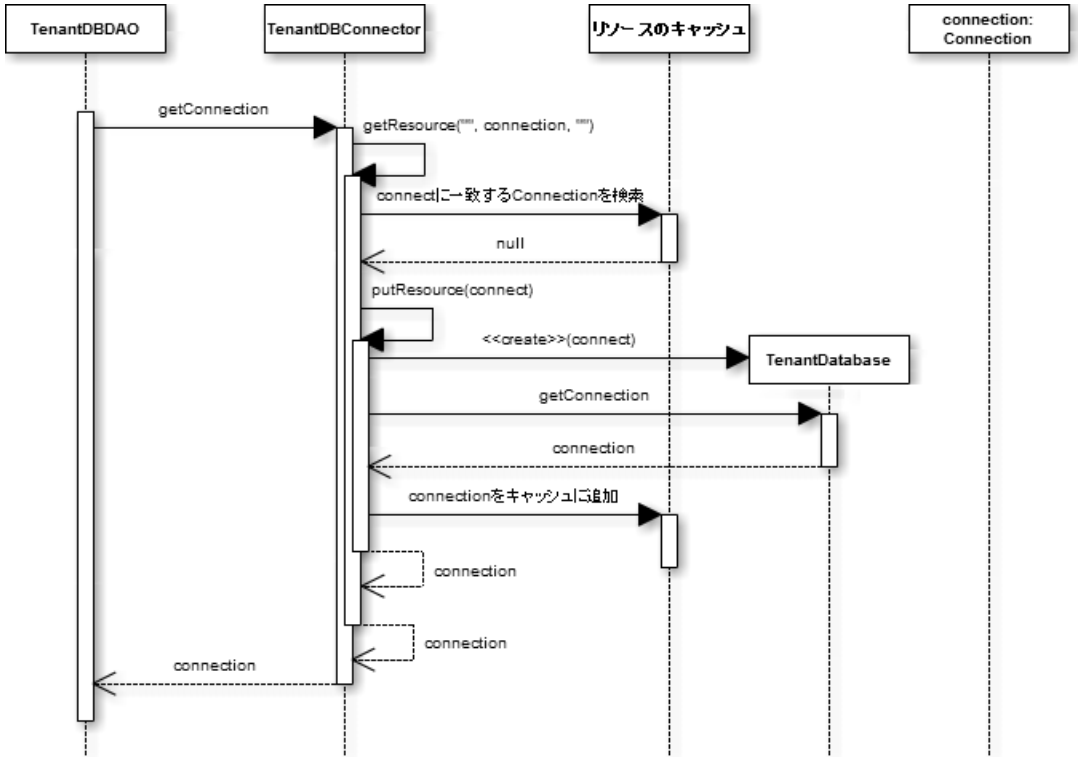


Figure 5-28 Obtaining Connection (TenantDBConnector)

Nothing is performed for commit or rollback in TenantDBConnector. How the commit or rollback is performed for TenantDBConnector is shown in [Figure 5-29 Commit or Rollback (TenantDBConnector)].

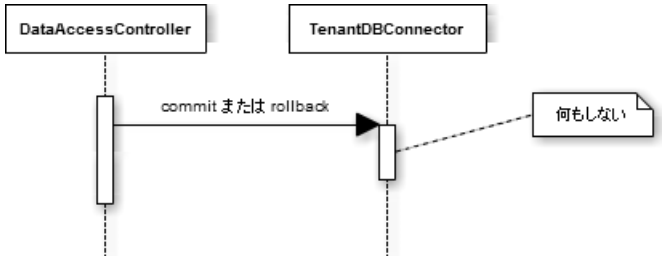


Figure 5-29 Commit or Rollback (TenantDBConnector)

5.3.3.1.5 **SharedDBConnector**

SharedDBConnector provides a function to connect to RDBMS that is set to the shared database of intra-mart. SharedDBConnector is mainly aimed to be used in SharedDBDAO. RDBMS that is set in intra-mart uses DataSource to connect, so the same merit with DataSourceConnector can be obtained.

If you use SharedDBConnector, you have to set the setting shown in [Figure 5-30 Setting Contents of SharedDBConnector] to data property.

Figure 5-30 Setting Contents of SharedDBConnector

Item	Contents
DataConnector	jp.co.intra_mart.framework.base.data.SharedDBConnector
Data connector name	Optional
Resource name	(None)

If you specify XmlDataPropertyHandler as DataPropertyHandler, the example of setting contents that corresponds to [List 5-9 Setting Example of data-config.xml (SharedDBConnector)] is shown in [List 5-10 Setting Example of data-config-application.xml (SharedDBConnector)]. In this example, application ID is application and data connector is myCon.

List 5-9 Setting Example of data-config.xml (SharedDBConnector)

```
・・・
<connector>
    <connector-name>shared_db</connector-name>
<connector-class>
jp.co.intra_mart.framework.base.data.SharedDBConnector
</connector-class>
  </connector>
・・・
```

List 5-10 Setting Example of data-config-application.xml (SharedDBConnector)

```
・・・
<dao-group>
  <dao-key>key</dao-key>
  <dao>
<dao-class>・・・</dao-class>
    <connector-name>shared_db</connector-name>
  </dao>
</dao-group>
・・・
```

SharedDBConnector structure is shown in [Figure 5-31 SharedDBConnector Structure].

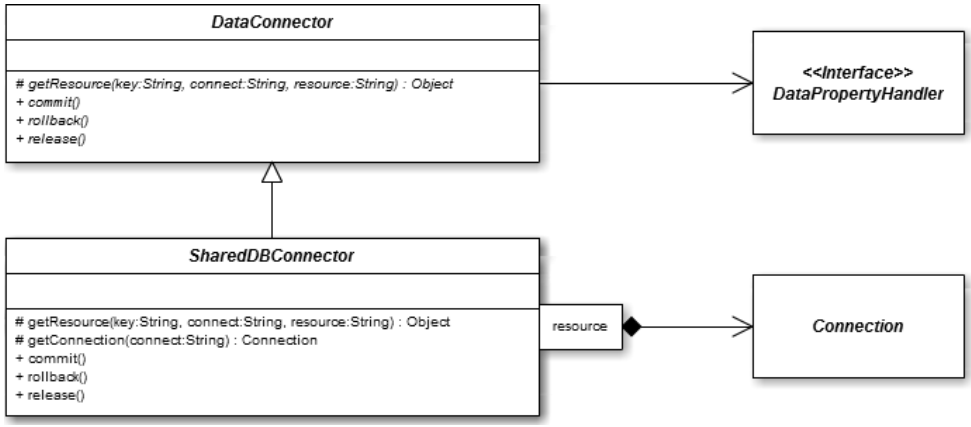


Figure 5-31 SharedDBConnector Structure

How the Connection is obtained by using SharedDBConnector is shown in [Figure 5-32 Obtaining Connection (SharedDBConnector)]. The developer who does coding by using DAO normally does not have to have deep understanding of this operation.

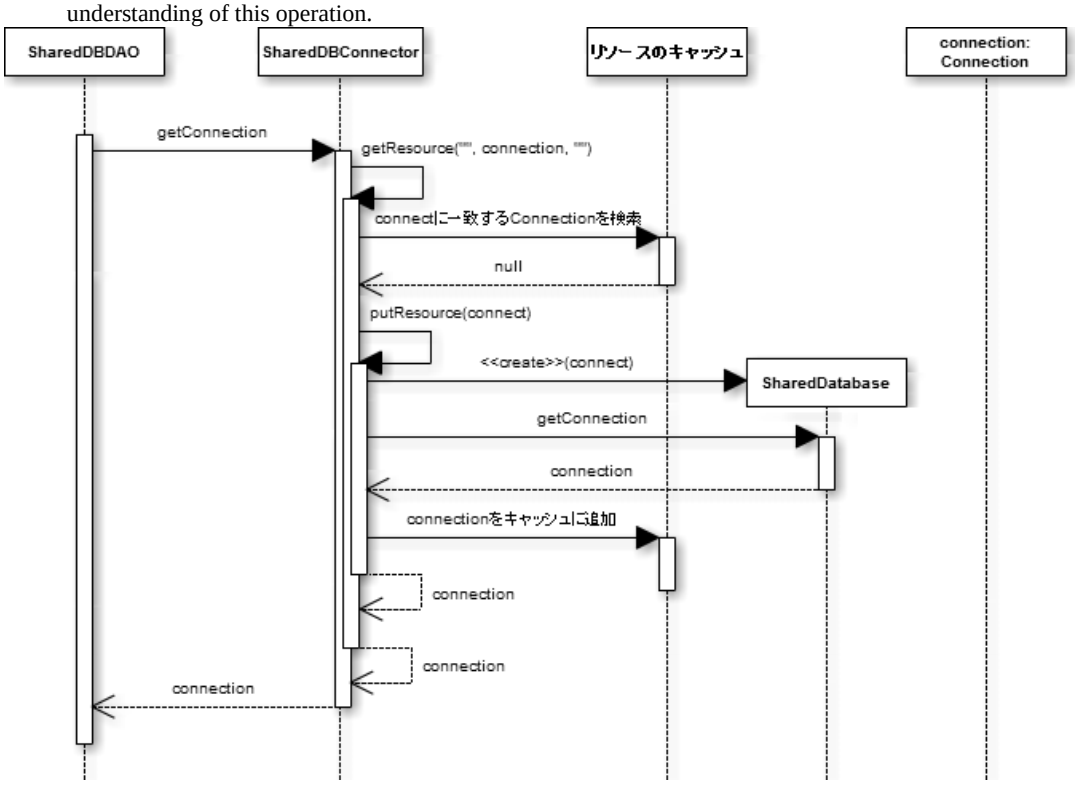


Figure 5-32 Obtaining Connection (SharedDBConnector)

Nothing is performed for commin and rollback in SharedDBConnector. How the commit or the rollback is performed for SharedDBConnector is shown in [Figure 5-33 Commit or Rollback (SharedDBConnector)].

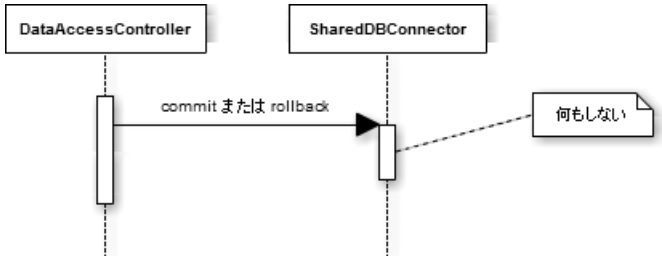


Figure 5-33 Commit or Rollback (SharedDBConnector)

5.3.3.1.6 **IntramartDBConnector**

IntramartDBConnector provides a function to connect to RDBMS that is set in intra-mart. IntramartDBConnector is mainly aimed to

be used in IntramartDBDAO. RDBMS that is set in intra-mart uses DataSource to connect, so the same merit with DataSourceConnector can be obtained.

DbsConnection is non-recommend method, and it is not operated unless you do not install compatible module. Therefore, IntramartDBDAO is operated when compatible module is installed.

TenantDBDAO or SharedDBDAO should be used.

If you use IntramarDBConnector, you need to set the setting shown in [Table 5-4 Setting Contents of IntramartDBConnector] to data property.

Table 5-4 Setting Contents of IntramartDBConnector

Item	Contents
DataConnector	jp.co.intra_mart.framework.base.data.IntramartDBConnector
Data connector name	Optional
Resource name	(None)

If you specify XmlDataPropertyHandler as DataPropertyHandler, the examples of setting contents that corresponds to [Table 5-2 Setting Contents of DataSourceConnector] are shown in [List 5-11 Setting Example of data-config.xml (IntramartDBConnector)] and [List 5-12 Setting Example of data-config-application.xml (IntramartDBConnector)]. In this example, application ID is application and data connector is myCon.

List 5-11 Setting Example of data-config.xml (IntramartDBConnector)

```
・・・
<connector>
   <connector-name>intra_mart_db</connector-name>
<connector-class>
jp.co.intra_mart.framework.base.data.IntramartDBConnector
</connector-class>
  </connector>
・・・
```

List 5-12 Setting Example of data-config-application.xml (IntramartDBConnector)

```
・・・
<dao-group>
  <dao-key>key</dao-key>
  <dao>
<dao-class>・・・</dao-class>
    <connector-name>intra_mart_db</connector-name>
  </dao>
</dao-group>
・・・
```

IntramartDBConnector structure is shown in [Figure 5-34 IntramartDBConnector Structure].

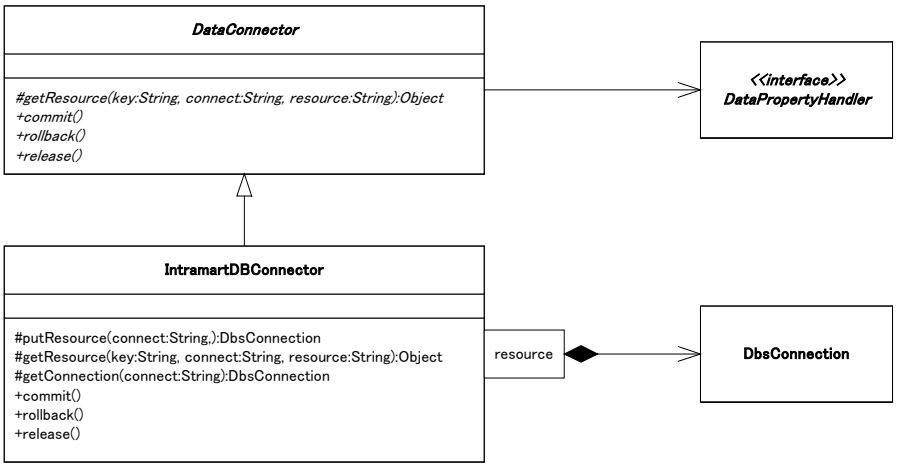


Figure 5-34 IntramartDBConnector Structure

How the DbsConnection is obtained by using IntramartDBConnector is shown in [Figure 5-35 Obtaining Connection (IntramartDBConnector]. The developer who does coding by using DAO normally does not have to have deep understanding of this operation.

