EJB Guide

JEUS 9

TMAXSOFT

owners.

The names of companies, systems, and products mentioned in this manual may not necessarily be indicated with a trademark symbol (TM, ®).

Open Source Software Notice

Some modules or files of this product are subject to the terms of the following licenses: APACHE2.0, CDDL1.0, EDL1.0, OPEN SYMPHONY SOFTWARE1.1, TRILEAD-SSH2, Bouncy Castle, BSD, MIT, SIL OPEN FONT1.1

Detailed Information related to the license can be found in the following directory: ${INSTALL_PATH}/license/oss_licenses

Document History

Product Version	Guide Version	Date	Remarks
JEUS 9	3.1.2	2025-03-24	-
JEUS 9	3.1.1	2024-12-24	-

Contents

Glossary

Secondary Storage Device

A storage medium that holds information permanently regardless of whether the computer is powered (e.g. a hard disk).

Load Balancing

Used to prevent tasks from being concentrated to one side in clustering.

Main Memory

The main memory of the computer is also known as runtime memory or RAM, standing for Random Access Memory.

Activation

A contrary concept to inactivation (or passivation). Indicates that a bean instance is moved from the secondary storage to the runtime memory.

Active Management

Indicates that email is automatically transmitted in an abnormal situation in JEUS.

Bean Pool

A pool of EJB instances.

Bean Type

Either of J2EE basic bean type (e.g. stateless, stateful, entity and message-driven bean) or a bean defined by a developer (a sub bean of the basic J2EE bean).

BMP

Bean-managed persistence entity bean.

CMP

Container-managed persistence entity bean.

Connection Pool

Pool of EJB objects. These objects are responsible for connecting a client to EJB and communicating.

DD

An abbreviation of deployment descriptor.

EJB

Enterprise Java Beans.

EJB Clustering

The settings of EJB beans which are distributed in multiple EJB engines. It may look like a bean from the client side, but internally loads are balanced for improving stability and performances.

EJB Client

A component using an EJB method (e.g. an independent Java application, Servlet or another EJB)

EJB Engine

"EJB container" in the J2EE specification. This provides a fundamental environment of the EJB component.

EJB Failover

Used to improve stability by using multiple engines which execute the same EJB application.

EJB Module

One EJB package in EJB JAR, or more. An EJB deployed to an EJB engine uses the EJB module.

Entity Cache

A special cache used to keep inactivated ones in entity beans. Only when this cache is full, those beans are inactivated to the secondary storage. Entity cache is used for performance improvement.

MDB

Message Driven Bean.

Object Management

Refer to 'connection pool' and 'bean pool environment'.

Passivation

To move an EJB instance from the runtime memory to the secondary storage until requested.

Round Robin

It is the same as the queue data structure, and selected from the one that entered the queue earlier. This guarantees fair load-balancing for each component.

SF

Stateful Session Bean.

SL

Stateless Session Bean.

Thread Ticket Pool

A pool of access tickets. To access an EJB from JEUS, the client must obtain "thread ticket" from the thread ticket pool.

1. Introduction to EJB

This chapter describes the components and characteristics of EJB, and how to install EJB and to configure its environment.

1.1. Overview

JEUS supports JSR 345 Jakarta™ Enterprise Beans 4.0 (hereafter EJB 4.0), which describes architecture for creating fully portable business components. The architecture simplifies the development efforts when implementing complex, mission-critical, and performance sensitive business logic.

> Starting with Jakarta EE 9, EJB 4.0 is introduced. However, it is not a completely new version that diverges from the existing EJB specifications. Instead, it mandates continued support for the features of EJB 2.x and EJB 3.x. As a result, in practice, users will primarily continue to work with EJB 2.x and 3.x. This guide, therefore, focuses on explaining the functionality of EJB 2.x and EJB 3.x, without delving into additional details about EJB 4.0.

According to the EJB 4.0 specification, the following is the concept of EJB.

"The Enterprise JavaBeans architecture is component architecture for the development and deployment of component-based distributed business applications. Applications written using the Enterprise JavaBeans architecture are scalable, transactional, and multi-user secure. These applications may be written once, and then deployed on any server platform that supports the Enterprise JavaBeans specification."

JEUS represents one such "server platform" mentioned in the excerpt. This guide describes how EJB is implemented in JEUS, and additional functions that are essential in an enterprise environment.

> For more detailed specification implementation information, refer to "JEUS Introduction Guide".

For increased performance and reliability, the JEUS implementation offers two non-standard features through clustering:

- Load-balancing for better performance
- Failover for increased reliability

Both of these functions require the clustering of two or more Managed Servers (hereafter MS).

> For more information about EJB clustering, see EJB Clustering. For details on MS clustering, see "JEUS Clustering" in JEUS Domain Guide.

1.2. Components

The following figure shows main components of an EJB implementation entity.
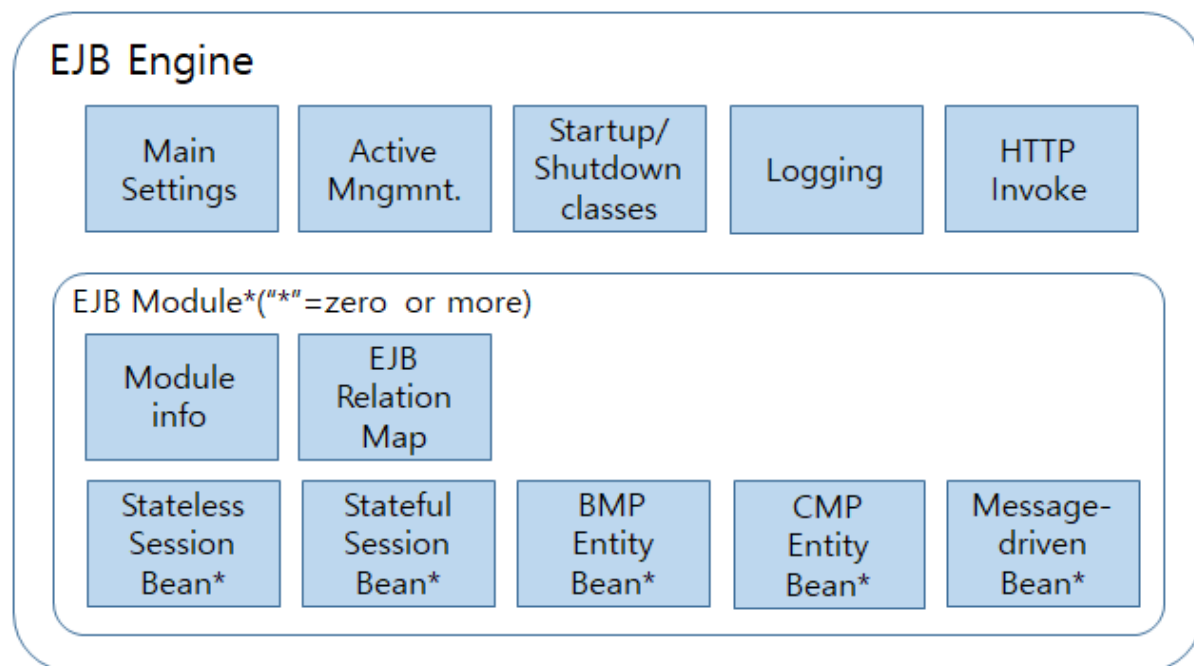
Main Components of an EJB Implementation Entity

- **The EJB engine**

  Refers to the EJB container described in EJB 4.0. It provides a runtime environment for deployed EJB modules. The EJB engine will be described in detail in EJB Engine.

- **The EJB module**

  Used to group and manage a set of Enterprise JavaBeans as a single unit. EJB module is the unit used to assemble, deploy, and control EJB in JEUS. Therefore, all EJBs must be part of an EJB module. The EJB module will be described in detail in EJB Module.

- **The EJB components**

  Represents the actual business components of an EJB module. For information about common characteristics of EJB components, refer to Common Characteristics of EJB.

  The following are the five types of EJB components. For more detailed information, refer to the following chapters:

  - Stateless Session Bean : Session Bean

  - Stateful Session Bean : Session Bean

  - BMP Entity Bean : Entity Bean

  - CMP 1.1 & CMP 2.x Entity Bean : Entity Bean

  - MDB : Message Driven Bean(MDB)

> Entity Bean has been replaced by JPA since EJB 3.0. For information about JPA, refer to "JEUS JPA Guide".

1.3. EJB Environment and Configuration

This section describes configuration elements related to JEUS EJB, such as the directory structure, EJB configuration files, and various tools. It also describes system properties related to EJB and their characteristics.

1.3.1. Directory Structure

This section describes directories and files related to JEUS EJB.

```
{JEUS_HOME}
    |--bin
    |    |--appcompiler
    |    |--jeusadmin
    |--domains
    |    |--<domain_name>
    |    |       |--config
    |    |       |    |--domain.xml
    |    |       |--servers
    |    |               |--<server_name>
    |    |                       |--logs
    |    |                               |--error.log
    |    |--.application
    |            |--<EAR_archive_file>
    |            |        |--APP-INF
    |            |        |--Lib
    |            |        |--META-INF
    |            |        |--<EAR_archive_file>_jar__
    |            |                |--META-INF
    |            |                |--EJB class file
    |            |--<EJB_archive_file>
    |                    |--<EJB_archive_file>_jar__
    |                            |--META-INF
    |                                    |--ejb-jar.xml
    |                                    |--jeus-ejb-jar.xml
    |--lib
    |    |--schemas
    |          |--jakartaee
    |          |--jeus
    |--sample
          |--ejb
              |--.java EJB sample
```

bin

The directory where scripts for launching tools, that are used to help the administrator manage

EJBs, are stored. These tools are: jeusadmin and appcompiler.

domains\\<*domain name*>\\config

The most important configuration files of JEUS system. This directory contains the domain.xml file.

domains\\<*domain name*>\\servers\\<*server name*>\\logs

The directory where the log files of EJB engine are stored. If a separate file handler is specified, a separate file will be created, otherwise, the setting in the server log configuration file will be followed.

domains\\<*domain name*>\\.application

The archive files of applications used by domains are copied and decompressed to this directory. It contains EJB implementation classes and helper classes.

File	Description
<EAR archive file>	EAR application configuration file is copied to the directory named EAR application-id.
<EJB archive file>	EJB module configuration file is copied to the directory named EJB module-id.

lib\\schemas

The schema files relating to EJB are located in each directory as shown below.

Sub directory	Description
jakartaee	Contains all Jakarta EE XML schema files related to EJB.
jeus	Contains all JEUS XML schema files related to EJB.

samples\\ejb

Contains several sub-directories and sample code, demonstrating the implementation of various types of EJBs.

> The bracket expressions used in the above directory list (such as "<*server name*>/") means that the string in between the brackets must be replaced with an actual system or configuration dependent value. For instance, if the current machine is called server1, the corresponding JEUS MS must also be called server1. Thus the "<*server name*>/" expression is replaced with the actual directory name of "server1/" on the actual system.

1.3.2. XML Configuration File

The following XML configuration files pertain to the management and configuration of JEUS EJB.

The contents of the previous files must always start with a standard JEUS-defined XML header. The XML schema files are located in the JEUS_HOME/lib/schemas/jeus/supportLocale/ko directory. And the root element should specify the namespace of JEUS XML schema as the current namespace.

- **domain.xml** (jeus-domain.xsd)

  EJB engine configuration. For more information, see "JEUS Server Guide" and EJB Engine in this guide.

  - Location

    ```
    JEUS_HOME/domains/<domain name>/config
    ```

  - XML header of **domain.xml**

    XML Header: <domain.xml>

    ```
    <?xml version="1.0"?>
    <domain version="8.0" xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
    ```

- **ejb-jar.xml** (ejb-jar_4_0.xsd)

  Main EJB engine configuration file. For more information, refer to the EJB 4.0 specification.

  - Location

    ```
    META-INF\ Structure in the standard JAR file
    ```

  - XML header of **ejb-jar.xml**

    The XML header of ejb-jar.xml is included in the EJB standard.

    XML Header: <ejb-jar.xml >

    ```
    <?xml version="1.0"?>
    <ejb-jar version="4.0" xmlns="https://jakarta.ee/xml/ns/jakartaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="https://jakarta.ee/xml/ns/jakartaee
    https://jakarta.ee/xml/ns/jakartaee/ejb-jar_4_0.xsd">
    ```

- **jeus-ejb-dd.xml** (jeus-ejb-dd.xsd)

  Configures JEUS Deployment Descriptors (hereafter DD) for JEUS EJB module. For more information, refer to EJB Module.

  - Location

    ```
    META-INF\ Structure in the standard JAR file
    ```

  - XML Header of **jeus-ejb-dd.xml**

XML Header: <jeus-ejb-dd .xml>

```xml
<?xml version="1.0"?>
<jeus-ejb-dd xmlns="http://www.tmaxsoft.com/xml/ns/jeus" version="8.0">
```

The order of all tags used in the manual follows the order of the XML schema configuration file. Refer to the Reference Book for more information about tag ordering. But since it is not easy to maintain the order in actual use, JEUS provides automatic function for ordering tags. This eliminates the need to consider the order when writing XML configuration file.

When an XML example from any of these files is used in this manual, the standard header will usually be omitted. You must make sure that these headers are included in the actual XML configuration files.