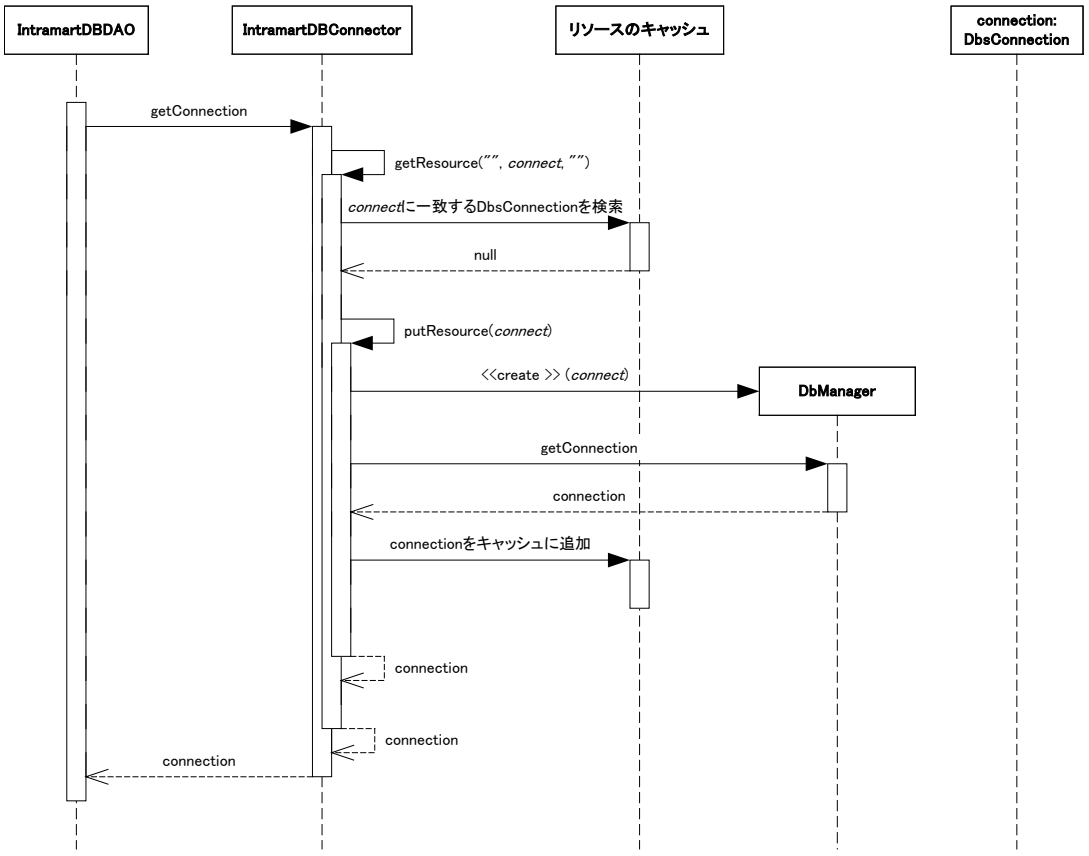


Figure 5-35 Obtaining Connection (IntramartDBConnector)

How the commit or the rollback is performed for IntramartDBConnector is shown in [Figure 5-36 Commit or Rollback (IntramartDBConnector)].

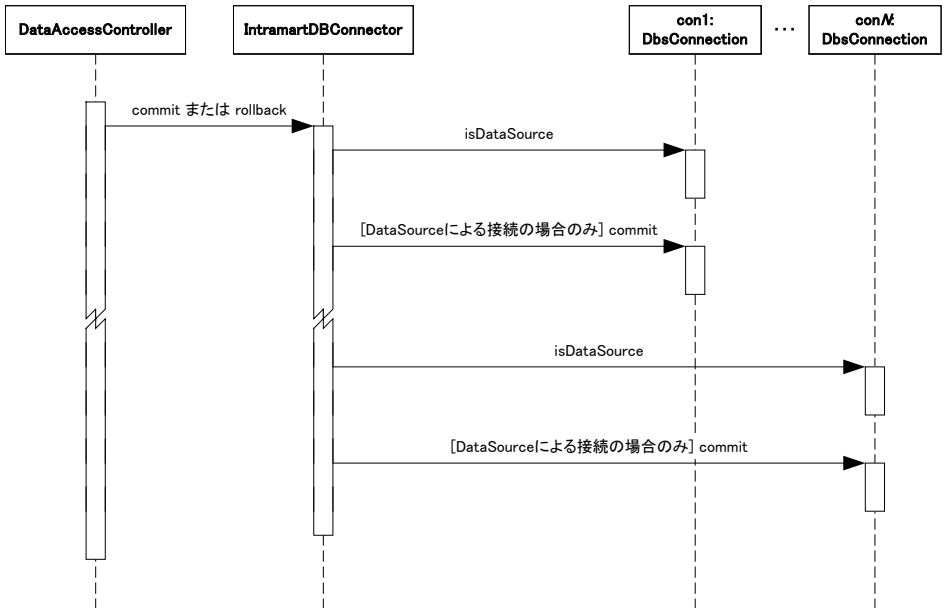


Figure 5-36 Commit or Rollback (IntramartDBConnector)

5.3.3.1.7 **IntramartStorageConnector**

IntramartStorageConnector provides a function to connect to Public Storage of intra-mart. IntramartStorageConnector is mainly aimed to be used in IntramartStorageDAO.

If you use IntramartStorageConnector, you need to set the setting shown in [Table 5-5 Setting Contents of IntramartStorageConnector] to data property.

Table 5-5 Setting Contents of IntramartStorageConnector

Item	Contents
DataConnector	jp.co.intra_mart.framework.base.data.IntramartStorageConnector
Data connector name	Optional
Resource name	(None)

If you set DefaultDataPropertyHandler as DataPropertyHandler, the examples of setting contents that corresponds として to [Table 5-2 Setting Contents of DataSourceConnector] are shown in [List 5-11 Setting Example of data-config.xml (IntramartDBConnector)] and [List 5-12 Setting Example of data-config-application.xml (IntramartDBConnector)]. Here, application ID is application and data connector name is myCon.

List 5-13 Setting Example of DataConfig.properties (IntramartStorageConnector)

```
…
<connector>
    <connector-name>intra_mart_storage</connector-name>
<connector-class>
jp.co.intra_mart.framework.base.data. IntramartStorageConnector
</connector-class>
  </connector>
…
```

List 5-14 Setting Example of DataConfig_application.properties (IntramartStorageConnector)

```
・・・
<dao-group>
  <dao-key>key</dao-key>
  <dao>
<dao-class>・・・</dao-class>
    <connector-name>intra_mart_storage</connector-name>
  </dao>
</dao-group>
・・・
```

IntramartFileServerConnector structure is shown in [Figure 5-37 IntramartStorageConnector Structure].

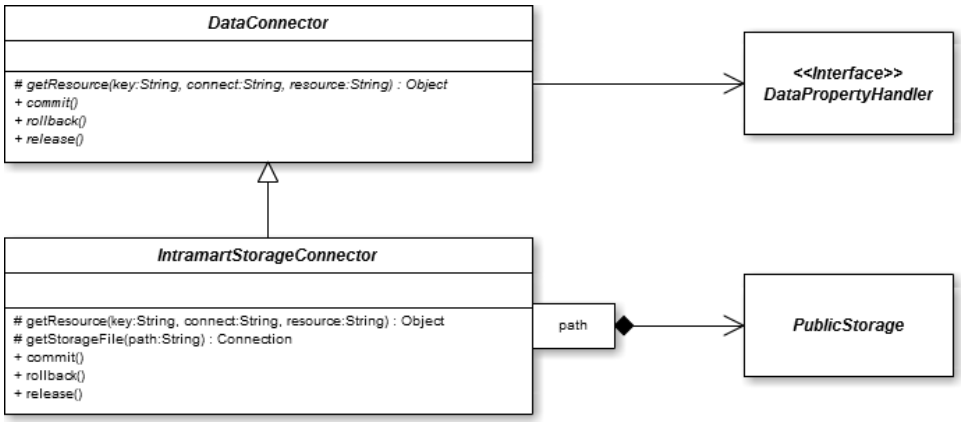


Figure 5-37 IntramartStorageConnector Structure

How the PublicStorage is obtained by using IntramartStorageConnector is shown in [Figure 5-38 Obtaining PublicStorage]. The developer who does coding by using DAO normally does not have to have deep understanding of this operation.

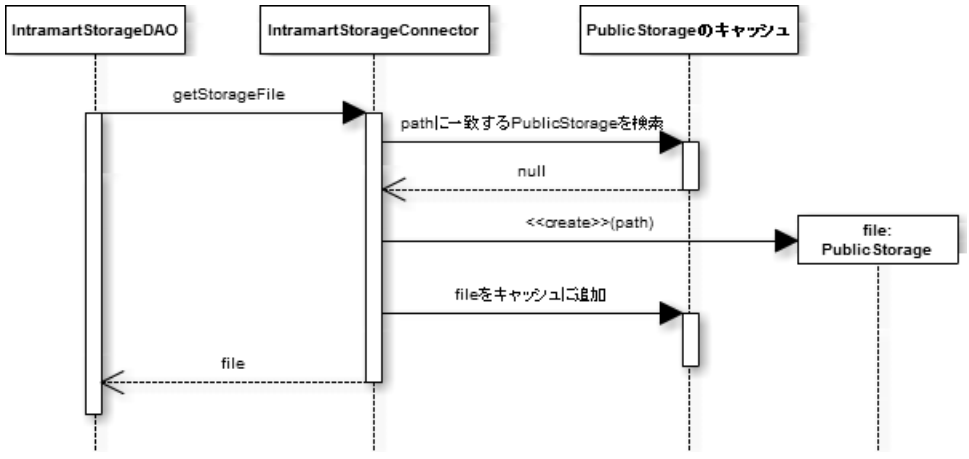


Figure 5-38 Obtaining PublicStorage

Nothing is performed for the commit or for the rollback in IntramartStorageConnector. How the commit or the rollback is performed is shown in [Figure 5-39 Commit or Rollback (IntramartStorageConnector)].

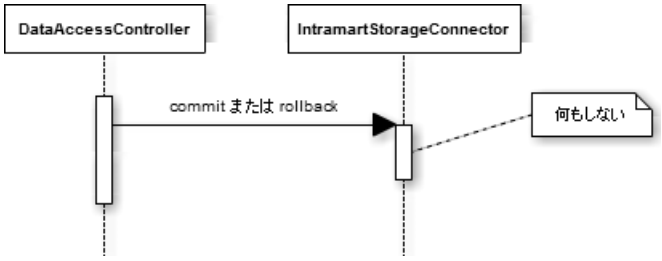


Figure 5-39 Commit or Rollback (IntramartStorageConnector)

5.3.3.2 Original DataConnector

If you access to the resource that is not provided in IM-JavaEE Framework, DataConnector should be originally created. If you create DataConnector originally, you need to meet the following requirements.

- jp.co.intra_mart.framework.base.data.DataConnector class is inherited.
- public default constructor (constructor with no argument) is defined.
- Following methods are implemented for the each appropriate operation.
  - getResource
  - commit
  - rollback
  - release

5.4 Property related to the Data Framework

In data framework of IM-JavaEE Framework, various properties can be set at outside. Data property is obtained from the class that implements jp.co.intra_mart.framework.base.data.DataPropertyHandler interface. IM-JavaEE Framework provides multiple implementation classes that implement this interface as standard (refer to [Figure 5-40 DataPropertyHandler]). Setting method of data property is not specially defined in IM-JavaEE Framework and it depends on the class that implements the interface described above.

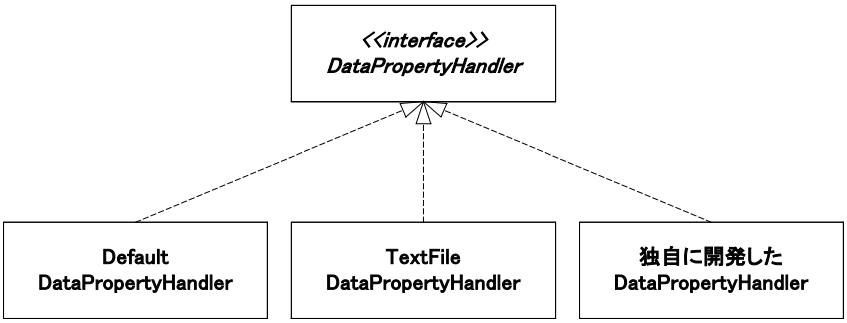


Figure 5-40 DataPropertyHandler

5.4.1 Obtaining Property related to the Data Framework

Property related to data framework is obtained from DataPropertyHandler. DataPropertyHandler can be obtained by getDataPropertyHandler method of jp.co.intra_mart.framework.data.DataManager. DataPropertyHandler must be obtained through this method and the developer cannot generate the implementation class of this DataPropertyHandler explicitly him/herself (generation by new or generation that uses newInstance of java.lang.Class or reflection).

Procedures for obtaining DataPropertyHandler and property are shown in [Figure 5-41 Obtaining DataPropertyHandler].

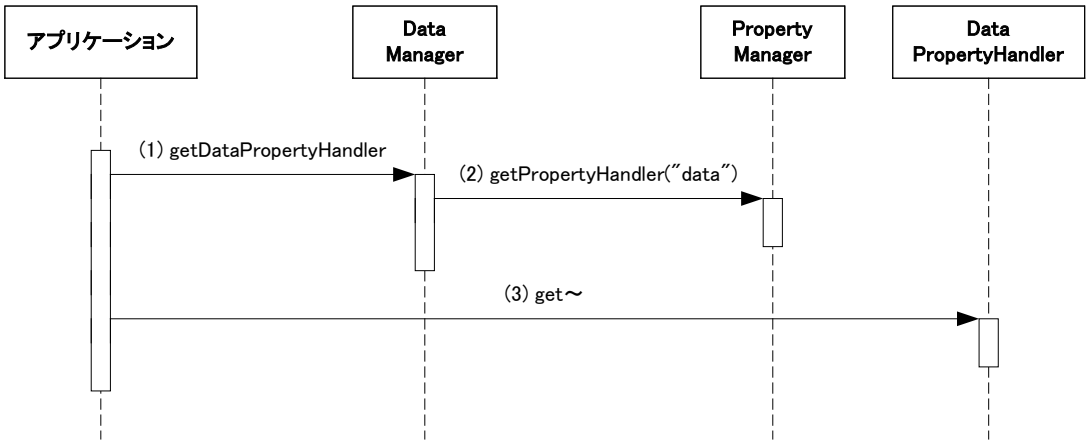


Figure 5-41 Obtaining DataPropertyHandler

1. It obtains DataPropertyHandler from DataManager.
2. In DataManager, it obtains DataPropertyHandler from PropertyManager, and returns it to the application. (this part is done in the DataManager, so the developer does not have to consider about it.)
3. It obtains each property by using DataPropertyHandler.

5.4.2 DataPropertyHandler provided as Standard

5.4.2.1 DefautlDataPropertyHandler

It is provided as jp.co.intra_mart.framework.base.data.DefaultDataPropertyHandler.

Property setting is done in resource file. The contents of resource file is set in the format of [property name=property value]. Characters that can be used are according to java.util.ResourceBundle. This resource file should be placed in a class path that can be obtained from application to be used. Please refer to API list for the detail of file name of resource file or property name to be set.

5.4.2.2 TextFileDataPropertyHandler

It is provided as jp.co.intra_mart.framework.base.data.TextFileDataPropertyHandler.

It uses the resource file that has the same format with DefaultDataPropertyHandler, but following points are different.

- No need to pass a class path.
- If the location can be accessed from the application, it can be placed at any location in the file system.
- Resource file can be re-loaded without stopping application according to the setting.

Please refer to API list for the detail of file name of resource or property name to be set.

5.4.2.3 XmlDataPropertyHandler

It is provided as jp.co.intra_mart.framework.base.data.XmlDataPropertyHandler.

Property setting is done in XML format. It should be placed in the class path which can be obtained by the application. Unique ID and Java package path are recognized as application ID. For example, if application ID is "foo.bar.example", it is placed as "foo/bar/data-config-example.xml" in class path.

XmlDataPropertyHandler supports dynamic loading.

Please refer to API for details.

5.4.3 Original DataPropertyHandler

If the developer creates DataPropertyHandler originally, the developer should meet the following requirements.

- jp.co.intra_mart.framework.base.data.DataPropertyHandler interface is implemented.
- public default constructor (constructor with no argument) is defined.
- Appropriate value must be returned to all methods (refer to [5.4.4 Property Contents])
- If isDynamic() method returns false, the value of method that obtains property will not change unless application server is restarted.

5.4.4 Property Contents

Setting method of property related to data is depending on the type of DataPropertyHandler that is to be used when operation, but the concept is the same.

Contents of property related to the data are described as below.

5.4.4.1 Common

5.4.4.1.1 Dinamic Loading

It can be obtained by isDynamic() method.

If the return value of this method is true, each property obtaining method (get~method) defined in this interface should load setting information initially everytime as the implementation. If it is false, each property obtaining method can cache the value to be obtained internally considering the performance.

5.4.4.1.2 DataConnector

It can be obtained by getConnectorClassName(String connectorName) method.

It sets the DataConnector class name that corresponds to the data connector name. If it is not set, it returns DataPropertyException. Logic name of DataConnector which was obtained in [5.4.4.2.2 Data Connector Name] or [5.4.4.2.3 Data Connecter Name (connection name specification)] will be specified for the argument of this method. The class to be specified here needs to implement jp.co.intra_mart.framework.base.data.DataConnector interface.

5.4.4.1.3 Resource Name

It can be obtained by getConnectorResource(String connectorName) method.

It sets the resource name that corresponds to the data connector name. If there is no corresponding resource name, it returns null.

5.4.4.1.4 Resource Parameter

It can be obtained by getResourceParams(String name) method.

It sets the parameter that corresponds to the resource name. If there is no corresponding parameter, it returns an array with the size 0.

5.4.4.2 Application Individual

5.4.4.2.1 DAO

It can be obtained by getDAOName(String application, String key, String connect) method.

It sets DAO class name that corresponds to the application ID, the key, and the connection name. If it is not set, the search result of DAO when the connection name is not specified will be the same.

The class to be specified here needs to implement jp.co.intra_mart.framework.base.data.DAO interface.

5.4.4.2.2 **Data Connector Name**

It can be obtained by getConnectorName(String application, String key) method.

It sets the logic name of DataConnection that corresponds to the application ID and the key. If it is not set, it returns null.

5.4.4.2.3 **Data Connecter Name (connection name specification)**

It can be obtained by getConnectorName(String application, String key, String connect) method.

It sets the logic name of DataConnection that corresponds to the application ID, the key, and the connection name. If it is not set, it returns the value to be obtained in [5.4.4.2.2 Data Connector Name].

5.4.5 Relationship between DataConnector and Resource

The relationship among DAO, DataConnector and resource on the parameter setting is shown in [Figure 5-42 Relationship between DataConnector and Resource].

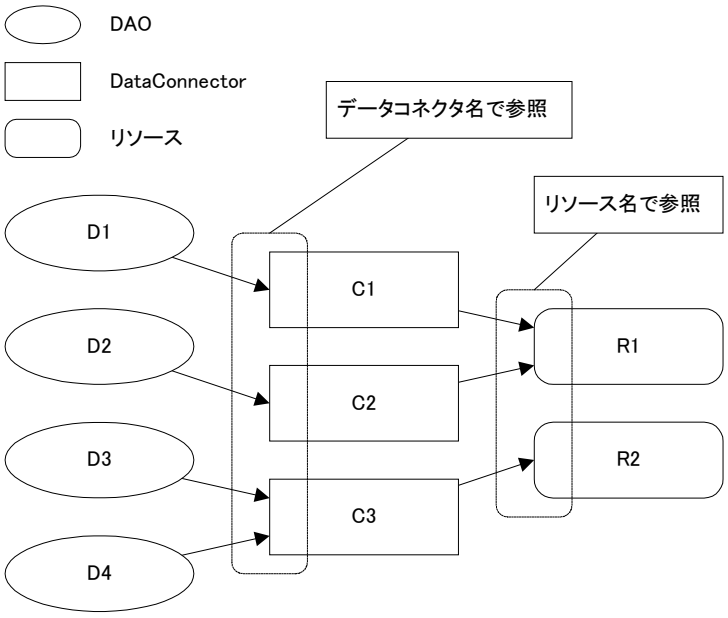


Figure 5-42 Relationship between DataConnector and Resource

Relationship among DAO, DataConnector and resource is described as below.

- One DAO refers to one DataConnector.
- One DataConnector is referred to by multiple DAO.
- One DataConnector refers to one resource.
- One resource is referred to by multiple DataConnector.

If you understand these relationships, you can minimize the setting of DataConnection or resource without copiying the same items.

5.5 Transaction

If you use data framework of IM-JavaEE Framework, you have to be aware of some points about transaction management.

5.5.1 Transaction Types

There are several methods to manage the transaction when accessing to EIS layer.

- Transaction management by UserTransaction (refer to Java Transaction API：JTA[7])
- Transaction management by DataAccessController
- Combination use of above transaction management

For transaction, it is recommended to use UserTransaction only. Transaction management by DataAccessController is simple and it does not meet high-level transaction requirement such as 2 phase commit.

5.5.1.1 Transaction Management by UserTransaction

How the transaction management by UserTransaction works is shown in [Figure 5-43 Transaction Management by UserTransaction]. In this case, please note that UserTransaction does not do commit or rollbacl for DataConnector related to DAO, which was obtained from DataAccesController.

If you do transaction management in UserTransaction, DataConnector should support the transaction of UserTransaction. In this case, implemetation of following methods in DataConnector should not perform anything on transaction process.

- commit
- rollback

In DataConnector provided in IM-JavaEE Framework, followings are corresponded to above conditions.

- jp.co.intra_mart.framework.base.data.DataSourceConnector
- jp.co.intra_mart.framework.base.data.TenantDBConnector
- jp.co.intra_mart.framework.base.data.SharedDBConnector

If you use above DataConnector, data resource to be used should allow transaction process.

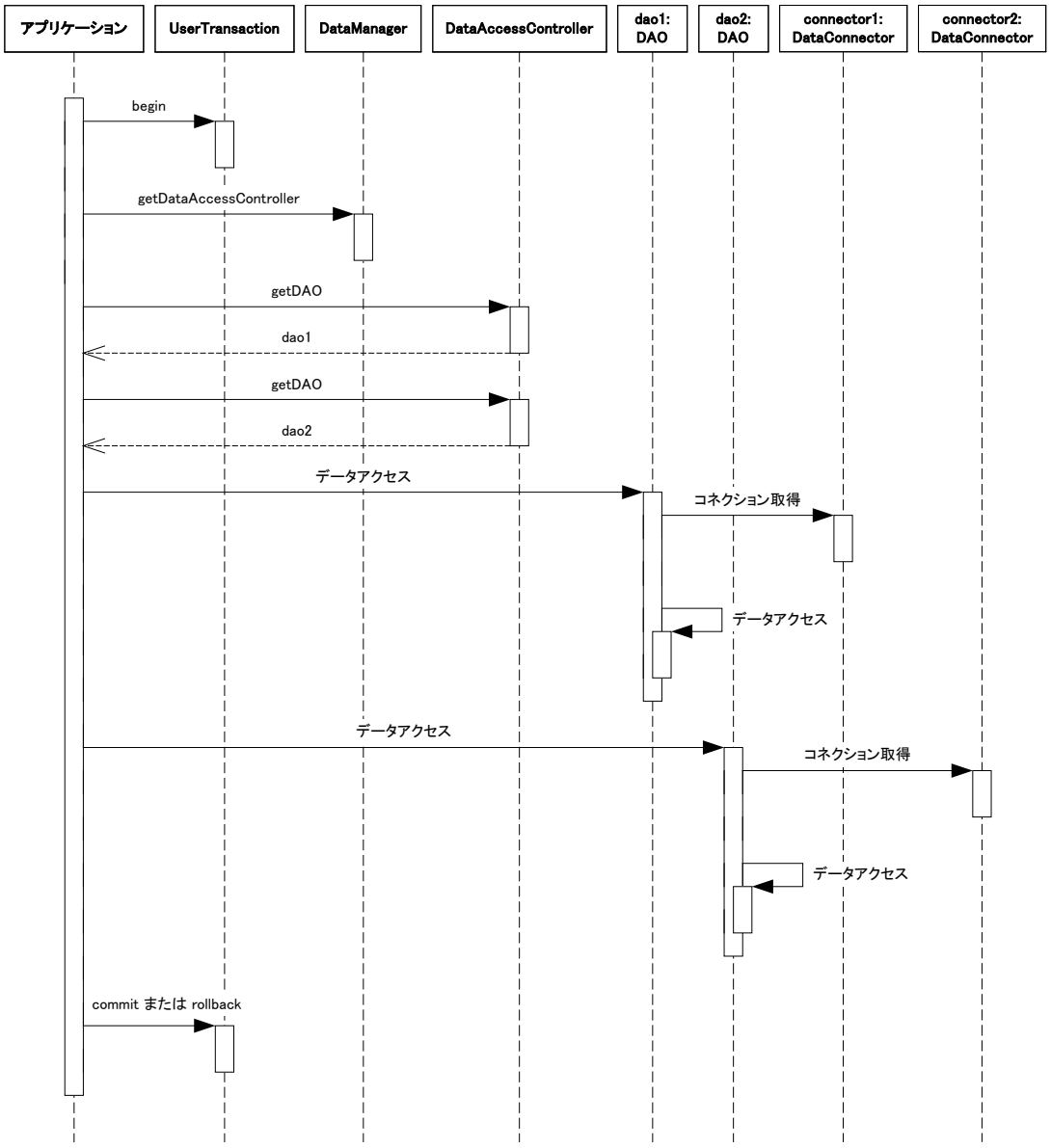


Figure 5-43 Transaction Management by UserTransaction

5.5.1.2 Transaction Management by DataAccessController

How the transaction management by DataAccessController works is shown in [Figure 5-44 Transaction Management by DataAccessController]. In this case, please note that DataAccesController does commit or rollback for each DataConnector. This indicates that commits may succeed for multiple DataConnectors and may fail for multiple DataConnectors. Thus, this transaction management method is not recommended.

If transaction management is made on DataAccessController, DataConnector should implements following methods.

- commit
- rollback

In DataConnector provided in IM-JavaEE Framework, following matches to the above conditions.