1.3.3. Related Tools

The following tools may be used to manage EJB engines, modules, and components:

- **Console Tool (jeusadmin)**

  The tool used to control and to monitor EJB engines. For more information, refer to "EJB Engine Commands" in JEUS Reference Guide.

  For detailed description of the appcompiler for EJB 2.x, refer to "appcompiler" in JEUS Reference Guide.

1.4. EJB Basic Settings

This section describes the installation steps for using EJB in JEUS. In the following example, we shall assume that the EJB engine is configured on the server named server1.

1. Make sure that JEUS has been properly installed on the system.

2. Execute the command screen and enter the following command in the $JEUS_HOME/bin directory to start MASTER.

```
$ startMasterServer -domain domain1 -u administrator -p passsword1 -verbose
. . .
[2016.08.06 21:33:15][2] [adminServer-1] [SERVER-0248] The JEUS server is STARTING.
. . .
[2016.08.06 21:33:16][3] [adminServer-1] [Security-0082] The JEUS security manager started.
. . .
[2016.08.06 21:33:16][3] [adminServer-1] [SERVER-0173] The JNDI naming server started.
. . .
[2016.08.06 21:33:28][2] [launcher-10] [Launcher-0034] The server[adminServer] initialization
completed successfully[pid : 2380].
```

```
[2016.08.06 21:33:28][0] [launcher-1] [Launcher-0040] Successfully started the server. The server
state is now RUNNING.
```

> [3] represents the log level, CONFIG. It is only displayed when the specified
> log level is higher than CONFIG.

3. The EJB engine can be set in domain.xml. If no setting is made, the default value is used.

> Modifying the MS setting using the tool provided by JEUS is possible only
> when MASTER is running.

4. Open another command prompt and enter the following. (Host information is that of MASTER.)

```
$ jeusadmin -host localhost:9736
User name:
```

5. Enter your administrator user name and password (as configured during JEUS installation).

```
User name: administrator
Password:
```

6. At the command line, enter the following. The server will start.

```
$ startManagedServer -domain domain1 -server server1 -u administrator -p passsword1 -verbose
...
[2016.08.07 14:10:10][2] [server1-1] [SERVER-0248] The JEUS server is STARTING.
...
[2016.08.07 14:10:17][2] [server1-1] [SERVER-0248] The JEUS server is RUNNING.
```

7. Enter 'help' to get a list of commands that can be used to monitor and control the EJB engine.

```
[MASTER]domain1.adminServer>help -g EJB
[ EJB] _____
  cancel-ejb-timer          Cancels active EJB timers.
  ejb-timer-info            Lists the active EJB timers. If no options are
                            specified, a list of all EJB modules that
                            containtimers will be displayed.
  modify-active-management  Modify the active management configuration.
  modify-check-resolution   Set the check resolution of the EJB engine on the
                            server.
  show-active-management     Shows the active management of the EJB engine on
                            the server.
  show-check-resolution      Shows the check resolution of the EJB engine on
                            the server.
To show detailed information for a command, use 'help [COMMAND_NAME]'.
```

```
ex) help connect
```

8. Enter stop-server (stop server) → local-shutdown (stop MASTER) → exit (stop tool) in at the command line.

```
[MASTER]domain1.adminServer>stop-server server1
Server [server1] was successfully stopped.
[MASTER]domain1.adminServer>local-shutdown
The server [adminServer] has been shut down successfully.
offline>exit
```

9. The entire JEUS system has been terminated now.

> For more information on installation and setup, refer to "JEUS Getting Started Guide" and "JEUS Server Guide".

2. EJB Engine

This chapter describes basic knowledge of the EJB engine. It also describes the top-level concepts of JEUS EJB, such as its structure, configuration, operation, monitoring, and tuning.

2.1. Overview

The JEUS EJB engine provides a runtime environment for EJB. The term EJB Container used in the EJB standard is equivalent to the term **EJB engine** that will be used throughout this manual. For more information about EJB modules and EJB deployment, refer to EJB Modules and Common Characteristics of EJB.

2.2. Key Features

The following explains major functions of the EJB engine.

- **EJB Engine and Managed Server (MS)**

  A single EJB engine can exist on a MS. However, multiple EJB engines can exist on a single domain because multiple MSs can exist on a domain.

  Usually, multiple MSs and EJB engines are configured and executed across multiple machines (CPUs). The MSs are clustered and managed by MASTER. Such configuration is often referred to as a cluster of engines. This is desirable since it results in better system performance and higher reliability and safety. In EJB Clustering, EJB clustering will be described in detail.

- **Main EJB Engine Settings**

  The main configuration file for the EJB engines is **<ejb-engine>** in **domain.xml**. More information will be discussed in Configuring the EJB Engine.

- **EJB Engine Logging**

  Logs can be written on MSs according to the setting in domain.xml, but a user can also keep a separate EJB engine log (jeus.ejb). Even if a separate EJB engine log is used, the EJB engine log is also recorded in the MS log .

  When a file handler is configured in the log handler, the log file is created using the specified file name. It is also possible to display any log messages using a console handler. Normally, in this case, the log messages will be displayed on the command screen, where the MS was started, through the pipe.

  > A user-generated user handler can also be registered. For more information, refer to "Logging" in JEUS Server Guide.

- **Active Management**

  In active management, the EJB engine sends out email notifications to the administrator if a problem occurs in the EJB modules.

  For example, when a fatal error occurs, such as an infinite loop or deadlock due to an error in the Bean class implementation, the EJB engine detects the error and sends out a notification. Additionally, you can configure an error-policy to send a notification e-mail and display a message to recommend a restart of the MS, where the relevant EJB is running, for any abnormal activities. For detailed explanations about active management configuration, see Configuring the EJB Engine.

- **HTTP Invoke**

  When a remote client looks up an EJB instance through the JNDI naming service, it will get back an EJB RMI stub that is used to invoke methods on the EJB. By default, this remote communication from stub to EJB is carried out through the RMI runtime. The RMI communication is based on a TCP socket, which has a constraint in an environment where there is a firewall because it requires a separate TMI communication port. In this case, a special communication mode called **HTTP invoke mode** is needed. In this mode, a RMI request from a remote client is wrapped with HTTP and sent to a web engine, which forwards the request to a servlet (jeus.rmi.http.ServletHandler) that handles RMI requests. The servlet sends the request to the RMI runtime, then calls an actual EJB method, and wraps and sends the result to the remote client. The HTTP invoke mode can be configured for each EJB engine or EJB component.

  If the HTTP invoke mode is configured in the <invoke-http> element of domain.xml, the mode is applied to all the modules in the EJB engine. The settings in jeus-ejb-dd.xml of the EJB module can only be applied to a specific EJB component. The settings in jeus-ejb-dd.xml takes precedence over those in domain.xml. For more information about configurations of domain.xml and jeus-ejb-dd.xml, see Configuring the HTTP Invocation Environment.

2.3. EJB Engine Directory Structure

The following figure shows directories and files used to manage EJB engines.

```
{JEUS_HOME}
    |--bin
    |    |--[01]appcompiler
    |    |--[01]jeusadmin
    |--domains
    |        |--<domain_name>
    |                |--config
    |                |    |--[X]domain.xml
    |                |--servers
    |                        |--<server_name>
    |                                |--logs
    |                                        |--[T]error.log
    |--lib
        |--schemas
            |--jakartaee
```

```
            |--jeus

* Legend
- [01]: binary or executable file
- [X] : XML document
- [J] : JAR file
- [T] : Text file
- [C] : Class file
- [V] : java source file
- [DD] : deployment descriptor
```

Basic directories are the same as Directory Structure. Main directories used by an EJB engine are as follows:

bin

Contains the tool for managing EJB engines.

EJB Engine Management Tool	Description
appcompiler	Creates classes necessary to deploy EJB, compiles and fast deploys the classes.
jeusadmin	Used to control and monitor an EJB engine.

domains/<*doamin name*>/servers/<*server name*>/logs

The directory where the log files of EJB engine are stored. If a separate file handler is specified, a separate file will be created, otherwise, the setting in the server log configuration file will be followed.

2.4. Configuring the EJB Engine

This section explains how to configure EJB engines using domain.xml.

EJB engine configuration is divided into three sections as follows. The specified settings are stored in the domain.xml file, under the JEUS_HOME/domains/<*domain name*>/config directory.

- [Basic]

- [Active Management]

- [Timer Service]

> There is one EJB engine in each MS. For details on how to add an MS, refer to "JEUS Configuration" in JEUS Server Guide.

2.4.1. Basic Configurations

The following example configures the basic settings of the EJB engine in domain.xml.

Configuring EJB Engine: <domain.xml>

```
<ejb-engine>
    <resolution>1000</resolution>
    <use-dynamic-proxy-for-ejb2>true</use-dynamic-proxy-for-ejb2>
    <enable-user-notify>false</enable-user-notify>
    <timer-service>
        <support-persistence>true</support-persistence>
        <max-retrial-count>1</max-retrial-count>
        <retrial-interval>5000</retrial-interval>
        <thread-pool>
            <min>2</min>
            <max>30</max>
            <period>3600000</period>
        </thread-pool>
    </timer-service>
    <async-service>
        <thread-min>0</thread-min>
        <thread-max>10</thread-max>
        <access-timeout>30000</access-timeout>
    </async-service>
</ejb-engine>
```

The following table describes the configuration items of **Basic Options** and**Advanced Options** by section.

- **Basic Options**

Tag	Description
Use Dynamic Proxy For Ejb2	Uses dynamic proxy instead of RMI stub.
Resolution	The interval for checking the active status of the EJB engine (activation/passivation). When Active Management is used, this checks the number of blocked threads. This also monitors the beans that have been idle for longer than the passivation timeout period.

- **Advanced Options**

  - Ejb Engine

Tag	Description
Enable User Notify	Records EJB exceptions in the user log that is configured on the server. For more information about the user log, refer to "Logging" in JEUS Server Guide.

  - Async Service

The following table describes the asynchronous service invocation options.

Tag	Description
Thread Min	The minimum number of threads that can be maintained.
Thread Max	The maximum number of threads that can be maintained.
Access Timeout	The waiting period before deleting the Future object if the client does not invoke the get method after an asynchronous method has finished processing. This prevents memory leaks that are caused by the client forgetting to call the get method.

◦ Invoke Http

This option is used when EJB RMI stubs cannot access an RMI runtime port.

Tag	Description
HTTP Port	Web server port to receive the HTTP-RMI requests. The RMI handler Servlet must be already deployed and running on this web server and web engine.
Url	This must be configured to the URI path of the RMI handler Servlet that will be invoked by the HTTP-RMI stub. ◦ This URI only contains the request path of the Servlet, and excludes the protocol, Web server IP address, and the port number. You must set the <contex-path> option of jeus-web-dd.xml and <url-pattern> option of web.xml. If a separate jeus-web-dd.xml file is not created, the war file name, which is the default setting for the <context-path> option, is used (in the example, rmiHandlerServlet). ◦ The protocol is implicitly set to HTTP and the IP address is assumed to be same as the IP address of the RMI runtime. This means that the Web server and the Web engine that will receive HTTP-RMI requests must be on the same machine as the RMI runtime. The address of the RMI runtime will be known to the RMI stub. The web server port must be configured in the <http-port> element. ◦ The jeus-web-dd.xml file is not included in the rmiHandlerServlet.war, provided by JEUS. Therefore, by default, context uses rmiHadlerServlet, which has the same name as the module. A Servlet handler is set in the <url-pattern> element. If you want to use configuration other than the default one, create the jeus-web-dd.xml file, edit the web.xml file, and then deploy the rmiHandlerServlet.war file.

2.4.2. Configuring Active Management

Active Management setting consists of the engine restart and email notification options. Engine restart condition is determined according to the maximum number of blocked EJB threads allowed before the EJB engine is restarted.

The following example configures the Active Management setting of an EJB engine in domain.xml.

```
<ejb-engine>
    <resolution>1000</resolution>
    <use-dynamic-proxy-for-ejb2>true</use-dynamic-proxy-for-ejb2>
    <enable-user-notify>false</enable-user-notify>
    <active-management>
        <max-blocked-thread>-1</max-blocked-thread>
        <max-idle-time>300000</max-idle-time>
        <email-notify>
            <smtp-host-address>gw.tmaxsoft.com</smtp-host-address>
            <sender-id>sender@tmaxsoft.com</sender-id>
            <sender-password>1234567</sender-password>
            <from-address>sender@tmaxsoft.com</from-address>
            <to-address>receiver@tmaxsoft.com</to-address>
            <property>
                <key>mail.smtp.port</key>
                <value>587</value>
            </property>
            <property>
                <key>mail.smtp.auth</key>
                <value>true</value>
            </property>
            <property>
                <key>mail.smtp.starttls.enable</key>
                <value>true</value>
            </property>
        </email-notify>
    </active-management>
</ejb-engine>
```

The following table describes the configuration items of **Basic Options** and **Advanced Options** by section.

- **Basic Options**

Tag	Description
Max Blocked Thread	Maximum number of blocked EJB threads allowed before recommending the restart of the EJB engine. If this number is set too low, the EJB engine restart will be recommended too often. The default value is set to -1, which means that there is no limit on the allowed number of blocked threads. This means that the EJB engine does not restart because of blocked threads.

Tag	Description
Max Idle Time	Maximum idle time for EJB threads. If a thread remains idle for this time period, it will be added to the blocked thread list. This setting is used to determine if a thread is blocked.

- **Advanced Options**
  - Email Notify

    Configures where to send an email message when the restart recommendation conditions trigger an EJB engine to restart.

Tag	Description
SMTP Host Address	Host address of the SMTP server.
Sender ID	ID to authenticate using SMTP address.
Sender Password	Password of the ID for authentication through the SMTP address.
From Address	Address of the sender.
To Address	Address of the recipient.
CC Address	Recipient of a carbon copy (CC).
Bcc Address	Recipient of a blind carbon copy (BCC).
Property	If there are other necessary properties provided by the javaMail API besides the basic SMTP properties, they can be configured as key-value pairs.

2.4.3. Configuring Timer Service

The EJB timer service enables you to receive timer callback periodically or at a specified time. Refer to the EJB specifications for information about basic usage of the timer service. This chapter will cover the explanation and setting of the timer service in JEUS EJB. For more information, refer to EJB Timer Service.

2.5. Configuring System Logs

As system logging and user logging settings are common settings that apply not only to the EJB engine but also to all other engines, refer to "Logging" in JEUS Server Guide.

2.6. Controlling and Monitoring of EJB Engine

Controlling the EJB engine is similar to controlling other JEUS engines (servlet or JMS). You can

monitor the execution environment information and status information of the EJB engine using the console tool. For more information, refer to "EJB Engine Commands" in JEUS Reference Guide.

2.7. Tuning the EJB Engine

There are a number of configuration settings that can be tweaked to increase the overall performance of individual JEUS EJB engines. In this section, we will briefly touch the following performance-related settings of the EJB engine.

The following are required for tuning purposes.

- Tuning the resolution setting
- Using the fast-deploy feature
- Setting up the system log for optimal performance
- Not using Active Management
- Using the HTTP invoke mode

> For more information or tips about the EJBMain.xml file, refer to "11. domain.xml EJB Engine Configuration" section of the JEUS XML Reference. In the XML/Schema, those names marked with 'P' are related to performance.

2.7.1. Tuning the Engine Resolution

Resolution checks for the EJB engine status are performed in the following two cycles.

- Active management check cycle: Checks the number of blocked threads and then restarts the EJB engine.
- Passivation check cycle: Checks all sub components (e.g. Bean pool) and checks whether each Bean should be inactivated.

Use a long resolution period for bette<par performance (by performing engine self-maintenance less often) at the expense of a less reliable engine that wastes system memory. If the value is low, the engine can be kept up-to-date but this will impact the overall performance. Therefore, it is very important to set an appropriate value.

> Depending on the resolution value, <passivation-timeout> or <disconnect-timeout> may not be triggered at the expected time.

2.7.2. Using the Fast-Deploy Feature

The fast deployment option configured for each application instead of being configured in the

EJBMain.xml file. Nevertheless, it has a significant effect on the system performance. If the EJB modules that are to be deployed during engine boot-time have already been compiled to generate the RMI stubs and skeletons, you must use the -fast option when running the **deploy** command. This will cause the engine to skip RMI class generation during the EJB module deployment. For more information about EJB modules and deployment, see EJB Modules.

2.7.3. Configuring System Logs for Optimal Performance

The system log settings can be tweaked in three ways for optimal performance:

- If possible, it is better to use a file handler in order to improve logging performance.

- Set a large value for the buffer size of the file handler.

- Set the log level to SEVERE.

> These suggestions only apply to a stable production environment. In the development environment, opposite settings might be preferred, e.g., using console handler, small buffer size, and the FINE log level.

2.7.4. Not Using Active Management

It is not necessary to use Active Management at the EJB engine level. By default, this setting is not used since it could lower the performance. Instead, it is more convenient to use Active Management defined in the Servlet engine. In most cases, Active Management is not configured in the EJBMain.xml file.

2.7.5. Using the HTTP Invoke Mode

Using the HTTP invoke mode can improve the performance when there are high number of requests. Web server controls the number of requests to JEUS by creating fewer connections than RMI, which creates a separate thread for each client request. But keep in mind that converting to HTTP protocol generally lowers the performance.

3. EJB Modules

This chapter describes EJB module structure and how to manage the modules.

3.1. Overview

EJB modules represent the deployable unit for a JEUS EJB engine. It is a concept of grouping EJB components and using deployment descriptors (hereafter DD) to describe the configuration settings. Even if you are only deploying one Enterprise JavaBean, it must be packaged in an EJB module.

> A DD (ejb-jar.xml) is required for components of EJB 2.1 or earlier. However, for components of EJB 3.0 or later, DD is optional because annotation is supported.

3.2. Managing EJB Modules

The following outlines the four main activities involved when managing an EJB module in JEUS. For more detailed information, refer to each section.

- Assembly: Assembling EJB Modules

- Deployment: Deploying EJB Modules

- Control (undeployment, redeployment, stop-application, start-application): Controlling EJB Modules

- Monitoring: Monitoring EJB Modules

The following figure shows the flow chart for managing EJB modules.
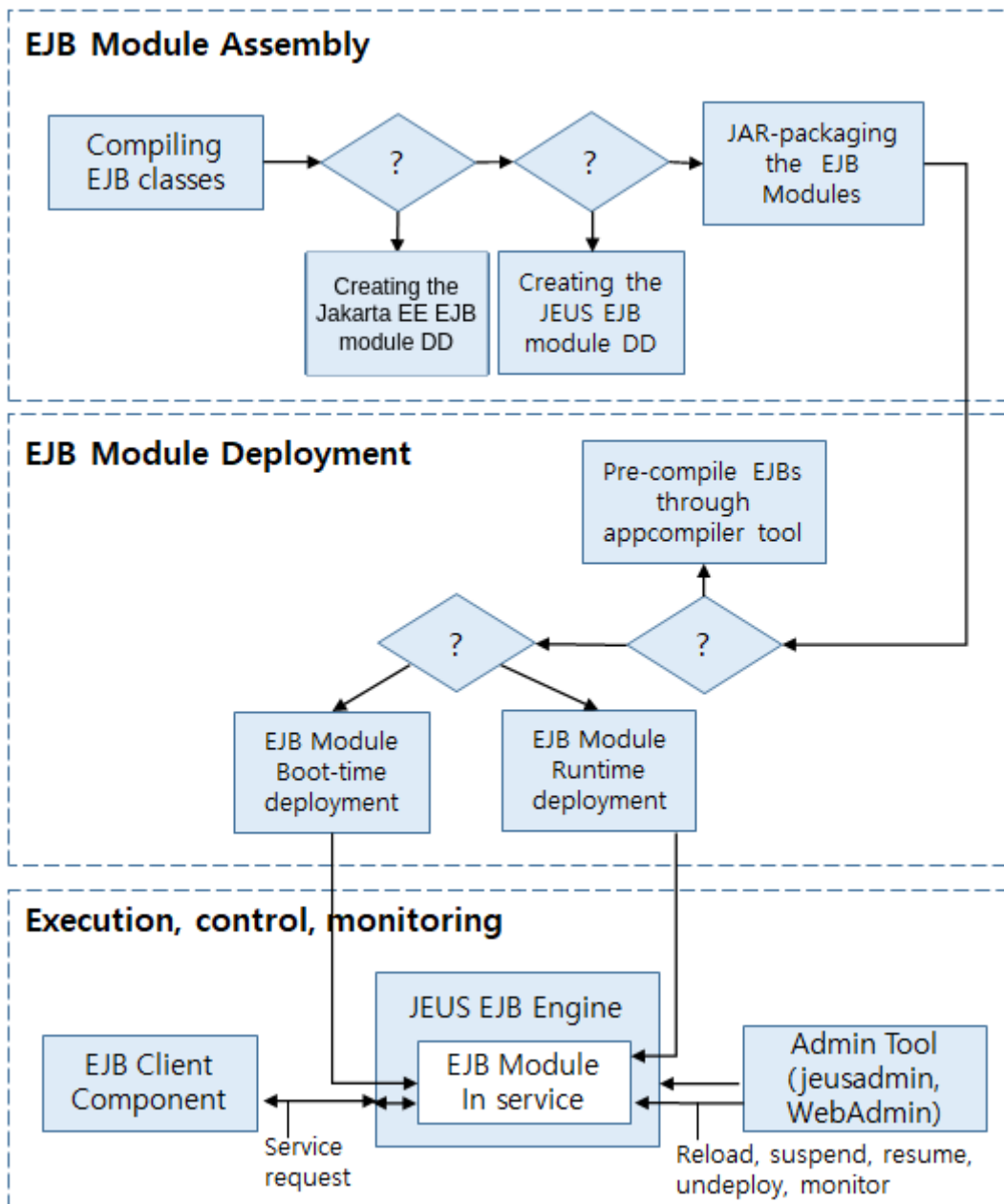
Flow Chart for Managing EJB Modules

Structure of EJB Module JAR File

The following figure explains the structure of standard EJB module JAR file.

```
EJB Module JAR
    |--META-INF
    |      |--[X]ejb-jar.xml
    |      |--[X]jeus-ejb-dd.xml
    |--<package_name>
    |      |--[C]Bussiness Interface
    |      |--[C]EJB Implementation
    |      |--[C]Helper class
    |--[J]Helper library

* Legend
- [01]: binary or executable file
- [X] : XML document
```

```
 - [J] : JAR file
 - [T] : Text file
 - [C] : Class file
 - [V] : java source file
 - [DD] : deployment descriptor
```

META-INF

The following describes subordinate files.

File	Description
ejb-jar.xml	Actual standard EJB DD. It may not exist if annotation is used to describe configuration information. It is required for EJB 2.x style.
jeus-ejb-dd.xml	DD needed when EJB is deployed to JEUS (optional).

<package_name>

Contains the EJB classes. They must be organized to reflect the Java package hierarchy as defined in the EJB source code. Example: if an EJB declares that it is part of the package called "mypackage", the EJB classes must reside directly beneath a directory named "mypackage" in the EJB JAR file.

The following table describes the components of EJB.

Element	Description
Interface	EJB 2.x, 3.0, or later style interface used for client access.
Enterprise Bean class	Bean classes that implement the actual business logic according to the interfaces.

Helper library

Library defined in the MANIFEST.MF Class-Path entry.

3.3. Assembling EJB Modules

As we see in Flow Chart for Managing EJB Modules, steps for assembling an EJB module are as follows:

1. Compile the EJB classes

   Compile the EJB source files implemented by the developer

2. Create the DDs
   - Create ejb-jar.xml DD
   - Create jeus-ejb-dd.xml DD

3. Package the EJB JAR file

Package the EJB classes and DDs into a JAR file

We will illustrate the procedures by using a simple example bean called counter. The counter bean is a stateful session bean that resides in the JEUS_HOME/samples/ejb/basic/statefulSession/ directory. It contains the following two Java source files:

- Counter.java (business interface)

- CounterEJB.java (bean implementation)

> For more information about stateful session beans and the other bean types, refer to Session Bean and Message Driven Bean(MDB).

3.3.1. Compiling EJB Classes

The first step in assembling an EJB module from existing EJB source files is to compile the source files. This is accomplished by using the JDK javac compiler. The default classpath is JEUS_HOME/lib/system/jakarta.ejb-api.jar, but if you are using annotations that correspond to the jeus-ejb-dd.xml file, set the classpath to JEUS_HOME/lib/system/jeusapi.jar. If there are other helper classes, add them all to the classpath.

The following example shows how to compile EJB Java files for the counter bean in a Unix environment.

```
$ javac –classpath JEUS_HOME/lib/system/jakarta.ejb-api.jar  *.java
```

The previous command will create the class files.

3.3.2. Creating Deployment Descriptors

There are two DDs that must be prepared before deploying an EJB module:

- Standard EJB DDs

  DD with the name, ejb-jar.xml, placed in the META-INF directory of the EJB module. Some or all parts of configuration information can be created using annotation when you create the EJB components. If you want to apply DD setting without using annotation, set the <metadata-complete> attribute in ejb-jar.xml as true. If annotation and DDs are used together, DD settings override the annotation settings.

- JEUS EJB DD

  DD with the name, jeus-ejb-dd.xml, placed in the META-INF directory of the EJB module. For more information, refer to Creating JEUS EJB DD (jeus-ejb-dd.xml).

Creating standard EJB DDs (ejb-jar.xml)

The DD format complies with the EJB specification. Annotation is enough in most cases, but a DD can be used when necessary.

```
package ejb.basic.statefulSession;
@Remote
public interface Counter
{
    public void increase();
    ...
}
```

```
package ejb.basic.statefulSession;
@Stateful
@EJB(name="counter")
@TransactionManagement( TransactionManagementType.BEAN)
public class CounterEJB implements Counter {
    private int count = 0;

    public void increase()
    {
        count++;
    }
    ...
}
```

The following example is a simple Java EE EJB DD of the counter bean.