- jp.co.intra_mart.framework.base.data.JDBCConnector

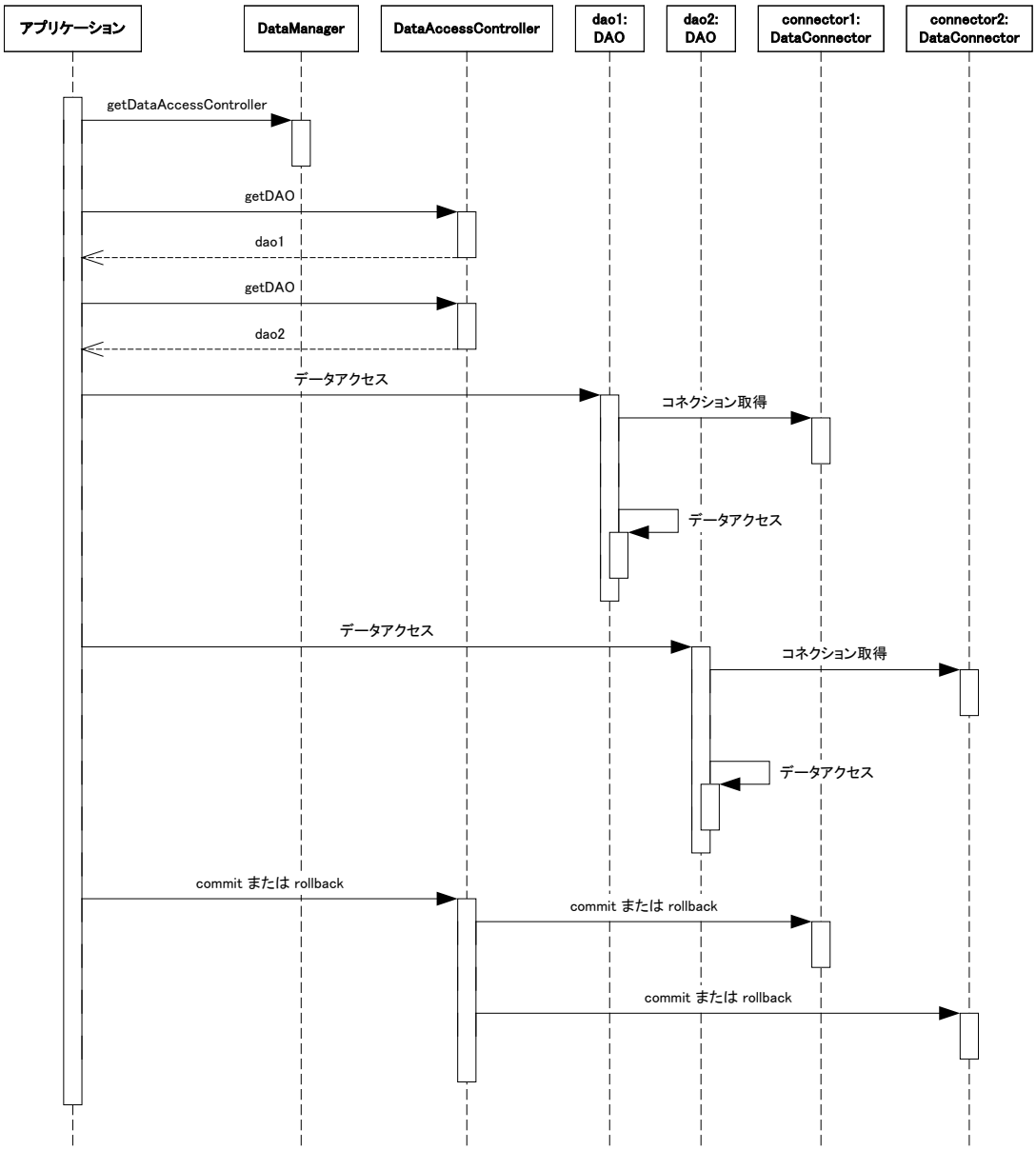


Figure 5-44 Transaction Management by DataAccessController

5.5.1.3 Transaction Management by Combination Use

If DataConnector that corresponds to UserTransaction and DataAccessController both exist, the transaction management should be performed by using both. This is depicked in [Figure 5-45 Transaction Management by Combination Use].

This case also has a deficiencies described in [5.5.1.2 Transaction Management by DataAccessController], so it is not recommended.

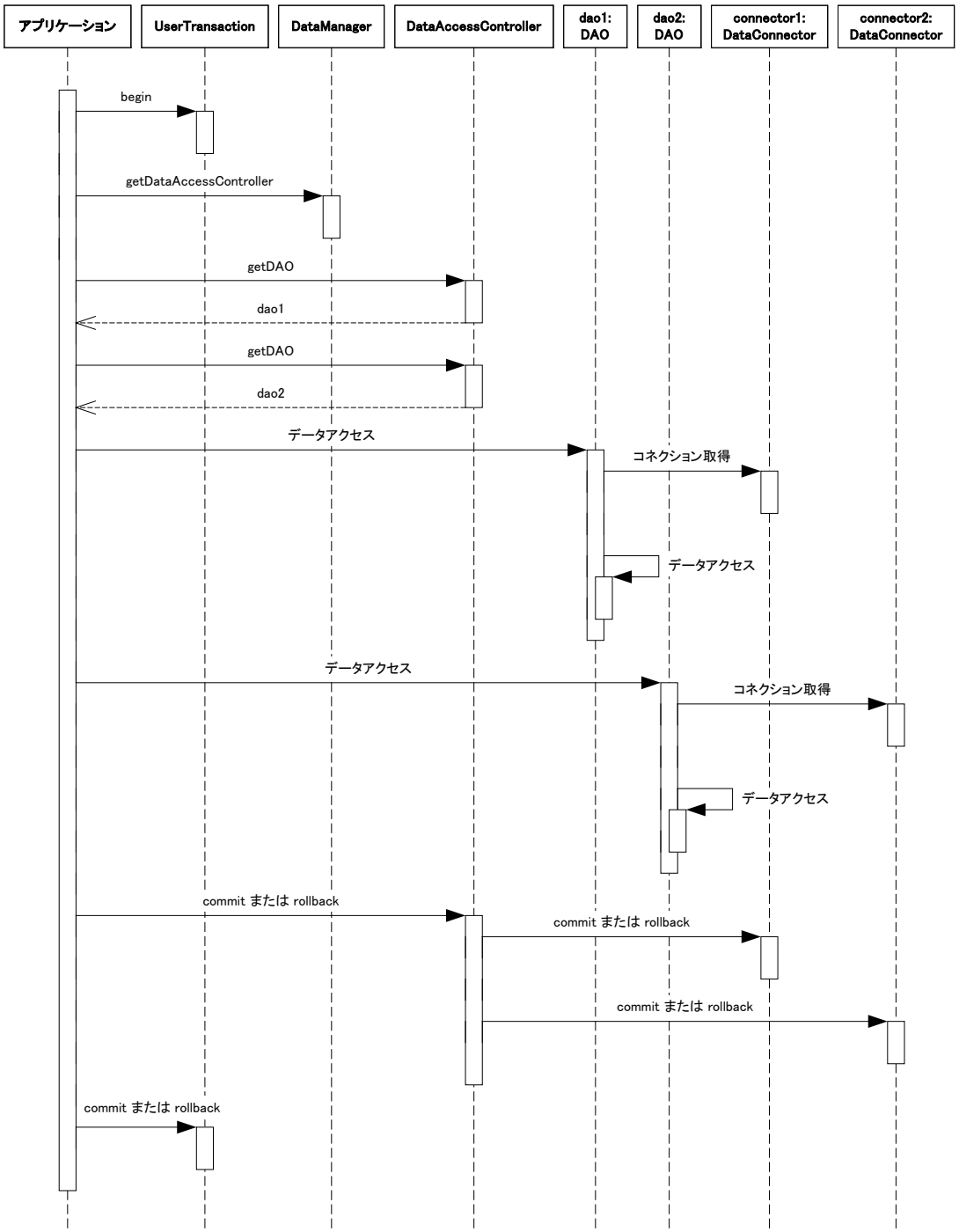


Figure 5-45 Transaction Management by Combination Use

5.5.2 Transaction Examples

Here, several examples that show how the transaction is managed are shown. Here, it is assumed that all DAOs access to the relational database, and the database supports 2 phase commit.

5.5.2.1 Single DAO + Single Connection（JDBCConnector）

Process contents when accessing database using JDBCConnector is shown in [Figure 5-46 Single DAO, Single Connection].

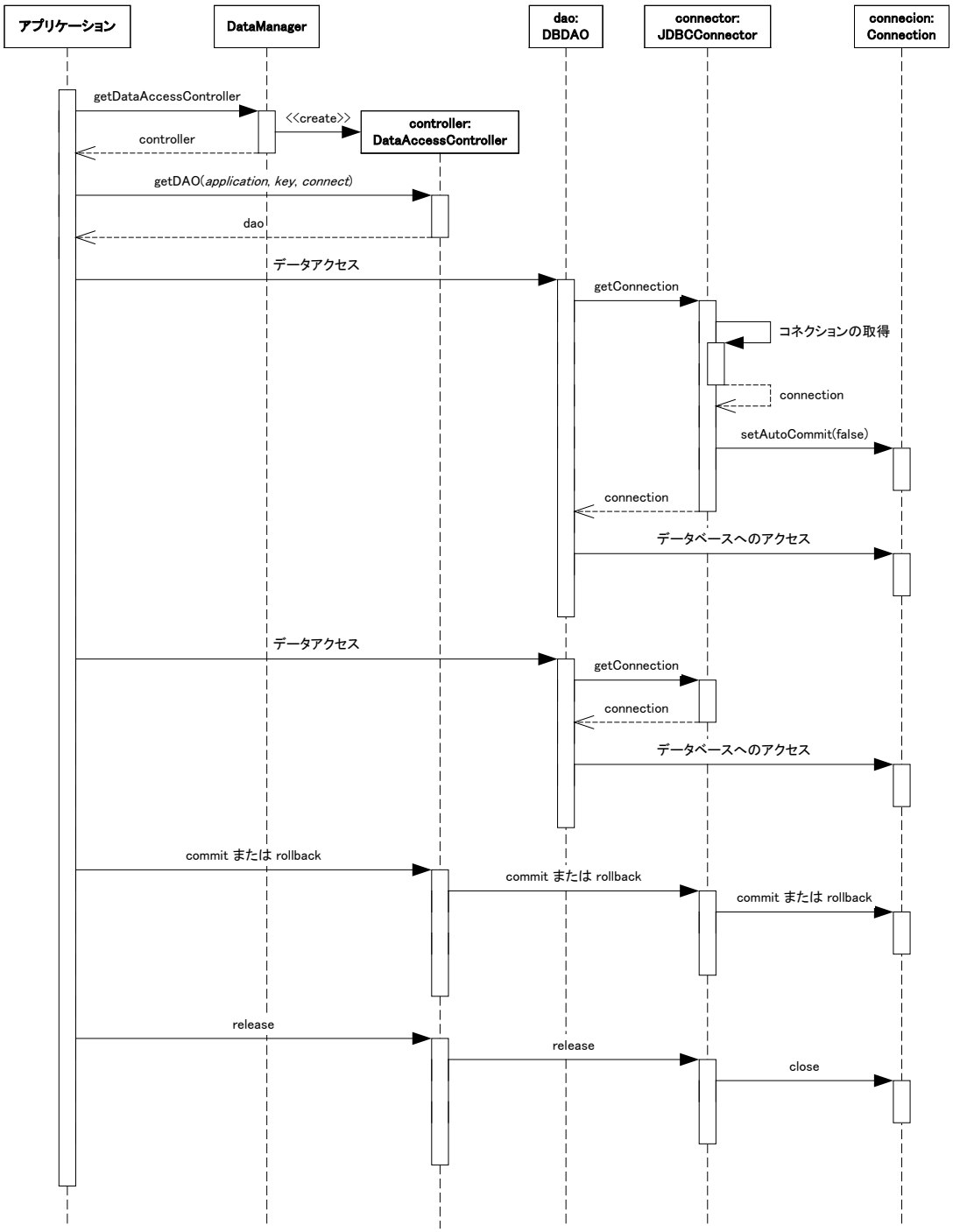


Figure 5-46 Single DAO, Single Connection

Sample code that corresponds to the contents shown in [Figure 5-46 Single DAO, Single Connection] is provided in [List 5-15 Transaction Process (Single DAO ＋ JDBCConnector)]. In [List 5-15 Transaction Process (Single DAO ＋ JDBCConnector)], some exception processes are ommited to simplify the code.

List 5-15 Transaction Process (Single DAO ＋ JDBCConnector)

```
// Obtaining DataAccessController
DataManager dm = DataManager.getDataManager();
DataAccessController dac = dm.getDataAccessController();

// Data access process
try {

    // dao の取得とデータアクセス
    MyDAOIF dao = (MyDAOIF)dac.getDAO("app", "key", "con");
    dao.access1(...);
    dao.access2(...);

    dac.commit();    // DataAccessController commit
} catch (Exception e) {

    try {
        dac.rollback();
    } catch (Exception ex) {
    }

} finally {

    // Resource releasing
    dac.release();

}
```

5.5.2.2 Multiple DAO + Single Connection（JDBCConnector）

Process contents when multiple DAOs access to the database using JDBCConnector is shown in [Figure 5-47 Multiple DAO, Single Connection].

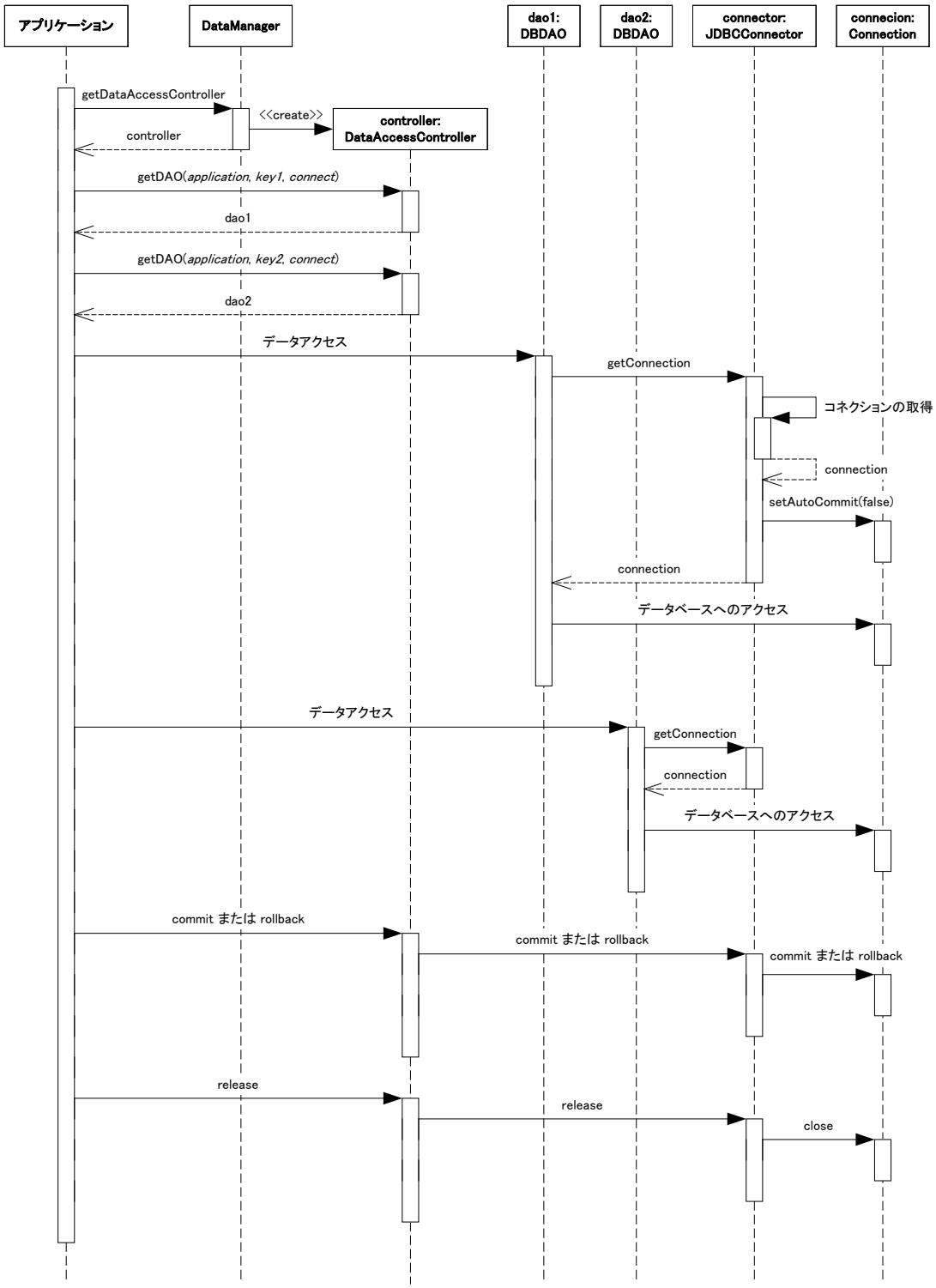


Figure 5-47 Multiple DAO, Single Connection

Sample code that corresponds to the contents shown in [Figure 5-47 Multiple DAO, Single Connection] is provided in [List 5-16 Transaction Process (Multiple DAO ＋ JDBCConnector]. In [List 5-16 Transaction Process (Multiple DAO ＋ JDBCConnector], some exception processes are omitted to simplify the code.

List 5-16 Transaction Process (Multiple DAO ＋ JDBCConnector)

```
// Obtaining DataAccessController
DataManager dm = DataManager.getDataManager();
DataAccessController dac = dm.getDataAccessController();

// Data access process
try {

    // Obtaining dao1 and data access
    MyDAO1IF dao1 = (MyDAO1IF)dac.getDAO("app", "key1", "con");
    dao1.access(...);

    // Obtaining dao2 and data access
    MyDAO2IF dao2 = (MyDAO2IF)dac.getDAO("app", "key2", "con");
    dao2.access(...);

    dac.commit();    // DataAccessController commit
} catch (Exception e) {

    try {
        dac.rollback();
    } catch (Exception ex) {
    }

} finally {

    // Resource releasing
    dac.release();

}
```

5.5.2.3 Multiple DAO + Multiple Connection (JDBCConnector)

Process contents when multiple DAOs access to database using multiple JDBCConnector is shown in [Figure 5-48 Multiple DAO, Multiple Connections].

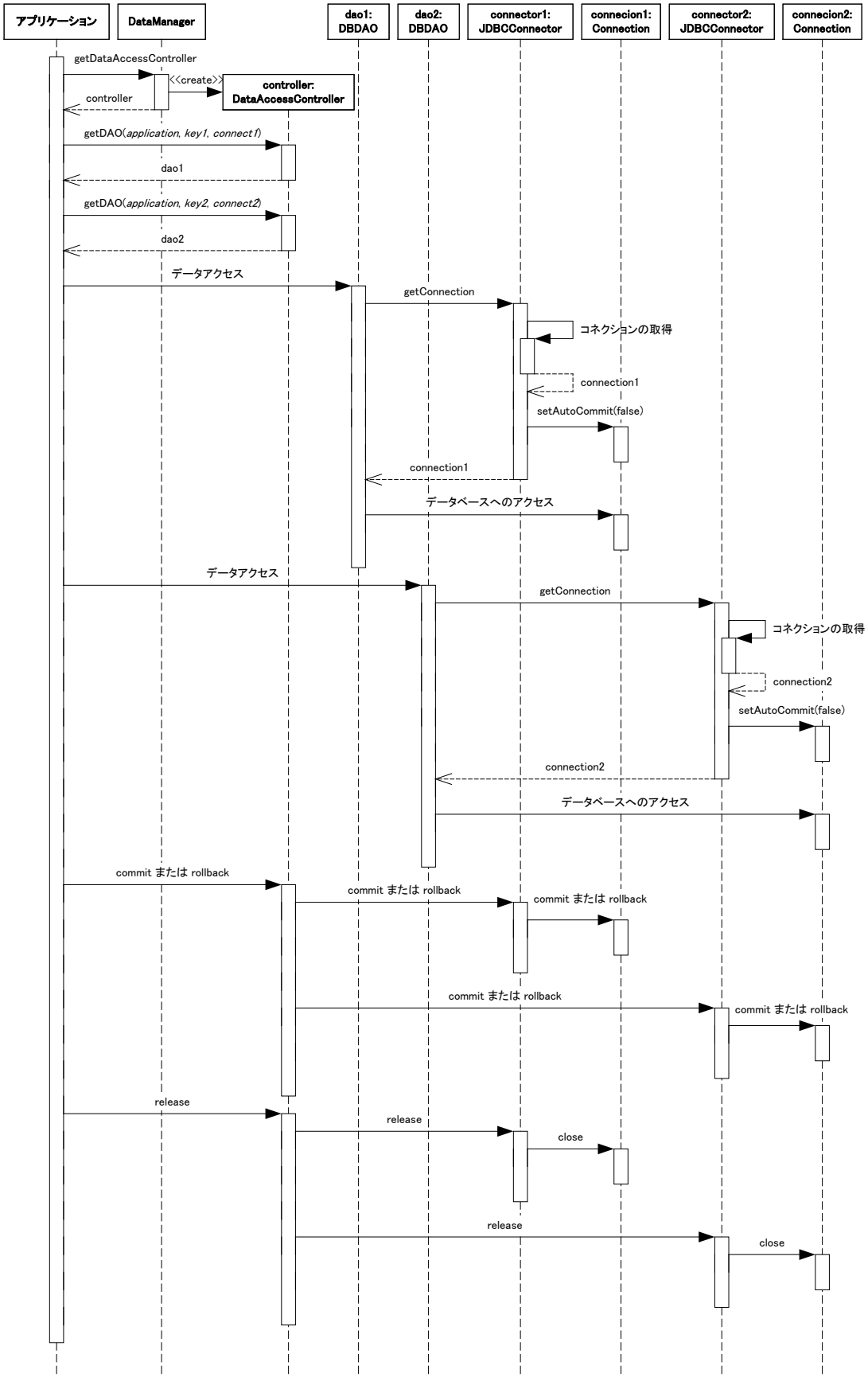


Figure 5-48 Multiple DAO, Multiple Connections

Sample code that corresponds to the contents shown in [Figure 5-48 Multiple DAO, Multiple Connections] is

provided in [List 5-17 Transaction Process (Multiple DAO ＋ Multiple JDBCConnector)]. In [List 5-17 Transaction Process (Multiple DAO ＋ Multiple JDBCConnector)], some exception processes are omitted to simplify the code.

List 5-17 Transaction Process (Multiple DAO ＋ Multiple JDBCConnector)

```
// obtaining DataAccessController
DataManager dm = DataManager.getDataManager();
DataAccessController dac = dm.getDataAccessController();

// data access process
try {

    // obtaining dao1 and data access
    MyDAO1IF dao1 = (MyDAO1IF)dac.getDAO("app", "key1", "con");
    dao1.access(...);

    //obtaining dao2 and data access
    MyDAO2IF dao2 = (MyDAO2IF)dac.getDAO("app", "key2", "con");
    dao2.access(...);

    dac.commit();    // DataAccessController commit
} catch (Exception e) {

    try {
        dac.rollback();
    } catch (Exception ex) {
    }

} finally {

    // Resource releasing
    dac.release();

}
```

If you look at the code of [List 5-17 Transaction Process (Multiple DAO ＋ Multiple JDBCConnector)], you can see that it matches with [List 5-16 Transaction Process (Multiple DAO ＋ JDBCConnector]. This means that the association between DAO and DataConnector is made by property setting and is not explicitly specified in the code. This indicates that whether to use the same connection in more than 2 different DAOs or to use different connections for each can be possibly set externally by property setting, and not in the source code.

If multiple JDBCConnectors are used, care should be taken when doing the commit. JDBCConnector does not support 2 phase commit, so the commit has to be done for each connection. If the commit is successfully performed for the first connection but failed to commit for the next connection, the commit for the first time will not be back to the original status.

5.5.2.4 Multiple DAO + Multiple Connection (DataSource)

Process contents when multiple DAO access to the database using multiple DataSourceConnector is shown in [Figure 5-49 Multiple DAO, Multiple Connections (DataSource)].

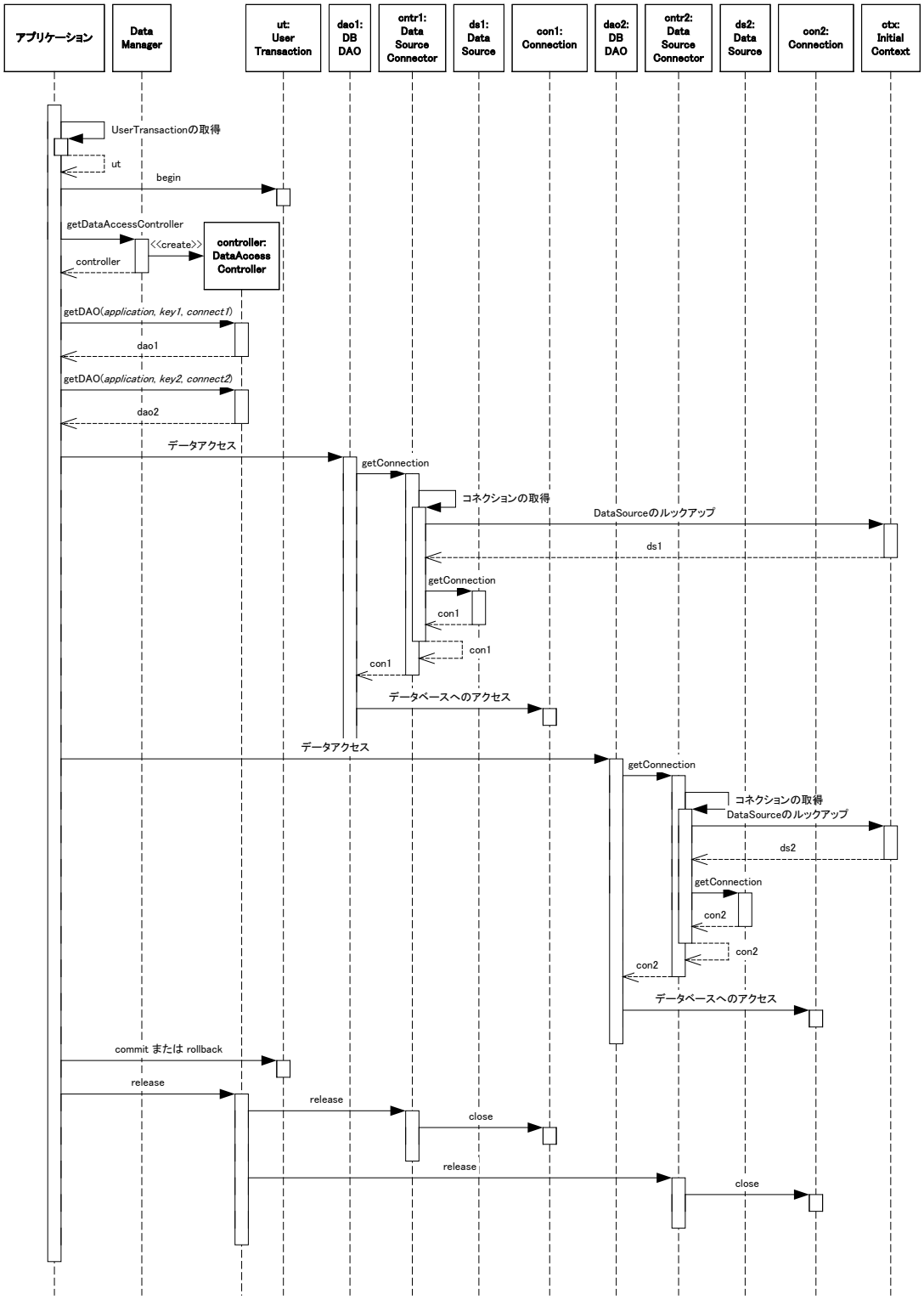


Figure 5-49 Multiple DAO, Multiple Connections (DataSource)

Sample code that corresponds to the contents shown in [Figure 5-49 Multiple DAO, Multiple Connections (DataSource)] is provided in [

List 5-18 Transaction Process (only when DataSourceConnector is used]. In [

List 5-18 Transaction Process (only when DataSourceConnector is used], some exception processes are omitted to simplify the code.