DD in the EJB Specification: <ejb-jar.xml>

```
<?xml version="1.0"?>
<ejb-jar version="4.0" xmlns="https://jakarta.ee/xml/ns/jakartaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="https://jakarta.ee/xml/ns/jakartaee
https://jakarta.ee/xml/ns/jakartaee/ejb-jar_4_0.xsd">
    <enterprise-beans>
        <session>
            <ejb-name>counter</ejb-name>
            <business-remote>ejb.basic.statefulSession.Counter</business-remote>
            <ejb-class>ejb.basic.statefulSession.CounterEJB</ejb-class>
            <session-type>Stateful</session-type>
            <transaction-type>Bean</transaction-type>
        </session>
    </enterprise-beans>
    <assembly-descriptor/>
</ejb-jar>
```

> For EJB 3.0 or later, ejb-jar.xml is optional because all contents of ejb-jar.xml can be described using annotation.

Creating JEUS EJB DD (jeus-ejb-dd.xml)

The main reason for creating JEUS EJB DD is to map the basic settings and external references, which are needed for deployment to an EJB engine. Instance pooling, entity bean, and DB table mappings are also configured in the DD.

> Such settings are needed to deploy EJB modules according to JEUS, but some parts can be configured using annotation. Also, since they can be deployed using the default values without the jeus-ejb-dd.xml file, this file is only necessary when separate environment settings are needed.

JEUS EJB DDs are additional settings added to the standard Java EE DDs. Like the standard EJB DDs, JEUS EJB DDs are also applied to an EJB module.

The following are the main elements of a JEUS EJB DD:

JEUS EJB DD : <jeus-ejb-dd.xml>

```xml
<?xml version="1.0"?>
<jeus-ejb-dd xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
    <module-info>
        <role-permission>
            <!-- See chapter  "Configuring EJB Security" -->
            . . .
        </role-permission>
    </module-info>
    <beanlist>
        <jeusbean>
            <!-- See chapters "Common Characteristics of EJB"
                and  "Session Bean" -->
            . . .
        </jeusbean>
        <jeusbean>
            <!-- See chapters "Common Characteristics of EJB"
                and  "Session Bean" -->
            . . .
        </jeusbean>
        . . .
    </beanlist>
    <ejb-relation-map>
        <!-- See chapter "Entity Bean" -->
    </ejb-relation-map>
    <message-destination>
        <jndi-info>
            <ref-name>ejb/AccountEJB</ref-name>
            <export-name>ACCEJB</export-name>
        </jndi-info>
    </message-destination>

    <library-ref>
        <library-name>jsf</library-name>
        <specification-version>
            <value>1.2</value>
        </specification-version>
        <implementation-version>
```

```
            <value>1.2</value>
        </implementation-version>
    </library-ref>

</jeus-ejb-dd>
```

The following are the main elements of a JEUS EJB DD:

Tag	Description
<module-info>	Specifies the information that is applied to all EJB modules. For more information about role assignment, refer to Configuring EJB Security.
<beanlist>	Defines EJBs included in the EJB module. The name and content of this element can have various features according to the bean type (SessionBean. Entity Bean, and MDB). For detailed explanations about <beanlist> and its child tags, see Common Characteristics of EJB and from Session Bean to Message Driven Bean(MDB).
<ejb-relation-map>	Configures EJB relationship mappings that define CMR mappings for CMP 2.0 beans. Since using CMP is no longer recommended, use JPA instead.
<message-destination>	Maps message destination defined in the <message-destination> element of ejb-jar.xml and the actual destination object registered with JNDI.
<library-ref>	Configures the shared library information that is used by the application.

3.3.3. Packaging an EJB JAR File

The following explains how to package EJB classes and DDs into a JAR file. You can create an EJB JAR file using a JAR utility. We assume that:

- The counter EJB class files reside in the ejb/basic/statefulSession directory.

- The ejb-jar.xml and jeus-ejb-dd.xml do not exist because annotations are used in the classes.

The following explains how to pack an EJB JAR file.

1. Pack the EJB classes into a JAR file as follows.

   ```
   $ jar cf countermod.jar ejb/basic/statefulSession
   ```

   The Jakarta EE EJB module named countermod.jar is created.

2. In order to use this EJB in a client, you need an interface file. Create the counterbeanclient.jar file, which will be distributed for client use, as follows:

   ```
   $ jar cf counterbeanclient.jar ejb/basic/statefulSession/Counter.class
   ```

> The counterbeanclient.jar is a library distributed for remote clients to use.

3.4. Deploying EJB Modules

This section describes how to deploy EJB modules.

3.4.1. Deployment

In general, deployment is accomplished by compiling EJB classes. But you can reduce the deployment time by using the appcompiler to precompile the classes. You can also use the Auto Deploy function for easy deployment of EJB modules. For more information about deployment, refer to "JEUS Applications & Deployment Guide".

EJB modules are deployed on JEUS by using the following methods as described in Flow Chart for Managing EJB Modules.

- Using the appcompiler tool

  For EJB 2.x modules, the stub and skeleton files can be created in advance by using the appcompiler tool, speeding up the deployment process.

- Boot-time deployment

  EJB modules are automatically deployed to an EJB engine whenever the engine boots.

- Runtime deployment of an EJB module

  EJB modules are deployed to an already running EJB engine using the console tool.

Using the appcompiler tool

For EJB 2.x modules, stub and skeleton files can be created in advance by using the appcompiler tool, speeding up the deployment. When you are using the appcompiler tool, specify the -fast option with the deployment command in the console tool. Without this option, the appcompiler tool is not effective. For more information about the appcompiler tool, refer to "appcompiler" in JEUS Reference Guide.

Boot-time Deployment of an EJB Module

Boot-time Deployment refers to deploying EJB modules to an EJB engine whenever an MS boots. Boot-time deployment is applied to the applications deployed through MASTER. For more information about application deployment settings, refer to "Application Implementation and Deployment" in JEUS Applications & Deployment Guide.

Multiple applications can be registered to a single domain.

Runtime Deployment of an EJB module

Runtime-deployment is an act of actually deploying an EJB module to an already running EJB engine. By using the **deploy** command in the console tool, an EJB module can be installed to a running EJB engine.

The following explains how the countermod EJB module is deployed. The following assumes that:

- On JEUS, adminServer (MASTER) and server 1 (MS) are already running.
- The countermod module exists in the form of countermod.jar file or countermod directory under the JEUS_HOME/apphome directory.

The runtime of an EJB module is deployed using the console tool.

- **Using the Console Tool**

  The following explains how to deploy an EJB module by using the console.

  1. Run the console tool. For detailed information about the console tool, see "Local Commands" in JEUS Reference Guide.

     ```
     $ jeusadmin -host localhost:9736
     ```

  2. Enter user name and password.

     ```
     User name: administrator
     Password:
     Attempting to connect to localhost:9736.
     The connection has been established to Domain Administration Server adminServer
     in the domain domain1.
     JEUS8 Administration Tool
     To view help, use the 'help' command.
     [MASTER]domain1.adminServer>
     ```

  3. Deploy the countermod EJB module using the **deploy** command.

     ```
     [MASTER]domain1.adminServer> deploy countermod -servers server1
     ```

  4. In order to verify the deployment result, execute the following command. All the deployed applications, including the countermod module, are displayed.

     ```
     [MASTER]domain1.adminServer> application-info
     ```

3.4.2. Directory Structure of a Deployed EJB Module

After an EJB module has been deployed, its class files and configuration files will be distributed throughout the JEUS installation directory as shown in the following structure.

```
{JEUS_HOME}
   |--domain/<domain_name>/servers/<server_name>
       |--.workspace/deployed
            |--<application_id>
            |    |--<EJB_archive_file>_jar__
            |    |    |--META-INF
            |    |    |    |--[X]Jakarta EJB DD
            |    |    |    |--[X]JEUS EJB DD
            |    |    |--[C]Bussiness Interfaxe(Home Inferface, Component Interface)
            |    |    |--[C]EJB Implementation
            |    |    |--[C]Helper calss
            |    |--[J]<EJB_archive_file>
            |--<EAR_archive_file>_ear__
                 |--APP-INF
                 |--lib
                     |--META-INF
                     |--<EJB_archive_file>_jar__
                     |    |--META-INF
                     |    |    |--[X]Jakarta EJB DD
                     |    |    |--[X]JEUS EJB DD
                     |    |--[C]Bussiness Interfaxe(Home Inferface, Component Interface)
                     |    |--[C]EJB Implementation
                     |    |--[C]Helper calss
                     |--[J]<EJB_archive_file>

 * Legend
- [01]: binary or executable file
- [X] : XML document
- [J] : JAR file
- [T] : Text file
- [C] : Class file
- [V] : java source file
- [DD] : deployment descriptor
```

JEUS_HOME/domains/<domain_name>/servers/<server_name>/.workspace/deployed/ directory contains applications deployed to the MS specified by the <server name> option. It will contain deployed EJB implementation classes of the EJB module as well as helper classes.

Deployed files are placed in the following directories.

Deployment Mode	Description
EAR Deploy Mode	When EAR file is deployed, the modules are placed under the directory created with the EAR file name.
Standalone Deploy Mode	When JAR file is deployed, the file is placed under the "<jar file_name>_jar___" directory, which is under the root directory that has the JAR file name.
Exploded EAR Deploy Mode	Unpacked EAR file is deployed.

Deployment Mode	Description
Exploded Standalone Deploy Mode	Unpacked JAR file is deployed.

For Exploded EAR deployment and Exploded Standalone deployment, the reference to the original directory is used, instead of copying the files to the webhome directory. JEUS classifies deploying the packaged archive file as a **COMPONENT type** and deploying the directory with the unpacked files as an **EXPLODED COMPONENT type**.

> The following description only covers the standalone module. For a detailed explanation of the EAR deployment method or deployment in general, see the deployment related sections in "JEUS Applications & Deployment Guide".

3.5. Controlling and Monitoring EJB Modules

You can control and monitor the status of the deployed EJB modules by using the console tool. (refer to Flow Chart for Managing EJB Modules.)

3.5.1. Controlling EJB Modules

You can control the execution status within a deployed EJB module by using the console tool's stop-application, start-application, redeploy-application, and undeploy commands.

Using the Console Tool

The deployed EJB module can be controlled by using the stop-application, start-application, redeploy-application, and undeploy commands. All the commands take the EJB module ID as a parameter. For more information about the console tool, refer to "jeusadmin" in JEUS Reference Guide.

- **redeploy-application**

  Reloads the specified EJB module. This means that the EJB module that is currently running will be discarded from the EJB engine's runtime environment and then re-read or redeployed from the disk.

  When the EJB module is redeployed, all the transactions performed by the EJB module will be rolled back together.

  ```
  redeploy-application <application-id>
  ```

- **stop-application**

  Suspends the specified EJB module. This means that the EJB module that is currently running will

be temporarily inaccessible. The start-application command can make the EJB module accessible.

```
stop-application <application-id>
```

- **start-application**

  Activates the EJB module which was suspended with the stop-application command.

  ```
  start-application <application-id>
  ```

- **undeploy**

  Removes the selected EJB module from the EJB engine's runtime memory so that the EJB clients are not able to access the EJB. However, the physical file is not deleted.

  ```
  undeploy <application-id>
  ```

Following is a simple example illustrating how to undeploy an EJB module using the console tool:

1. Execute the console tool.

   ```
   $ jeusadmin -host localhost:9736
   ```

2. Enter user name and password.

   ```
   User name: administrator
   Password:
   Attempting to connect to localhost:9736.
   The connection has been established to Domain Administration Server adminServer
   in the domain domain1.
   JEUS8 Administration Tool
   To view help, use the 'help' command.
   [MASTER]domain1.adminServer>
   ```

3. To undeploy the countermod EJB module, enter the following:

   ```
   [MASTER]domain1.adminServer> undeploy countermod
   ```

4. To verify the undeployment result, enter the following. The countermod module is not included in the list of deployed modules that is displayed.

   ```
   [MASTER]domain1.adminServer> application-info
   ```

3.5.2. Monitoring EJB Modules

EJB modules can be monitored using the console tool. To monitor the status and runtime information of an EJB module, execute the application-info command in the console tool. For more information about the application-info command, refer to "EJB Engine Commands" in JEUS Reference Guide.

- **Monitoring module information**

    Displays information about the module.

    ```
    [MASTER]domain1.adminServer>application-info -server server1 -id countermod -detail
    General information about the EJB module [countermod].
    ============================================================
    +-------------+------------------------------------------+
    | Module Name |              Unique Module Name          |
    +-------------+------------------------------------------+
    | countermod  | countermod                               |
    +-------------+------------------------------------------+
    ============================================================

    Beans
    ================================================================
    
    +-----------+------------------------+-----------------+-------------------+
    | Bean Name |          Type          | Local Export Name | Remote Export Name |
    +-----------+------------------------+-----------------+-------------------+
    | Count     | StatelessSessionBean   |                 | Count             |
    +-----------+------------------------+-----------------+-------------------+
    
    ================================================================
    ```

- **Monitoring EJB list**

    Displays a list of EJBs of the relevant module("application-id").

    ```
    [MASTER]domain1.adminServer>application-info -server server1 -id countermod -bean Count
    Module name : countermod
    Bean name : Count
    ================================================================
    
    +---------------+----------+------------------+-------------+--------------+
    |     Name      | (Count)  | WaterMark(High:Low| Bound(Upper:L| Time(Max:Min:T|
    |               |          |       :Cur)      |    ower)     |     otal)     |
    +---------------+----------+------------------+-------------+--------------+
    | create        | times(0) |                  |             |              |
    +---------------+----------+------------------+-------------+--------------+
    | comitted      | transactio|                 |             |              |
    |               |n(0)       |                  |             |              |
    +---------------+----------+------------------+-------------+--------------+
    | total-remote-t|          |thread(100:100:100)|             |              |
    |hread          |          |                  |             |              |
    +---------------+----------+------------------+-------------+--------------+
    | timed-rb      | transactio|                 |             |              |
    ```

```
|               |n(0)      |                 |             |             |
+---------------+----------+-----------------+-------------+-------------+
| remove        | times(0) |                 |             |             |
+---------------+----------+-----------------+-------------+-------------+
| active-bean   |          | bean(0:0:0)     |             |             |
+---------------+----------+-----------------+-------------+-------------+
| request       | request(0)|                |             |             |
+---------------+----------+-----------------+-------------+-------------+
| total-bean    |          | bean(0:0:0)     |             |             |
+---------------+----------+-----------------+-------------+-------------+
| rolledback    | transactio|                |             |             |
|               |n(0)      |                 |             |             |
+---------------+----------+-----------------+-------------+-------------+
| active-thread |          | thread(0:0:0)   |             |             |
+---------------+----------+-----------------+-------------+-------------+
| MethodReadyCou|          | bean(0:0:0)     |             |             |
|nt             |          |                 |             |             |
+---------------+----------+-----------------+-------------+-------------+


================================================================================
```

EJB modules can be monitored in detail by using the console tool. For more information about EJB module monitoring, refer to "Checking Bound Objects" in JEUS Server Guide, and Timer Monitoring in this guide.

4. Common Characteristics of EJB

This chapter describes the general characteristics of EJBs in JEUS environment.

This chapter does not cover the EJB standard. It only provides information that is required to use EJBs in the JEUS environment. For information about EJB deployment, refer to EJB Modules.

4.1. Overview

This section discusses EJB types and their DD configurations.

EJB type

The following EJB types are supported in JEUS.

- Stateless Session Bean : Session Bean
- Stateful Session Bean : Session Bean
- BMP Entity Bean : Entity Bean
- CMP 1.1 & CMP 2.x Entity Bean : Entity Bean
- Message-Driven Bean : Message Driven Bean(MDB)

This chapter describes characteristics and configurations common to all bean types. In the previous list, you can find reference to additional information about each type.

> Since Entity Bean has been replaced by JPA since EJB 3.0 (refer to "JEUS JPA Guide" for information about JPA), it is not recommended to use it. Entity Bean is only supported for backward compatibility.

Deployment Descriptor Configuration by EJB Type

JEUS EJB Deployment Descriptor (hereafter JEUS EJB DD) is a generic XML file named jeus-ejb-dd.xml, and it is placed in the META-INF directory of an EJB JAR file. For more information about JEUS EJB DD, refer to EJB Modules.

The following table outlines the functions and components that can be configured in JEUS EJB DD ("@" indicates that the entity can be configured using annotation). The items in the leftmost column represent the actual configurable XML tags of a JEUS EJB DD. The next six columns show whether the component is applicable to each of the EJB types.

Configurable Item	Stateless	Stateful	BMP	CMP1.1	CMP2.0	MDB
1. EJB name @	√	√	√	√	√	√
2. Export name @	√	√	√	√	√	

Configurable Item	Stateless	Stateful	BMP	CMP1.1	CMP2.0	MDB
3. Local export name @	√	√	√	√	√	
4. Export port	√	√	√	√	√	
5. Export IIOP switch	√	√	√	√	√	
6. use-access-control	√	√	√	√	√	√
7. Run-as Identify	√	√	√	√	√	√
8. Security CSI Interop.	√	√	√	√	√	√
9. Env. Refs @	√	√	√	√	√	√
10. EJB refs @	√	√	√	√	√	√
11. Resource Refs @	√	√	√	√	√	√
12. Resource env. Refs @	√	√	√	√	√	√
13. Thread ticket pool settings	√	√`	√	√	√	√
14. Clustering settings @	√	√	√	√	√	
15. HTTP Invoke	√	√	√	√	√	
16. Pooling Bean Switch		√				
17. Object management	√	√	√	√	√	√
18. Persistence Optimize			√	√	√	
19. CM persistence opt.				√	√	
20. Schema info				√	√	
21. Relationship map					√	
22. Connect. fact. name						√
23. MDB resource adaptor						√
24. Destination @						√
25. JNDI SPI settings						√
26. Max messages						√
27. Activation Config @						√
28. Durable Timer Service	√		√	√	√	√

The previous table is partitioned as follows:

- Items 1 through 16 are common to all six EJB types (Items 2-5 and 14-16 do not apply to MDB). All these items are discussed in this chapter.

- Items 7 (run-as Identify), 8 (security interoperability), and 14 (clustering) are common to all EJB types, and are discussed in separate chapters titled Configuring EJB Security, EJB Interoperability and RMI/IIOP, and EJB Clustering respectively.

- Item 17 only applies to stateful session beans and are discussed in Session Bean.

- Item 18 (object management) is common to all six EJB types, but it varies slightly for each EJB type. Basic information about this item is discussed in Session Bean, and the differences in each chapter.

- Items 19 through 22 only apply to entity beans. They will be discussed in Entity Bean.

- Items 23 through 28 only apply to message-driven beans (MDB). MDBs are covered in Message Driven Bean(MDB).

- Item 29 is common to beans (all beans excluding stateful session bean) that support the Timer Service included in EJB specification. For more information, refer to the EJB Timer Service.

4.2. Configuring EJBs

All the EJB settings are configured in the jeus-ejb-dd.xml file, the JEUS EJB DD file. The parent XML tag is <jeus-bean> for all bean types.

The next section will describe settings applicable to XML parent tags for the five EJB types. Other tags outside the <beanlist> have already been discussed in EJB Modules. For more information about specific XML parent tags (bean type), refer to the next chapters in this guide.

> For information about clustering settings, refer to EJB Clustering.

4.2.1. Configuring the Basic Environment

The following settings are only for JEUS, and are defined in the jeus-ejb-dd.xml file. For more information about the file, refer to EJB Modules.

- **<ejb-name>**

  - It uses the <ejb-name> value from the ejb-jar.xml file, or the "name" value of @Stateless, @Stateful, and @MessageDriven annotations in the implementation class.

- **<export-name>**

  - It has to have a unique name within the system to be registered to JNDI.

  - If this element is not defined in the DDs, either the mappedName attribute value of @Stateless or @Stateful annotation, or the <mapped-name> value in ejb-jar.xml is applied.

    If none is specified, a Remote Business Interface name is used as the default value. But the default value is applied only when there is one Remote Business interface and no Remote Home Interface, or vise versa. Otherwise, the deployment fails because the JNDI name cannot be determined.

  - The priority is as follows:

    1. <export-name> of jeus-ejb-dd.xml

    2. <mapped-name> of ejb-jar

3. mappedName of @EJB

- **\<local-export-name>**

    ◦ It has to have a unique name within the system to be registered to JNDI.

    ◦ It is applied to the mappedName attribute of @EJB annotation when this element is not configured in DD. If remote business interface exists, '_Local' is attached to this value.

    If neither exists, the local business interface name is used as the default value. But the default value is applied only when there is one Local Business interface and no Local Home Interfaces, or vise versa. Otherwise, an exception occurs because the JNDI name cannot be found.

    ◦ The priority is as follows:

        1. \<local-export-name> of jeus-ejb-dd.xml

        2. If a remote business interface exists, \<mapped-name>+ _Local of ejb-jar.xml

            If a remote business interface does not exist, \<mapped-name> of ejb-jar.xml

        3. If a remote business interface exists, mappedName + _Local of @EJB

            If a remote business interface does not exist, mappedName of @EJB

- **\<export-port>**

    ◦ This setting is used when RMI service port needs to be specified explicitly.

    ◦ It is useful when system access is possible only through a certain port because of a firewall. The value of this tag should be same as the firewall's allowed port. If a value is not specified, ports are applied according to the system properties as follows:

Tag	Description
jeus.ejb.export Port	Determines export ports of EJBs whose \<export-port> is not specified for a specific EJB in jeus-ejb-dd.
jeus.rmi.usebas eport	◦ true: Indicates MS base port is used. ◦ false: Indicates MS base port + 7 is used.

- **\<export-iiop>**

    ◦ If this setting is activated, it enables the interfaces of the bean to be exported to COS naming server as IIOP stubs and skeletons. This enables all clients, who can access IIOP, to access the bean.

- **\<use-access-control>**

    ◦ This Boolean element determines whether a method execution is monitored by Java SE security manager according to Jakarta EE principal. For more information, refer to "JEUS Security Guide" and JACC specification (default: false).

The following examples show how the previous settings can be applied to the session bean classes and DDs. Elements of corresponding annotations and DDs are displayed in bold.

Stateful Session Bean Class and DD: <CounterEJB.java>

```java
package ejb.basic.statefulSession;

@Stateful(name="counter", mappedName="counterEJB")
public class CounterEJB implements Counter, CounterLocal
{
   private int count = 0;

   public void increase()
   {
      count++;
   }
 ...
}
```

Stateful Session Bean Class and DD: <jeus-ejb-dd.xml>

```xml
<jeus-ejb-dd>
    . . .
    <beanlist>
        <jeus-bean>
            <ejb-name>counter</ejb-name>
            <export-name>counterEJB</export-name>
            <local-export-name>counterEJB_Local</local-export-name>
            <export-port>7654</export-port>
            <export-iiop/>
            . . .
        </jeus-bean>
        . . .
    </beanlist>
    . . .
</jeus-ejb-dd>
```

4.2.2. Configuring Thread Ticket Pools

The following figure illustrates the concept of EJB thread ticket pool (hereafter TTP).

If there is a client request, it is assigned with a new RMI thread. If the number of assigned threads is greater than the maximum value, the request waits for an available thread.

Transition format:
<transition name>
[<guard cond.(Boolean cond.)>]
/ <Actions performed>
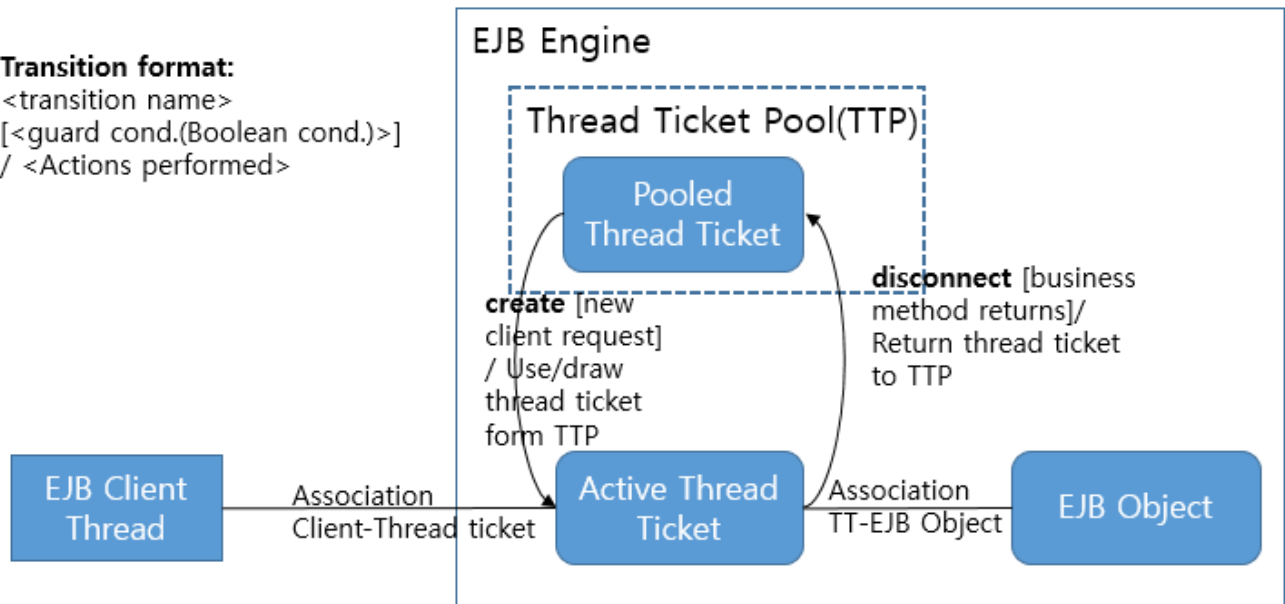
TTP State Chart

As you can see in the figure, a thread pool contains thread tickets (hereafter TT). One TT permits one client thread to access the EJB engine. When a client request arrives, one TT is drawn from the TP and assigned to the client. This implies that each client thread will be assigned to one EJB object that will relay the request to an actual EJB instance. An EJB object is thus an object that lives in the EJB engine and manages the connection between an EJB client and an actual EJB instance.

If more client requests arrive than there are TTs in the thread ticket pool, the new requests must wait until a TT is returned to the pool. If the waiting period exceeds ten minutes, it throws a RemoteException due to a timeout.

TTs are returned to TTP when the client gets disconnected after an EJB process terminates (for stateless session bean), or a connection timeout occurs. Each remote-call enabled EJB component contains a TTP. There are also connection and bean pools. A bean pool, instead of a TTP, should be configured for an MDB.

> For more information about connection and bean pools, refer to Session Bean and Entity Bean. For more information about using bean pool in MDBs, refer to Message Driven Bean(MDB).