List 5-18 Transaction Process (only when DataSourceConnector is used)

```
// obtaining and starting user transaction
InitialContext ctx = new InitiralContext();
UserTransaction ut = (UserTransaction)ctx.lookup("java:comp/UserTransaction");
ut.begin();

//obtaining   DataAccessController
DataManager dm = DataManager.getDataManager();
DataAccessController dac = dm.getDataAccessController();

// data access process
try {

    //obtaining dao1 and data access
    MyDAOIF1 dao1 = (MyDAOIF1)dac.getDAO("app1", "key1", "con1");
    dao1.access(...);

    // obtaining dao2 and data access
    MyDAOIF2 dao2 = (MyDAOIF2)dac.getDAO("app2", "key2", "con2");
    dao2.access(...);

    // commit
    ut.commit();

} catch (Exception e) {

    // rollback when exception occurs
    ut.rollback();

} finally {

    // resource releasing
    dac.release();

}
```

When you look at the code of [
List 5-18 Transaction Process (only when DataSourceConnector is used], you can see that it almost matches with [List 5-16 Transaction Process (Multiple DAO ＋ JDBCConnector] or [List 5-17 Transaction Process (Multiple DAO ＋ Multiple JDBCConnector)]. Start and End of the transation is different. Transaction obtains UserTransaction and will be started explicitly from where begin method is executed and will be ended by commit method or rollback method. In the mean time, DataSourceConnector which was associated with DAO will look-up DataSource from InitialContext. Connection obtained from this DataSource is managed by UserTransaction. If all DataSource handled here support 2 phase commit, all Connection will be completely handled as one transaction (it will be either all commits succeed or all commits fail and be rollback).

5.5.2.5 Multiple DAO + Multiple Connection (combination use of DataSource and JDBC)

Process contents when multiple DAO access to the database using both DataSourceConnector and JDBCConnector is shown in [Figure 5-50 Multiple DAO, Multiple Connection (combination use of DataSource and JDBC].

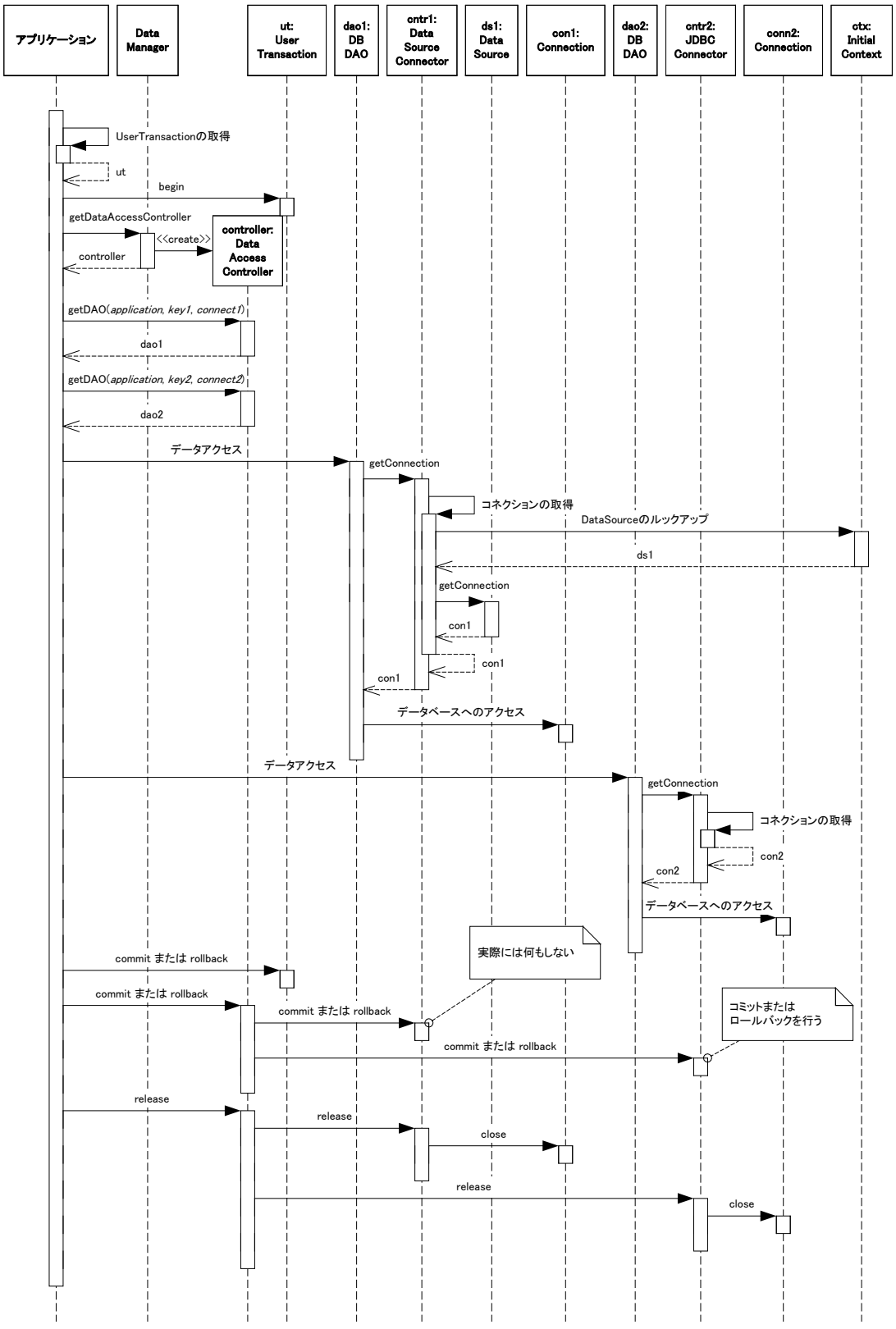


Figure 5-50 Multiple DAO, Multiple Connection (combination use of DataSource and JDBC)

Sample code that corresponds to the contents shown in [Figure 5-50 Multiple DAO, Multiple Connection (combination use of DataSource and JDBC)] is provided in [List 5-19 Transaction Process (combination use of DataSourceConnector and JDBCConnector]. In [List 5-19 Transaction Process (combination use of DataSourceConnector and JDBCConnector], some exception processes are omitted to simplify the code.

List 5-19 Transaction Process (combination use of DataSourceConnector and JDBCConnector)

```
// obtaining and starting of user transaction
InitialContext ctx = new InitiralContext();
UserTransaction ut = (UserTransaction)ctx.lookup("java:comp/UserTransaction");
ut.begin();

// obtaining DataAccessController
DataManager dm = DataManager.getDataManager();
DataAccessController dac = dm.getDataAccessController();

// data access process
try {

    try {
        // obtaining dao1 and data access
        MyDAOIF1 dao1 = (MyDAOIF1)dac.getDAO("app1", "key1", "con1");
        dao1.access(...);

        // obtaining dao2 and data access
        MyDAOIF2 dao2 = (MyDAOIF2)dac.getDAO("app2", "key2", "con2");
        dao2.access(...);

        dac.commit();    // DataAccessController commit
    } catch (Exception e) {
        try {
            dac.rollback();
        } catch (Exception ex) {
        }
        throw e;
    }
    ut.commit();      // user transaction commit

} catch (Exception e) {

    // rollback when exception occurs
    ut.rollback();

} finally {

    // resource releasing
    dac.release();

}
```

If you look at the code of [List 5-19 Transaction Process (combination use of DataSourceConnector and JDBCConnector], you can see that it almost matches with [List 5-16 Transaction Process (Multiple DAO ＋ JDBCConnector], [List 5-17 Transaction Process (Multiple DAO ＋ Multiple JDBCConnector)], and [
List 5-18 Transaction Process (only when DataSourceConnector is used]. Start and End of the transation is different. Transaction obtains UserTransaction and will be started explicitly from where begin method is executed and also started when DAO associated with JDBCConnector is obtained. Transaction by UserTransaction is ended by commit method or rollback method and the transaction of JDBCConnector is ended by commit or rollback method of DataAccessController. This indicates that 2 different transactions exist in parallel.

If 2 different transactions are executed in parallel, there could be a situation where commit succeeds for one and fails for another. In order to avoid such event, it is preferred to be use DataConnector managed in UserTransaction only.

6 Message Framework

6.1 Overview

In order to create the application that supports intarnationalized display, it is necessary to implement the display that supports each region. Message framework obtains character strings that corresponds to the region based on the locale.

6.2 Structure

6.2.1 Structure Element

Message framework is structured by followings.

- MessageManager
- MessagePropertyHandler

These relationship is shown in [Figure 6-1 Class Figure of Message Framework].

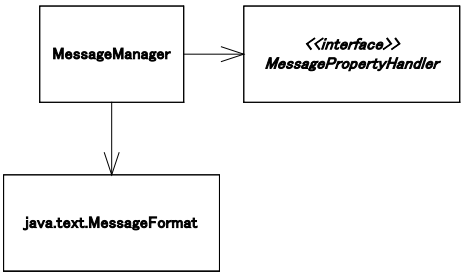


Figure 6-1 Class Figure of Message Framework

6.2.2 Message Obtaining Process

Overview when you obtain message that has region support in message framework of IM-JavaEE Framework is shown in [Figure 6-2 Overview of Message Obtaining].

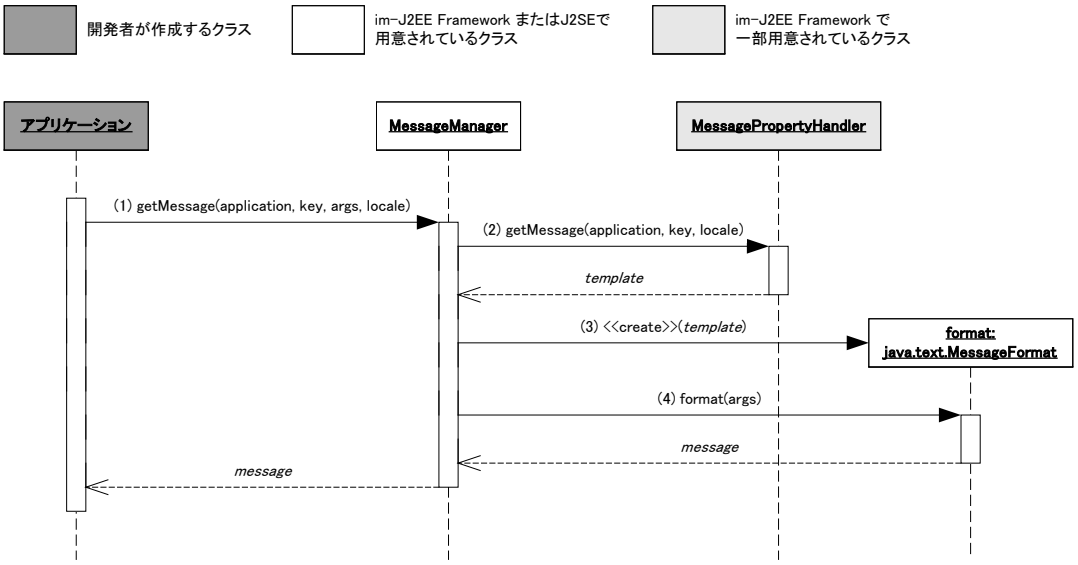


Figure 6-2 Overview of Message Obtaining

1. Application requests to obtain a message that supports application ID and message key by using getMessage method of MessageManager.
2. MessageManager obtains message prototype that supports application ID and message key by using MessagePropertyHandler.
3. MessageManager generates the instance of java.text.MessageFormat based on the message prototype.
4. MessageManager uses format method of generated MessageFormat instance, and generates the message in a format with parameter embedded

6.3 Property related to the Message

In the message framework of IM-JavaEE Framework, message prototype can be set at outside. Message property is obtained from the class that implements jp.co.intra_mart.framework.base.message.MessagePropertyHandler interface. In IM-JavaEE Framework, multiple implementation class that implements this interface is provided as standard (refer to [Figure 6-3 MessagePropertyHandler]). Setting method of message property is not specially defined in IM-JavaEE Framework, it depends on the class that implements the previous interface.

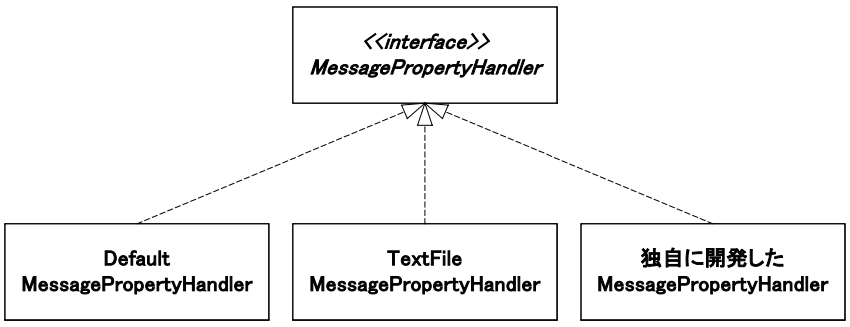


Figure 6-3 MessagePropertyHandler

The message to be obtained here (prototype) is for the specified region.

6.3.1 Obtaining Property about the Message

Property related to the message is obtained from MessagePropertyHandler. MessagePropertyHandler can be obtained by getMessagePropertyHandler method of jp.co.intra_mart.framework.base.message.MessageManager. MessagePropertyHandler must be obtained through this method and the developer should not generate the

implementation class of MessagePropertyHandler explicitly him/herself (generation by new or instance generation that uses newInstance method or reflection of java.lang.Class).

Procedure related to obtaining MessagePropertyHandler and property is shown in [Figure 4-22 Obtaining EventPropertyHandler].

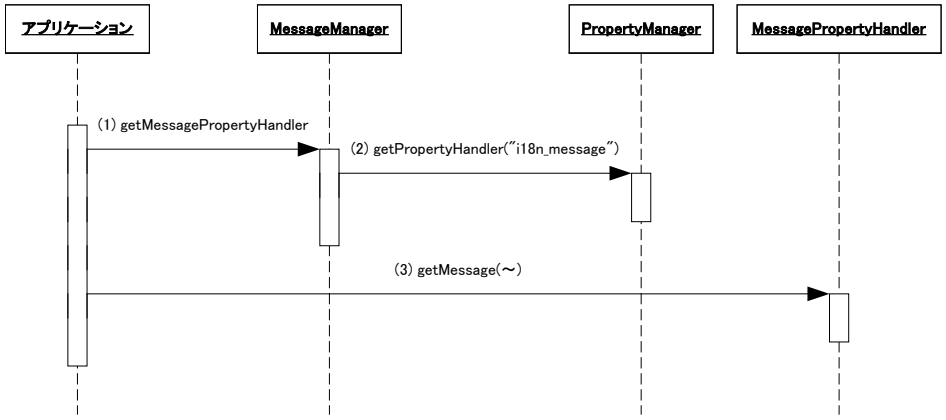


Figure 6-4 Obtaining MessagePropertyHandler

1. It obtains MessagePropertyHandler from MessageManager.
2. In MessageManager, MessagePropertyHandler is obtained from PropertyManager, and returns to the application. This part is done inside the MessageManager and the developer does not have to consider about it.
3. It obtains each property using MessagePropertyHandler.