TT of EJB is configured in the **<thread-max>** element of the root XML tag of the bean. This value represents the maximum number of allowable threads that can operate in each EJB, in other words, the maximum number of TT. The default value is 100. If the number of client requests in a queue is larger than this value, new requests have to wait until a running job is finished and the TT is returned to the pool. If the waiting period exceeds ten minutes, it throws a RemoteException due to a timeout.

The following shows how to configure a BMP bean as above.

Configuring a BMP Bean: <jeus-ejb-dd.xml>

```
<jeus-ejb-dd>
    . . .
```

```
    <beanlist>
        <jeus-bean>
            . . .
            <thread-max>200</thread-max>
            . . .
        </jeus-bean>
        . . .
    </beanlist>
    . . .
</jeus-ejb-dd>
```

4.2.3. Configuring and Mapping External References

External references that the bean uses can be configured through annotation or JNDI export name in jeus-ejb-dd.xml, which is mapped to the reference name used in ejb-jar.xml. In the actual system, all defined external references must be mapped to JNDI export name. Annotations or all the external references defined in ejb-jar.xml must be mapped to relevant tags in jeus-ejb-dd.xml using the JNDI export-name used in the actual system.

The following describes the reference tags of ejb-jar.xml and JEUS EJB DD files.

ejb-jar.xml Tag	jeus-ejb-dd.xml Tag	Description
<env-entry>	<env>	Defines or ignores the EJB environment configuration.
<ejb-ref>	<ejb-ref>	Binds an EJB reference to an actual export name.
<resource-ref>	<res-ref>	Binds an external resource manage reference to an actual export name.
<resource-env-ref>	<res-env-ref>	Binds a managed object reference to an actual export name.
<message-destination-ref>	<message-destination-ref>	Binds a message destination reference to an actual export name.
<service-ref>	<service-ref>	Binds a webservice endpoint reference to an actual export name.

The following child tags must be configured for every <env> tag has.

Tag	Description
<name>	Name of env. Same value as the <env-entry-name> defined in the ejb-jar.xml.
<type>	Full Java class name corresponding to the data type of the value. Basic data types use a wrapper class.
<value>	Specifies actual value of env.

Other three tags (<ejb-ref>, <res-ref>, and <res-env-ref>) have several <jndi-info> child tags. These tags, in turn, have to set the following child tags to map JNDI export name to reference.

Tag	Description
<ref-name>	Reference name defined in EJB source code and ejb-jar.xml. <ejb-ref>, <res-ref> and <res-env-ref> are equivalent to <ejb-ref>, <resource-ref> and <resource-env-ref>, respectively in ejb-jar.xml.
<export-name>	Used when an entity or object is exported to JNDI.

<ref-name> and <export-name> tags must be same with the names defined in ejb-jar.xml.

The following is an example of annotation and XML that illustrates how to map external references to JNDI.

Mapping External Reference to JNDI: <CounterEJB.java>

```
@Stateful

@EJB(name="AccountEJB", beanInterface=ejb.Account.class, mappedName="ACCEJB")
@Resource(name="jdbc/DBDS", type=javax.sql.DataSource.class, mappedName="ACCOUNTDB")
public class CounterEJB implements Counter{
    @Resource(name="minAmount")
    private int minAccount = 100;

 . . .
}
```

Mapping External Reference to JNDI: <ejb-jar.xml>

```
<ejb-jar.xml>
    . . .
    <enterprise-beans>
        <session>
            . . .
            <env-entry>
                <env-entry-name>minAmount</env-entry-name>
                <env-entry-type>java.lang.Integer</env-entry-type>
            </env-entry>
            <ejb-ref>
                <ejb-ref-name>AccountEJB</ejb-ref-name>
                <ejb-ref-type>Session<ejb-ref-type>
                <remote>ejb.Account</remote>
            </ejb-ref>
            <resource-ref>
                <res-ref-name>jdbc/DBDS</res-ref-name>
                <res-ref-type>javax.sql.DataSource</res-ref-type>
            </resource-ref>
            . . .
        </session>
    </enterprise-beans>
    . . .
</ejb-jar.xml>
```

Mapping External Reference to JNDI: <jeus-ejb-dd.xml>

```
<jeus-ejb-dd>
    . . .
    <beanlist>
        <jeus-bean>
            . . .
            <env>
                <name>minAmount</name>
                <type>java.lang.Integer</type>
                <value>100</value>
            </env>
            <ejb-ref>
                <jndi-info>
                    <ref-name>AccountEJB</ref-name>
                    <export-name>ACCEJB</export-name>
                </jndi-info>
            </ejb-ref>
            <res-ref>
                <jndi-info>
                    <ref-name>jdbc/DBDS</ref-name>
                    <export-name>ACCOUNTDB</export-name>
                </jndi-info>
            </res-ref>
            . . .
        </jeus-bean>
        . . .
    </beanlist>
    . . .
</jeus-ejb-dd>
```

4.2.4. Configuring the HTTP Invocation Environment

In order to set up HTTP invocation on a specific EJB module, you need to configure <invoke-http> in the <jeus-bean> element of jeus-ejb-dd.xml. If an EJB engine needs HTTP invocation, add the <invoke-http> tag to the ejb-engine in the jeus-ejb-dd.xml file (mentioned in EJB Engines). Settings in the domain.xml file will be applied to all modules, but if the DD of each module contains the <invoke-http> setting, the setting in DD file is used instead.

The following example configures the HTTP invoke environment properties in jeus-ejb-dd.xml.

Configuring HTTP Invocation: <jeus-ejb-dd.xml>

```
<jeus-ejb-dd>
    . . .
    <beanlist>
        <jeus-bean>
            . . .
            <invoke-http>
                <url>
                    /mycontext/RMIServletHandler
                </url>
                <http-port>
                    80
                </http-port>
```

```
            </invoke-http>
            . . .
        </jeus-bean>
    </beanlist>
    . . .
</jeus-ejb-dd>
```

The <invoke-http> tag has the following two child tags.

Tag	Description
<url>	URI of the RMI Handler Servlet (jeus.rmi.http.ServletHandler) that is called from the HTTP-RMI stub. Enter only the request path of the Servlet excluding the protocol, IP, and port. (required) Protocol and IP are assumed to be the same as HTTP and RMI runtime, respectively. This means the Web server and Web engine that received HTTP-RMI request should be on the same machine with the RMI runtime. This way, the RMI runtime address is known to the RMI stub. Web server port should be specified in the following <http-port> element.
<http-port>	Port to receive HTTP-RMI requests. The RMI Handler Servlet must be deployed and executed on the Web engine.

> The RMI handler servlet must be deployed to successfully execute an HTTP invoke. The class to implement the RMI handler servlet is jeus.rmi.http.ServletHandler. The jeus.rmi.http.ServletHandler must be deployed to the URL specified in the <invoke-http> element of the context. For more information about deployment, refer to "JEUS Web Engine Guide".

4.2.5. Configuring EJB Security

This section describes how to configure security for EJB in JEUS. The security refers to certification and authorization.

JEUS EJB security configuration includes the following settings.

- Role assignment of EJB modules

- EJB run-as Identify setting

> The actual JEUS user list used in EJB settings is defined in accounts.xml. For information about the settings, refer to "JEUS Security Guide".

4.2.5.1. Security Settings

The following explains how to configure security settings for each case.

Configuring Role Assignment

The role assignment setting of EJB is mapping developer-defined role of the ejb-jar.xml file with principal and group name of the actual JEUS security domain. Configure this setting in the **<role-permission>** element inside the **<module-info>** element of the jeus-ejb-dd.xml file.

The following is an example of configuring a role in XML.

Configuring Role Assignment: <jeus-ejb-dd.xml>

```
<jeus-ejb-dd>
    <module-info>
        <role-permission>
            <role>manager</role>
            <principal>peter</principal>
        </role-permission>
    </module-info>
    <beanlist>
      . . .
    </beanlist>
    . . .
</jeus-ejb-dd>
```

The following are two required child tags:

Tag	Description
<role>	Specifies the logical role name defined in the EJB (through annotation or ejb-jar.xml file).
<principal>	Specifies actual user name defined in the accounts.xml file. A role name and user name pair defines the mapping between a developer-defined role and JEUS principal. A role can be mapped to multiple user names.

Configuring Run-as Identify

Run-as identity setting provides the actual mapping of developer-defined role name in the <run-as> tag of the ejb-jar.xml file with the actual principal specified in the JEUS security domain configuration (e.g., in the accouts.xml file).

This principal, in general, uses a simple user name. This setting is configured in the **<run-as-identity>** tag, which is at the same level as the bean configuration in jeus-ejb-dd.xml.

The following is an example of XML setting for run-as identity.

Configuring Run-as Identify: <jeus-ejb-dd.xml>

```
<jeus-ejb-dd>
    . . .
    <beanlist>
        <jeus-bean>
            . . .
<run-as-identity>
                <principal-name>peter</principal-name>
            </run-as-identity>
            . . .
        </jeus-bean>
        . . .
    </beanlist>
    . . .
</jeus-ejb-dd>
```

Tag	Description
<principal-name>	Specifies the user name to map with role in the <run-as> element of the ejb-jar.xml file. This user name should be specified in the accounts.xml file.

4.2.5.2. Security Setting Example

The following examples show how the role of role-permission and role of run-as are mapped to an actual JEUS user.

Some parts of Java classes and XML are taken from three configuration files (ejb-jar.xml, jeus-ejb-dd.xml, and accounts.xml). The identical parts in these files are highlighted in bold.

Security Setting: <CustomerBean.java>

```
@Stateless(name="CustomerEJB")
@RolesAllowed("CUSTOMER")
public class CustomerBean implementes Customer {
    ...
}
```

Security Setting: <EmployeeServiceBean.java>

```
@Stateful(name="EmployeeService")
@RunAs("ADMIN")
public class EmployeeServiceBean implements EmployService{
    ...
}
```

Security Setting: <ejb-jar.xml>

```
<?xml version="1.0"?>
<ejb-jar version="4.0" xmlns="https://jakarta.ee/xml/ns/jakartaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="https://jakarta.ee/xml/ns/jakartaee
https://jakarta.ee/xml/ns/jakartaee/ejb-jar_4_0.xsd">
```

```
    <enterprise-beans>
        <session>
            <ejb-name>CustomerEJB</ejb-name>
            . . .
<security-role-ref>
                <role-name>CUSTOMER</role-name>
            </security-role-ref>
            <security-identity>
                <use-caller-identity />
            </security-identity>
        </session>
        <session>
            <ejb-name>EmployeeService</ejb-name>
            . . .
            <security-identity>
<run-as>
                <role-name>ADMIN</role-name>
                </run-as>
            </security-identity>
        </session>
        . . .
    </enterprise-beans>
    <assembly-descriptor>
<security-role>
            <role-name>CUSTOMER</role-name>
        </security-role>
        <method-permission>
            <role-name>CUSTOMER</role-name>
            <method>
                <ejb-name>CustomerEJB</ejb-name>
                <method-name>*</method-name>
                <method-params />
            </method>
        </method-permission>
        . . .
    </assembly-descriptor>
    . . .
</ejb-jar>
```

The previous example defines two session beans.

- The first bean (CustomerEJB) has a security role connected to the CUSTOMER role.

- The second session bean (EmployeeService) defines the <run-as> role as ADMIN. The roles
  CUSTOMER and ADMIN should be mapped to actual system Principals when EJB is deployed.

The following example shows how the roles of CUSTOMER and ADMIN are mapped to the actual
system principal names (customer and peter, respectively).

Security Setting: <jeus-ejb-dd.xml>

```
<?xml version="1.0" encoding="UTF-8"?>
<jeus-ejb-dd xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
    <module-info>
        . . .
<role-permission>
            <principal>customer</principal>
```

```
            <role>CUSTOMER</role>
        </role-permission>
    </module-info>
    <beanlist>
        <jeus-bean>
            <ejb-name>EmployeeService</ejb-name>
 <run-as-identity>
                <principal-name>peter</principal-name>
            </run-as-identity>
        </jeus-bean>
        . . .
    </beanlist>
    . . .
</jeus-ejb-dd>
```

The following example shows how these users are configured in the accounts.xml file.

Security Setting: <accounts.xml>

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<accounts xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
    <users>
        <user>
            <name>customer</ name>
            <password>{SHA}blSUPYpdjb8QDcq+ozfbIEZx5oY=</name>
            <group>test</group>
        </user>
        <user>
            <name>peter</ name>
            <password>{SHA}McbQlyhI3yiOG1HGTg8DQVWkyhg=</name>
            <group>test</group>
        </user>
    </users>
</accounts>
```

The previous example shows definition for the users, customer and peter. These users are connected to the roles, CUSTOMER and ADMIN, of the ejb-jar.xml file through the jeus-ejbdd.xml file.

4.3. Monitoring EJBs

JEUS does not provide a method to control individual EJBs included in the EJB module. However, you can monitor each bean by using the console tool.

In the console tool, information about each EJB, including bean name, export name, and status, can be displayed using the application-info command. The following example is based on the assumption that an EJB called 'countermod' has been deployed. For more information about the application-info command, refer to "application-info" in JEUS Reference Guide.

As the result of the previous command, the name and export name of each bean registered to the EJB module are displayed.

```
[MASTER]domain1.adminServer>application-info -server server1 -id countermod -detail
General information about the EJB module [countermod].
============================================================
+-------------+---------------------------------------------+
| Module Name |               Unique Module Name            |
+-------------+---------------------------------------------+
| countermod  | countermod                                  |
+-------------+---------------------------------------------+

============================================================

Beans
===========================================================================
+-----------+----------------------+------------------+-------------------+
| Bean Name |         Type         | Local Export Name | Remote Export Name |
+-----------+----------------------+------------------+-------------------+
| Count     | StatelessSessionBean  |                  | Count             |
+-----------+----------------------+------------------+-------------------+
===========================================================================
```

The status of each bean in EJB module can be monitored with the following command.

```
[MASTER]domain1.adminServer>application-info -server server1 -id countermod -bean Count
Module name : countermod
Bean name : Count
===========================================================================
+--------------+----------+------------------+------------+--------------+
|     Name     | (Count)  | WaterMark(High:Low| Bound(Upper:L| Time(Max:Min:T|
|              |          |      :Cur)       |    ower)   |     otal)    |
+--------------+----------+------------------+------------+--------------+
| create       | times(0) |                  |            |              |
+--------------+----------+------------------+------------+--------------+
| comitted     | transactio|                 |            |              |
|              | n(0)     |                  |            |              |
+--------------+----------+------------------+------------+--------------+
| total-remote-t|          |thread(100:100:100)|           |              |
|hread         |          |                  |            |              |
+--------------+----------+------------------+------------+--------------+
| timed-rb     | transactio|                 |            |              |
|              | n(0)     |                  |            |              |
+--------------+----------+------------------+------------+--------------+
| remove       | times(0) |                  |            |              |
+--------------+----------+------------------+------------+--------------+
| active-bean  |          | bean(0:0:0)      |            |              |
+--------------+----------+------------------+------------+--------------+
| request      | request(0)|                 |            |              |
+--------------+----------+------------------+------------+--------------+
| total-bean   |          | bean(0:0:0)      |            |              |
+--------------+----------+------------------+------------+--------------+
| rolledback   | transactio|                 |            |              |
|              | n(0)     |                  |            |              |
+--------------+----------+------------------+------------+--------------+
| active-thread |          | thread(0:0:0)    |            |              |
+--------------+----------+------------------+------------+--------------+
| MethodReadyCou|          | bean(0:0:0)      |            |              |
|nt            |          |                  |            |              |
+--------------+----------+------------------+------------+--------------+
```

4.4. Tuning EJBs

There are some settings that can be uniformly applied to all EJB types in order to enhance EJB operation performance (or to prevent system resources from being wasted):

- Adjust the max value of TTP.

- Configure beans after clustering beans from several EJB engines. Refer to EJB Clustering.

- Set up an appropriate JDBC connection pool. For more information about the settings, refer to "JEUS Server Guide".

- For EJB 2.x, use the -fast option and the appcompiler to save the deployment time of EJB modules. Refer to EJB Modules for more information.

There are many tuning tips for each bean type besides the already mentioned basic optimization options. Therefore, look up the corresponding chapter for optimization and tuning method recommended for each EJB type.

4.4.1. Configuring Thread Ticket Pools

The TTP should be configured correctly for an EJB to exhibit the best performance. You can apply the following principles for this.

- The larger the specified max pool size, the more TTs can be generated and more client requests can be processed. However, a large max pool size consumes more memory resources.

- If the number of clients who request a service of a bean increases rapidly at a specific time, increase the pool size by subtracting the min pool size from the max pool size. This enables the new TTs to be created in batches.

TTP setting acts as a valve that controls the volume of accesses to a certain EJB.

> The max pool size is the most important parameter in the thread ticket pool. Therefore, this value should be adjusted and tuned first to improve performance. Set this value to <thread-max>.

5. Interoperability and RMI-IIOP

This chapter describes interoperability and RMI-IIOP protocol used by EJB.

5.1. Overview

JEUS supports RMI-IIOP protocol used to invoke EJB from a different WAS or system. RMI-IIOP is based on Common Object Request Broker Architecture (CORBA) that is defined by OMG (http://www.omg.org/), and EJB specification standard for interoperability. Consider using the RMI-IIOP interoperability function to support remote calls within JEUS or from a different WAS. The RMI-IIOP remote call method is one of the methods that support calls from a different WAS or system. There are other methods that call EJB on JEUS from a different WAS and integrates the transaction, or vice versa, but this chapter only covers the RMI-IIOP method.

Interoperability can mean calling EJB on JEUS from another system and calling EJB on WAS from JEUS, which means JEUS can be both a RMI-IIOP server and client. There needs to be runtime support for Object Request Broker (ORB), and support for transaction and security interoperability.

JEUS contains ORB runtime which automatically creates dynamic stub in memory without using COS Naming Server or stub classes that support the CORBA Naming Service. It implements the Object Transaction Service (OTS) specification for transaction interoperability and the Common Secure Interoperability version 2 (CSIv2) specification for security interoperability.

For more information about CORBA and RMI-IIOP, refer to the following links.

- CORBA 2.3.1 Specification
- Glossary of Java IDL Terms
- Java RMI over IIOP
- OTS 1.2.1 Specification
- CSIv2 Specification

5.1.1. Transaction Interoperability (OTS)

Transaction interoperability is implemented based on the OTS specification. The object mentioned here is the CORBA Object, which is affected by the transaction.

When using OTS, an interceptor (a listener type) which can process transactions is added to CORBA. This additional interceptor handles the transaction related part of the IIOP protocol header. Through this, JEUS, which plays a role of a client or server, supports transaction propagation between different remote systems such as WAS or JEUS.

> OTS specification 1.2 is the currently supported version.

5.1.2. Security Interoperability (CSIv2)

Common Secure Interoperability version 2 (CSIv2) specification guarantees security interoperability. If this CSI is used, like the OTS, an interceptor that can process security is added to ORB, and this interceptor handles the header and other security related parts.

Security interoperability starts by attaching security information for CSI to Interoperable Object Reference (IOR) of EJB reference. If IOR is registered to a naming service, the client side ORB that is trying to used it and the JEUS MS handles negotiation on the security level of the IIOP protocol. JEUS MS acts as a client when it needs CSI Services to call another EJB bean.

> For more information about security interoperability, refer to CSIv2 specification of CORBA.

5.2. Configuring Interoperability

The interoperability must be configured for use because it is not configured by default.

In order to expose JEUS EJB to RMI-IIOP, activate the Enable Interop option of COS Naming Service and MS. EJB should be deployed with IIOP export related settings configured. If JEUS is only used as a RMI-IIOP client, only the 'Enable Interop' option of the relevant MS needs to be activated. The interoperability settings can be configured in domain.xml.

5.2.1. COS Naming Service Configuration

Besides the basic JNDI Naming Service provided by MS, the COS Naming Service for CORBA objects can also be added. An EJB reference (stub) exposed to RMI-IIOP is saved in the COS Naming Service, and an EJB client looks up the reference and uses it. The COS Naming Service operates in the JEUS Manager JVM and uses **'BASEPORT + 4 (e.g. 9740)'** as the service port.

5.2.2. Activating Interoperability

In order to use interoperability needed for RMI-IIOP clients and servers, interoperability property must be activated. This must be done to initialize the ORB.

To activate the interoperability property, set <enable-interop/> in domain.xml. For more information, see "Configuring domain.xml settings" in JEUS XML Reference Guide.

For more information about CSIvs, refer to Configuring the CSIv2 Security Interoperability.

5.2.3. Configuring the CSIv2 Security Interoperability

In order to perform CSIv2 protocol, use the information from two additional security environment

files and JEUS Security Manager. The two additional security environment files are **keystore** and **truststore**. The paths and necessary information of these two files can be configured through the the system properties.

Security Environment File	Description
Keystore file	File that stores keys for X.509 and implements Oracle's X.509 keystore. It is sent to the client when Secure Socket Layer (SSL) is called.
Truststore file	Certificate configuration file for the client side X.509. The format is same as the keystore file.

CSIv2-related settings are specified in the <interop-ssl-config> tag in domain.xml. For more information, refer to "Configuring domain.xml settings" in JEUS XML Reference Guide.

Configuring CSIv2: <domain.xml>

```
<enable-interop>
    <interop-ssl-config>
        <keystore-path>/jeus/domains/domain1/ssl</keystore-path>
        <keystore-alias>changeit</keystore-alias>
        <keystore-password>changeit</keystore-password>
        <keystore-keypassword>{DES}FQrLbQ/D8O1l</keystore-keypassword>
        <truststore-path>/jeus/domains/domain1/ssl<</truststore-path>
        <truststore-password>changeit</truststore-password>
    </interop-ssl-config>
</enabpe-interop>
```

CSIv2 settings can also be configured through the system property of the execution script. **Nevertheless, settings in domain.xml take precedence.**

The following are options for this setting. Set this value by using "-D" option.

Parameter	Description
jeus.ssl.keystore	Absolute path to the keystore file. (Default value: DOMAIN_HOME/config/keystore)
jeus.ssl.keypass	Password for the keystore file. (Default value: jeuskeypass)
jeus.ssl.truststore	Absolute path to the truststore file. (Default value: DOMAIN_HOME/config/truststore)
jeus.ssl.trustpass	Password for the truststore file. (Default value: jeustrustpass)

Authentication and authorization may not be needed between trusted nodes.

By setting the IP address to jeus.ejb.csi.trusthosts, you can prevent any unnecessary security checks between trusted nodes. JEUS Security Manager uses the guest user, which represents an anonymous principal for anonymous user access.

5.2.4. Configuring EJB RMI-IIOP

In order to expose EJB to RMI-IIOP, configure the **<export-iiop>** option for each EJB in jeus-ejb-dd.xml. If the <export-iiop> option is set, EJB Home Reference is registered to a COS Naming Service using the given <export-name>. This way, EJB Reference can be fetched and used through the COS Naming Service.

The following is an example of the jeus-ejb-dd.xml file in which EJB RMI-IIOP is configured.

Configuring EJB RMI-IIOP - <jeus-ejb-dd.xml>

```
...
<jeus-bean>
    <ejb-name>CalcEJB</ejb-name>
    <export-name>ejb/Calc</export-name>
<export-iiop>
      <only-iiop>true</only-iiop>
  </export-iiop>
</jeus-bean>
...
```

Tag	Description
<only-iiop>	Specifies whether to expose EJB through RMI and RMI-IIOP or only through RMI-IIOP. If it is exposed through both, RMI/IIOP stub gets registered to the COS Naming Service and the general RMI stub gets registered to the JEUS Naming Server.

> Only EJB Home and EJB Object Reference can be currently exposed through RMI-IIOP. EJB 3.0 business views cannot be exposed through RMI-IIOP.

5.3. RMI/IIOP Client

RMI-IIOP clients can be standalone clients, or WAS from another JEUS or another vendor. This section describes the methods for using EJB.

5.3.1. JEUS Managed Server

The following is the method for calling RMI-IIOP EJB on JEUS or another vendor's WAS from applications running on a JEUS MS.

First, COS Naming Server looks up EJB Home Reference (Stub). The Naming Server can get it directly by using the JNDI APIs corbaname URL as follows:

Using Corbaname Lookup

```
    InitialContext ctx = new InitialContext();
```

```
    Object obj = ctx.lookup("corbaname:iiop:1.2@192.168.11.22:9740#ejb/Calc");

    CalcHome home = (CalcHome)PortableRemoteObject.narrow(obj, CalcHome.class);
    Calc calcRef = home.create();
```

URL has the form of "corbaname:iiop:1.2@<host>:<port>#<name>", and <host> and <port> are the address of the COS Naming Server. Also, by directly specifying the COS Naming Server as the PROVIDER URL, you can get and use the InitialContext.

Since there are multiple methods to specify the PROVIDER URL, you should specify one to use as in the following example.

Using PROVIDER URL

```
    Hashtable env = new Hashtable();
    env.put(Context.PROVIDER_URL, "iiop://192.168.11.22:9740");
 // env.put(Context.PROVIDER_URL, "iiopname://192.168.11.22:9740");
 // env.put(Context.PROVIDER_URL, "corbaname:iiop:1.2@192.168.11.22:9740");
 // env.put(Context.PROVIDER_URL, "corbaloc:iiop:1.2@192.168.11.22:9740");

    InitialContext ctx = new InitialContext(env);
    Object obj = ctx.lookup("ejb/Calc");

    CalcHome home = (CalcHome)PortableRemoteObject.narrow(obj, CalcHome.class);
    Calc calcRef = home.create();
```

For clients using dependency injection or a logical JNDI name in the form of "java:comp/env/", the <export-name> option should be mapped to the previous corbaname URL in the JEUS application DD file, such as the jeus-web-dd.xml file.

Servlet EJB Injection

```
 @EJB(name="ejb/Calc")
 private CalcHome calcHome;
```

RMI-IIOP EJB Mapping: <jeus-web-dd.xml>

```
 <ejb-ref>
     <jndi-info>
         <ref-name>ejb/Calc</ref-name>
         <export-name>corbaname:iiop:1.2@192.168.11.22:9740#ejb/Calc</export-name>
     </jndi-info>
 </ejb-ref>
```

5.3.2. Other Vendors' WASs

The method to call RMI-IIOP EJB of JEUS from applications running on another vendor's WAS is similar to the method described in JEUS Managed Server. In this case, check whether ORB configuration is needed like in JEUS, and configure it if necessary. For more information, refer to the relevant product documentation.

5.3.3. Standalone Clients

Since ORB runtime and related settings are not configured for standalone clients, you need to add the jclient.jar client library, as well as the omgapi.jar and peorb.jar files to the class path. You must also add the jeus.client.interop=true system property.

The method to obtain the EJB Home Reference (Stub) is similar to that in JEUS Managed Server. However, you should also configure the JNDI provider from JEUS.

Using Standalone Client

```
Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY, "jeus.jndi.JNSContextFactory");
env.put(Context.PROVIDER_URL, "iiop://192.168.11.22:9740");

InitialContext ctx = new InitialContext(env);
Object obj = ctx.lookup("ejb/Calc");

CalcHome home = (CalcHome)PortableRemoteObject.narrow(obj, CalcHome.class);
Calc calcRef = home.create();
```

5.4. Known Issues

Some known issues are as follows:

- **Continued retries to access a server for one minute when server access fails**

  ORB embedded in JEUS performs retries to access a server for 60 seconds when it cannot access the server. A separate sleep time is not used in this case. Therefore, too many log messages may occur or CPU may experience extreme overload. To prevent this, the following option should be set as a system property of a client (unit: ms, the default value: 60,000).