6.3.2 MessagePropertyHandler provided as Standard

In IM-JavaEE Framework, some classes that implement jp.co.intra_mart.framework.base.message.MessagePropertyHandler are provided. The setting methods or its features are all different, so the operator can switch these if necessary.

6.3.2.1 DefaultMessagePropertyHandler

It is provided as jp.co.intra_mart.framework.base.message.DefaultMessagePropertyHandler.

Property setting is performed by resource file. Resource file contents are set in a format called [property name=property value]. Characters that can be used are according to java.util.ResourceBundle. This resource file needs to be placed in the class path that can be obtained from the application to be used. Please refer to API list for the details of file name or resource file or property name to be set.

When the message is obtained from resource file, internationalization is considered. Please refer to the document related to java.util.ResourceBundle in the API list attached to JavaTM 2 SDK, Standard Edition for details.

6.3.2.2 TextFileMessagePropertyHandler

It is provided as jp.co.intra_mart.framework.base.message.TextFileMessagePropertyHandler.

Resource file that has the same format with DefaultMessagePropertyHandler is used. The linternationalization rule is the same. Resource file that is handled in TextFileMessagePropertyHandler and DefaultMessagePropertyHandler have following different points.

- No need to pass a class path.

- If the location can be accessed from the application, it can be placed at any location in the file system.
- Resource file can be re-loaded without stopping application according to the setting.

Please refer to API list for the details of file name of resource file or the property to be set.

6.3.3 Original MessagePropertyHandler

If the developer creates MessagePropertyHandler originally, following requirements must be set.

- jp.co.intra_mart.framework.base.message.MessagePropertyHandler interface is implemented.
- public default constructor (constructor with no argument) is defined.
- Appropriate value must be returned to all methods (refer to [6.3.4 Property Contents])
- If isDynamic() method returns false, the value of method that obtains property will not change unless application server is restarted.

6.3.4 Property Contents

Setting method of property related to the message is depending on the type of MessagePropertyHandler that is to be used when operation, but the concept is the same.

Contents of property related to the message are described as below.

6.3.4.1 Common

6.3.4.1.1 Dynamic Loading

It can be obtained by isDynamic().

If the return value of this method is true, each property obtaining method (get~method) defined in this interface should load setting information initially everytime as the implementation. If it is false, each property obtaining method can cache the value to be obtained internally considering the performance.

6.3.4.2 Application Individual

6.3.4.2.1 **Message**

It can be obtained by getMessage(String application, String key) method or getMessage(String application, String key, Locale locale) method.

getMessage(String application, String key) method of MessagePropertyHandler works the same when calling getMessage(application, key, Locale.getDefault()) inside.

It sets the message prototype that supports application ID, message ID, and message key. If getMessage(String application, String key, Locale locale) method of MessagePropertyHandler is called, the setting contents will be obtained by the following search sequence, and returns the one that was first obtained.

1. Value specified by language of the specified locale (code to be obtained by getLanguage() method of locale), country (code to be obtained by getCountry() method of locale), and variant (code to be obtained by getVariant() method of locale)
2. Value specified by language of the specified locale (code to be obtained by getLanguage() method of locale), country (code to be obtained by getCountry() method of locale) Value specified by language of the specified locale (code to be obtained by getLanguage() method of locale)
3. Value specified without locale

If the value searched by above sequence is not set, it throws MessagePropertyException.

Character string to be specified in this property should have the same format with the constructor argument of java.text.MessageFormat.

7 Log Framework

7.1 Overview

Log could be a key to get internal information at development or operation. Console, file or database could be the destination for log output. It comes in a variety of formats such as CSV, XML, or database record. There could be a situation where these items cannot be fixed at development time or need to be changed during operation.

Log framework of IM-JavaEE Framework provides the method that manages log output timing, contents, output destination and its format separetely in order to solve these problems.

7.2 Structure

7.2.1 Structure Element

Log framework is structured by followings.

- LogAgent
- LogManager
- LogPropertyHandler

The relationship of these are shown in [Figure 7-1 Log Framework Structure].

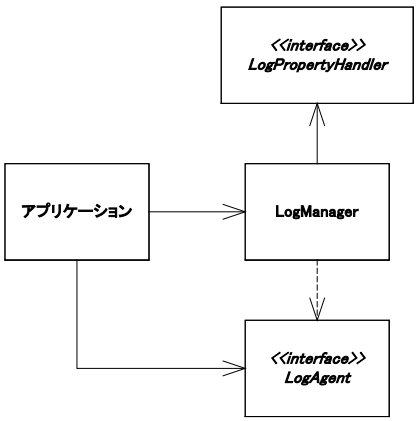


Figure 7-1 Log Framework Structure

7.2.2 Log Output Process

Overview when log is output by log framework of IM-JavaEE Framework is shown in [Figure 7-2 Log Output Process Overview].

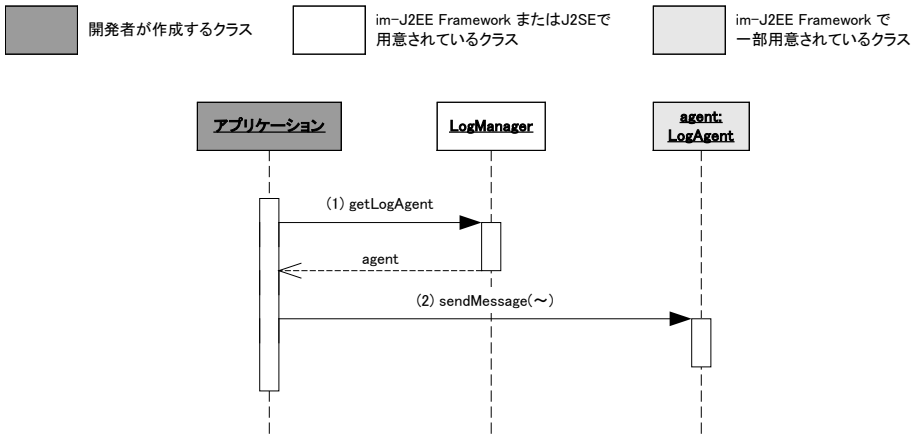


Figure 7-2 Log Output Process Overview

1. Application obtains the instance of LogAgent by using getLogAgent method of LogManager.
2. Application outputs the log by using sendMessage method of obtained LogAgent.

7.3 LogAgent

jp.co.intra_mart.faramework.system.log.LogAgent interface is the interface of object to be output. Log output destination (such as file, console, and database) or its format (such as listing of time series, CSV, and XML) is dependning on the class that implements this interface.

7.3.1 LogAgent Function

Following 2 methods are provided for LogAgent.

- sendMessage(java.lang.String category,
  java.lang.String level,
  java.lang.String message)
- sendMessage(java.lang.String category,
  java.lang.String level,
  java.lang.String message,
  java.lang.Object detail)

Argument category is used for classifying the logs. How to classify is depending on the developer who handles the logs. For example, it can be the log that is related to the executing part inside the system such as database access or business logic, or the log that is related to the business such as procurement business or customer registration task.

Argument level represents the importance of the log. Type of importance is depending on the developer who handles the logs. Debug, information, warning, and exception are listed as the example of main type of importance. In jp.co.intra_mart.framework.system.log.LogConstant. there are constant values that are intentended to repsent these (They are defined as LEVEL_DEBUG, LEVEL_INFO, LEVEL_WARNNING, LEVEL_ERROR). The developer also can define these originally.

Argument message shows the message that is actually output in the log.

Argument detail shows the detail of message to be output in the log. Since this argument is java.lang.Object, the developer of this LogAgent should handle this value appropriately. For example, if java.lang.Throwable is passed, it should displays the contents to be obtained by printStackTrace method, and in other cases, it should display the

contents to be obtained by toString method.

7.3.2 LogAgent Preparation

LogAgent is obtained by using getLogAgent of LogManager. LogManager returns LogAgent that is generated by the procedures shown in [Figure 7-3 Initialization of LogAgent] at this point.

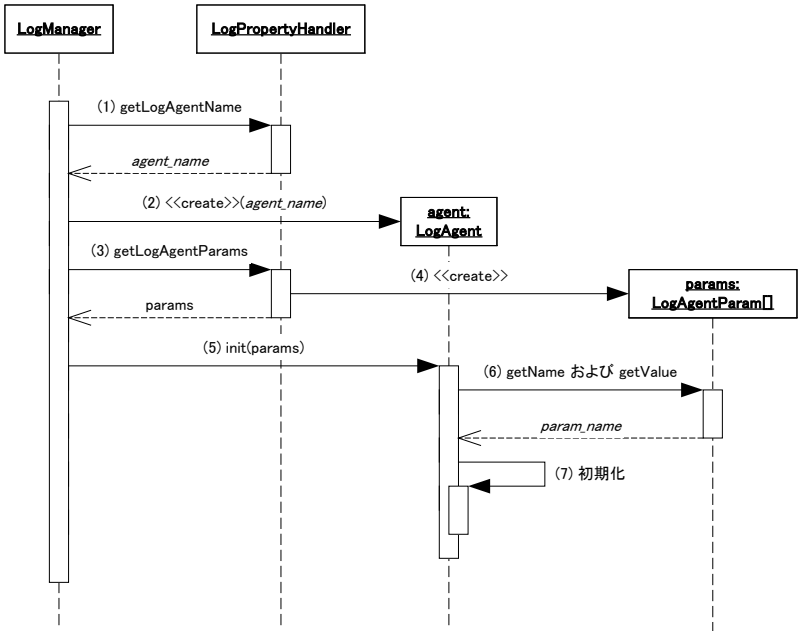


Figure 7-3 Initialization of LogAgent

1. LogManager obtains the class name of LogAgent by using getLogAgentName method of LogPropertyHandler.
2. LogManager generates the instance of LogAgent based on the obtained class name.
3. LogManager obtains initialization parameter of LogAgent by using getLogAgentParams method of LogPropertyHandler.
4. LogPropertyHandler returns the array of initialization parameter of LogAgent (LogAgentParam) based on the property setting contents.
5. LogManager initialize LogAgent by using init method of LogAgent based on the obtained initialization parameter.
6. LogAgent obtains the value of initialization parameter by using getName method and getValue method of LogAgentParam.
7. LogAgent does initialization based on the obtained initialization parameter.

If the exception is occurred somewhere in [Figure 7-3 Initialization of LogAgent], getLogAgent method of LogManager returns DefaultLogAgent as LogAgent (refer to [7.3.3.1 DefaultLogAgent]).

7.3.3 LogAgent provided as Standard

In IM-JavaEE Framework, some basic LogAgent are provided.

7.3.3.1 DefaultLogAgent

jp.co.intra_mart.framework.system.log.DefaultLogAgent is the LogAgent that outputs the log in standard output (java.lang.System.out) or standard exception (java.lang.System.err). Format to be output is described as below.

```
[category][level]message
```

Here, category, level, and message support first argument, second argument, and third argument of sendMessage

method for each. Output destination is normally standard output, but if the category is equal to LogConstant.LEVEL_ERROR, it will be output in standard exception.

sendMessage(String, String, String, Object) method is basically output in the same format. Normally, the value of fourth argument is displayed as character string which is converted by toString method, but if it is a subclass of java.lang.Throwable, the stack trace is displayed in the output destination.

7.3.3.2 IntramartLogAgent

IntramartLogAgent is the LogAgent that is specialized for log output which was set in intra-mart.

7.3.4 Original LogAgent

If the LogAgent is originally created by the developer, following requirements should be met.

- jp.co.intra_mart.framework.system.log.LogAgent interface is implemented.
- public default constructor (constructor with no argument) is defined.
- Appropriate initialization should be performed in init method.
- It should be implemented to allow appropriate log output by sendMessage method.

7.3.5 Property Contents

Property setting method related to the log is depending on the type of LogPropertyHandler to be used for operation, but the concept is the same.

Property contents related to the log are described as below.

7.3.5.1 Common

7.3.5.1.1 Log Agent Class Name

It can be obtained by getLogAgentName() method.

Implementation class of LogAgent to be used for log output is specified by the complete class name that includes package name. If nothing is set, it should be implemented to return null.

7.3.5.1.2 Log Agent Initialization Parameter

It can be obtained by getLogAgentParams() method.

It sets the parameter when you initialize LogAgent. If there is corresponding parameter, it returns the array with the size 0.

7.3.5.2 Application Individual

There is no setting for application individual in log framework.

8 PropertyHandler

8.1 Overview

Various sub frameworks of IM-JavaEE Framework have been introduced from [3 Service Framework] to [7 Log Framework]. These operations are controlled by property setting in each service framework. Setting method of this property is depending on the class that implements the interface named ~PropertyHandler. This ~PropertyHandler is not fixed, and it can be exchanged with unique class that implements ~PropertyHandler to be used in each sub framework. By this, you can dynamically change properties when there is a need to change property setting frequently. You can use PropertyHandler where property is cached in the operation where the setting change is rarely required and performance has higher priority.

Setting method of PropertyHandler in IM-JavaEE Framework is described in this chapter.

8.2 Structure

8.2.1 Structure Element

Property in IM-JavaEE Framework is structured by followings.

- PropertyManager
- PropertyHandler
- ～PropertyHandler

The relationship of these is shown in [Figure 8-1 PropertyHandler Structure].

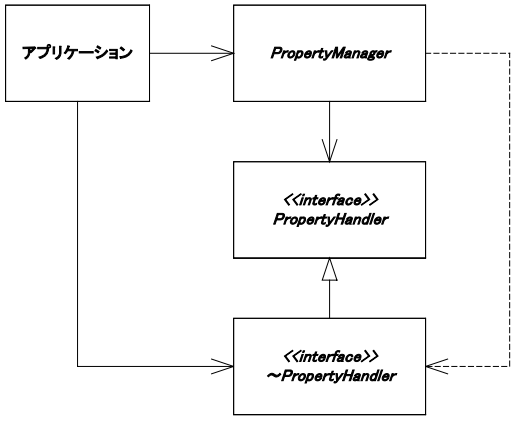


Figure 8-1 PropertyHandler Structure

8.2.1.1 PropertyManager

jp.co.intra_mart.framework.system.property.PropertyManager has role to generate/obtain PropertyHandler.

8.2.1.2 PropertyHandler

jp.co.intra_mart.framework.system.property.PropertyHandler interface is the PropertyHandler interface that is to be generated from PropertyManager. All PropertyHandler used in IM-JavaEE Framework should implement this interface.

8.2.1.3 ~PropertyHandler

It is the PropertyHandler interface that is specialized for each sub framework such as service framework or event framework of IM-JavaEE Framework.

8.2.2 Obtaining PropertyHandler

When you obtain PropertyHandler, it specifies [key] that represents types of PropertyHandler in PropertyManager. PropertyManager confirms whether PropertyHandler that corresponds to the key exists in cache, and returns the PropertyHandler if it exists. How the PropertyHandler is obtained when it is in cache is shown in [Figure 8-2 Obtaining PropertyHandler (when it exist in cache)].

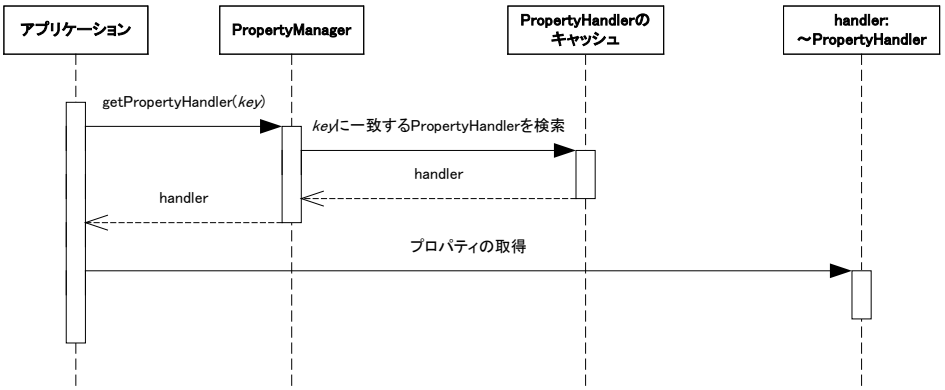


Figure 8-2 Obtaining PropertyHandler (when it exist in cache)

If PropertyHandler does not exist in cache, PropertyManager newly generates the PropertyHandler that corresponds to the key. If there is information that initializes for PropertyHandler, set it to the PropertyHandler (init method of PropertyHandler). Eventually, PropertyManager adds the initialized PropertyHandler to the cache and returns. How it is obtained is shown in [Figure 8-3 Obtaining PropertyHandler (when it does not exist in cache)].

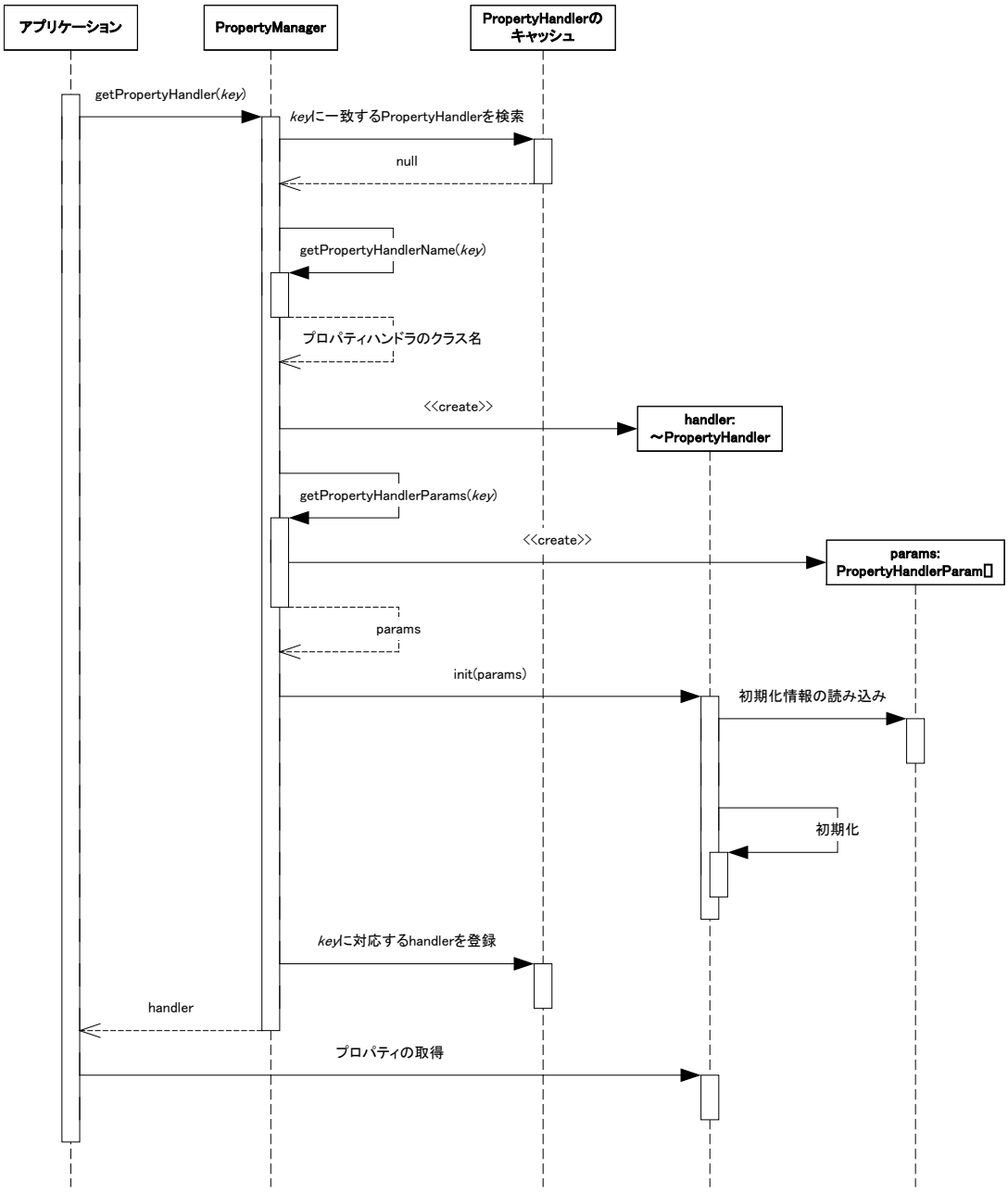


Figure 8-3 Obtaining PropertyHandler (when it does not exist in cache)

8.2.3 Obtaining PropertyManager

jp.co.intra_mart.framework.system.property.PropertyManager is structured in singleton pattern. PropertyManager itself is an abstract class and the constructor of this class cannot be used directly (refer to [Figure 8-4 PropertyManager Structure]).

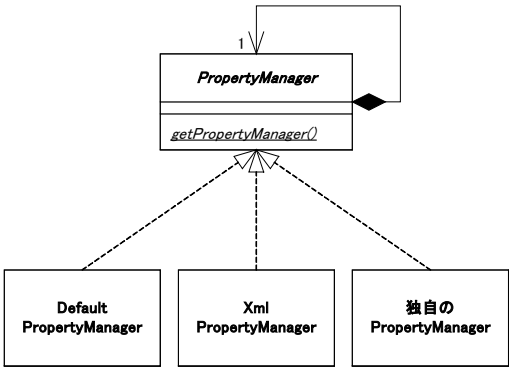


Figure 8-4 PropertyManager Structure

getPropertyManager method of PropertyManager is used in order to obtain the instance of PropertyManager. In this method, it obtains the class name of PropertyManager that is actually used from system property when JRE start up, and generates the instance of corresponding class. System property key at this point is the value to be obtained in PropertyManager.KEY[10].

If this system property is not set, instance of class name to be obtained in PropertyManager.DEFAULT_SYSTEM_MANAGER[11] will be generated.

How the PropertyManager is obtained is shown in [Figure 8-5 Obtaining PropertyManager (when system property is not set)] and [Figure 8-6 Obtaining PropertyManager (when system property is set)].

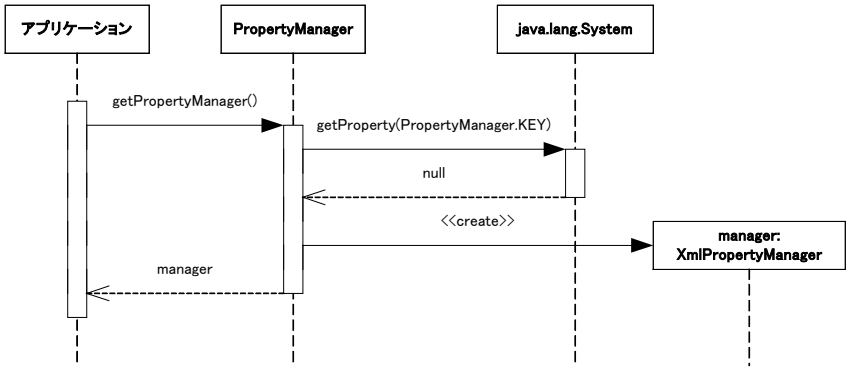
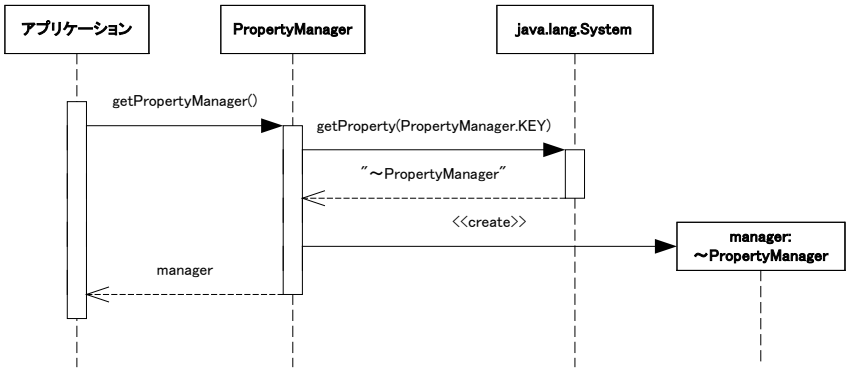


Figure 8-5 Obtaining PropertyManager (when system property is not set)



[10] In IM-JavaEE Framework Version 5.0, this value is "jp.co.intra_mart.framework.system.property.PropertyManager"

[11] In IM-JavaEE Framework Version 5.0, this value is "jp.co.intra_mart.framework.system.property.XmlPropertyManager"

Figure 8-6 Obtaining PropertyManager (when system property is set)

8.2.3.1 DefaultPropertyManager

jp.co.intra_mart.framework.system.property.DefaultPropertyManager is the PropertyManager provided as standard in IM-JavaEE Framework.

Property is set in resource file. Contents of resource file are set in the format called [property name=property value]. Characters that can be used are according to the java.util.ResourceBundle. This resource file should be placed in a class path that can be obtained from the application to be used. Please refer to API list for the detail of file name of resource file or property name to be set.

8.2.3.2 XmlPropertyManager

jp.co.intra_mart.framework.system.property.XmlPropertyManager is the PropertyManager provided as standard in IM-JavaEE Framework.

Property os set in XML. This XML file should be placed in a class path that can be obtained from the application to be used. Please refer to API list for the detail of file name of resource file or property to be set.

8.2.3.3 Original PropertyManager

If the developer creates PropertyManager originally, following conditions should be met.

- jp.co.intra_mart.framework.system.property.PropertyManager class is inherited.
- public default constructor (constructor with no argument) is defined.
- Appropriate value is returned for the following methods.
  - getPropertyHandlerName(String key)
    It returns complete class name of PropertyHandler corresponding to key.
    This class should implement jp.co.intra_mart.framework.system.property.PropertyHandler interface.
  - getPropertyHandlerParams(String key)
    It returns the array of parameter which is required for initialization of PropertyHandler that corresponds to key. Each element of the array is jp.co.intra_mart.framework.system.property.PropertyHandlerParam, and it should be possible that parameter name can be obtained by getName method and parameter value can be obtained by getValue method for this class.

9 Resource File Migration

9.1 Purpose

This chapter describes the migration method from the conventional resource files to the XML file that are available from Version 5.0 or later.

9.2 Resource File that can be migrated

Following resource files can be migrated in XML format.

- Service framework
- Event Framework
- Data Framework

9.3 Migration Method

It uses dedicated converter that is stored in the following directory.
/bin/convert.jar

- It displays help by following command.
  >java –jar convert.jar

- Example of use
  java -jar convert.jar -app shopping -src c:/imart/doc/imart/WEB-INF/classes -dir c:/xml

10 Addendum

[1] Java 2 Platform, Standard Edition (J2SE)
http://java.sun.com/j2se/1.4/

[2] Java 2 Platform, Enterprise Edition (J2EE)
http://java.sun.com/j2ee/1.3/docs/

[3] Extensible Markup Language (XML) 1.1
http://www.w3.org/TR/xml11/

[4] JavaTM BluePrints EnterPrise BluePrints
http://java.sun.com/blueprints/enterprise/

[5] JavaTM Servlet Specification Version 2.3
http://java.sun.com/products/servlet/download.html#specs

[6] Enterprise JavaBeansTM Specification, Version 2.0
http://java.sun.com/products/ejb/docs.html

[7] Java Transaction API (JTA) Version 1.0.1
http://java.sun.com/products/jta/index.html

intra-mart Accel Platform
IM-JavaEE Framework Specification

2012/10/01 Initial Version

TEL: 03-5549-2821
FAX: 03-5549-2816
E-MAIL: info@intra-mart.jp
URL: http://www.intra-mart.jp/