  ```
  com.sun.corba.ee.transport.ORBCommunicationsRetryTimeout=1
  ```

- **NullPointerException occurs after PortableRemoteObject.narrow() is invoked**

  When a client executes the following, home object is returned as null and the NullPointerException may occur.

  NullPointerException Occurrence

  ```
  CalcHome home = (CalcHome)PortableRemoteObject.narrow(obj, CalcHome.class);
  // null is returned
  Calc calcRef = home.create(); // NullPointerException
  ```

  This exception is thrown when CORBA stub class can not be found. This occurs when the **Enable Interop** setting is incorrectly configured in JEUS MS. If it is correctly configured, a correct ORB is initialized and Stub is automatically created in memory. For external clients, this error may occur

if there is no function to create dynamic stub in memory.

For standalone clients, check whether the following is configured in the system property.

```
jeus.client.interop=true
```

- **Unable to find EJB with a name that includes a period (.) in WebLogic**

  If <export-name> is specified with a value that includes a period (.) like in "com.acme.CalcHome", lookup fails in WebLogic. Because WebLogic handles a period (.) like a slash (/), it forwards the request as "com/acme/CalcHome". In this case, you must not use a name that includes a period (.).

6. EJB Clustering

This chapter describes the concept of EJB clustering and how to configure main functions of EJB.

6.1. Overview

In order to use the failover and load balancing functions in EJB, individual beans may be deployed across several EJB engines to form a cluster. Clustering is accomplished on the component level (individual beans) and may be used by Stateless/Stateful session beans and entity beans. A Message Driven Bean (MDB) is not a clustering target.

JEUS EJB clustering has the following two functions:

- Load Balancing

  Improves the overall response speed by distributing EJB requests.

- Failover

  If an EJB instance or EJB engine stops before or during executing a bean method, the request is handled by an instance of another EJB engine.

The following figure explains two major functions of EJB clustering, load balancing and failover.
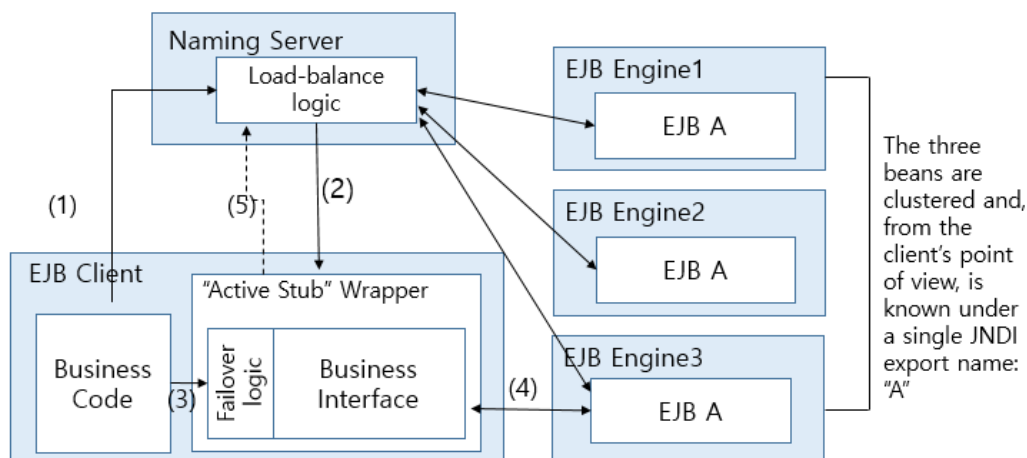
(1) Client looks up EJB cluster named "A"
(2) One of the EJB engines is selected (In this case engine3) and a wrapper stub is returned.
(3) Client interacts with the standard EJB interfaces of EJB engine3.
(4) The Wrapped EJB interfaces is communicating with bean in engine 3(as usual).
(5) In case engine 3 fails. The active stub will attempt to look up the same bean in a different engine.

EJB Clustering Architecture

(1) Client looks up EJB cluster named "A"
(2) One of the EJB engines is selected (In this case engine3) and a wrapper stub is returned.
(3) Client interacts with the standard EJB interfaces of EJB engine3.
(4) Stub Endpoints in the wrapped EJB interface is communicating with Engine3 (as usual).
(5) In case engine 3 fails, the active stub will select another Endpoint of Engine.
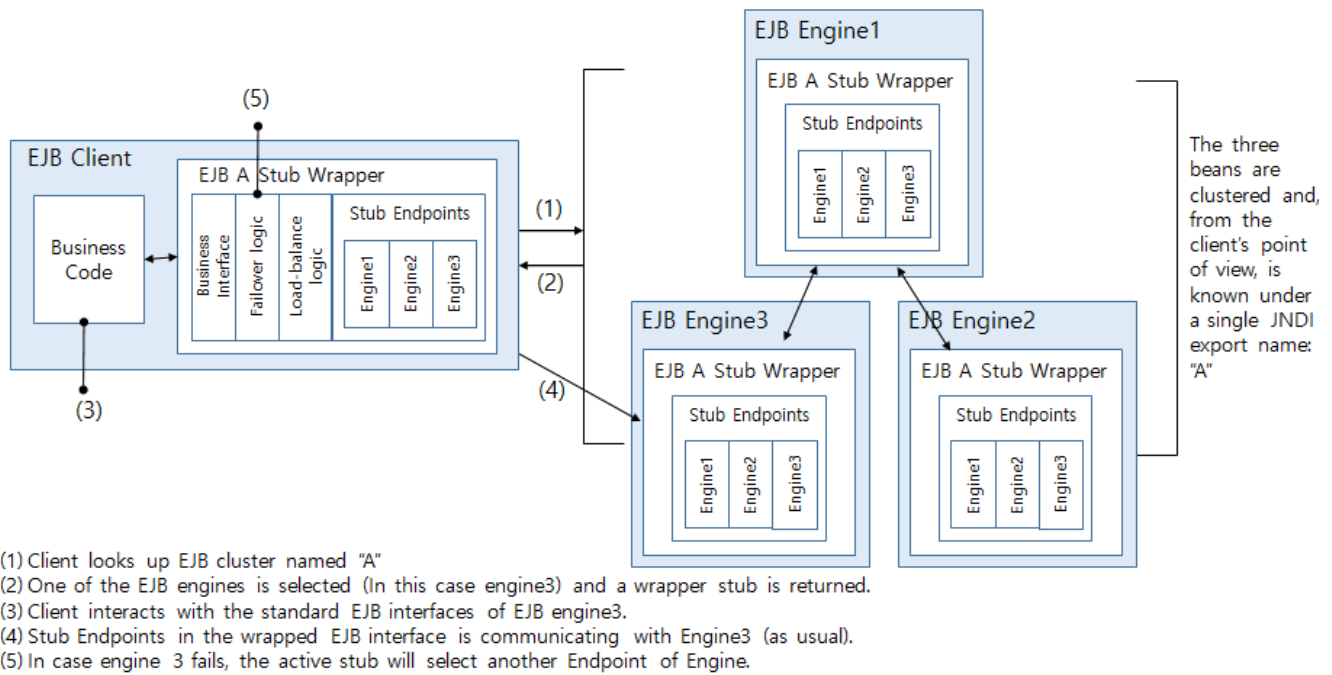
EJB 3 Stateless Clustering Architecture

If you deploy modules that need clustering, they are all bound to the same name in the Naming Service. Using this name, client can perform load-balancing and failover. Therefore, even if you deploy the same module by binding it to the Naming Server using a different name, the module will not be included in the cluster.

> Stateful session beans use the JEUS session manager for failover. There is only one session manager for each EJB engine, and therefore the clustering scope of a bean should not be dependent on the individual bean. For example, if bean A is bound to EJB engine1 and EJB engine2, and bean B is bound to EJB engine1 and engine3, the session manager wrongly assumes that the bean A and B are clustered on EJB enigne1, engine2, and engine3.

6.2. Key Features

The following explains major functions of EJB clustering.

6.2.1. Load Balancing

When a client requests for bean A through lookup or injection, a Naming Server randomly selects and returns one of the three beans. This implies that each of the three beans will receive an equal share of requests. Thus, the client can expect to increase the maximum system performance by three times than when having only one engine that services all the requests, not counting the small overhead associated for load balancing.

Based on the bean type, the client operates as follows:

- Stateless

  - EJB 3.x

    The client performs load balancing for each invoked method. Note that the first call goes to the EJB engine selected in a lookup.

    In JEUS 7 or earlier, clients could sent requests only to the EJB engine to which they sent the initial lookup request since only lookup load balancing was supported. However, starting from JEUS 8, they can send requests to engines other than the one to which the lookup request has been sent. Such load balancing of method calls is enabled because the engine contains endpoints such that client requests can be sent to other engines. Refer to EJB 3 Stateless Clustering Architecture.

    The following are the available load balancing methods:

Category	Description
RoundRobin	Forwards requests to server instances on a round-robin basis.
Random	Forwards requests to servers randomly.
LocalLinkPreference	Forwards requests to the server instance on which a lookup has been performed.

    A load balancing policy can be configured in the system property jeus.ejb.cluster.selection-policy. For more information, refer to "EJB System Properties" in JEUS Reference Guide.

  - EJB 2.x

    Clients select an EJB engine for a lookup using jeus.jndi.clusterlink.selection-policy, and all requests are sent to the engine.

- Stateful

  Clients select an EJB engine for a lookup using jeus.jndi.clusterlink.selection-policy, and all requests are sent to the engine.

The following table summarizes the load balancing mechanism for each EJB version and engine method.

Category	New InitialContext()	Lookup	Method call
EJB 2.x, EJB 3.x - Stateful	O (Random)	O	X
EJB 3.x - Stateless	O (Random)	X	O

To perform load balancing for stateless EJB 3.x engines in the same way as EJB 2.x engines even at the expense of compromising performance, refer to Configuring EJB Clustering and use the clustering method for JEUS 7 and earlier versions.

6.2.2. Failover (EJB Recovery)

Failover is the ability to recover from a failure in a single EJB service (e.g., OS crash, network disruption, or severe EJB engine failure).

There are two types of recoveries that the JEUS system can perform:

- **Re-issuing of a pending (incoming) request to a different bean than the one originally intended if that bean is found to be inaccessible when the client's request is received.**

    Simply perform load balancing algorithm by excluding the failed/unreachable EJB engine. Select a working, accessible bean or engine and process the new client request.

    In JEUS, this failover re-routing of a request is actually handled by the EJB stub from the client side. This stub is called an "active stub" and is simply a wrapper class for the standard EJB interfaces. The difference when using a wrapper class is that they contain logic that determines if the currently connected bean or EJB engine has failed. If a failure is detected, the active stub, on its own initiative, will request JNDI server for a new stub on behalf of the currently running EJB engine. (Refer to step 5 in either EJB Clustering Architecture or EJB 3 Stateless Clustering Architecture.)

- **Re-issuing a request in progress to another bean in the cluster if the current bean experiences some kind of runtime failure**

    Recovery method is very limited in this situation. If the failure is detected during a request, we have no way of telling how much the bean has processed the request and which runtime error occurred at the time of the failure. Simply re-invoking the same method again in a different bean in the cluster could potentially be dangerous, since that may lead to other side effects.

    To make this clear, consider a bean instance A. One of its business methods increments the value of a database field by one (e.g., i++). Let's assume that a failure occurs right after the increment is executed. In response to the failure, if you simply try to re-invoke the same business method through another bean B, the method will re-execute the increment. Thus, we would end with an inconsistent, erroneous value in the database. You can avoid such a critical error by declaring an idempotent method. For details about using an idempotent method for recovery, refer to EJB Recovery through Idempotent Methods.

The two scenarios differ by when an error condition is detected. The first method is used for a situation when a remote business method has not been invoked yet, and the second is used for a situation when the bean is already processing the request.

6.2.3. EJB Recovery through Idempotent Methods

An Idempotent method is a getter method that has no side effects. This means that we can invoke an idempotent method with the assurance that no states (e.g., instance variables, database fields, etc.) are changed during its execution.

An idempotent method can be used to deal with the second recovery issue of Failover (EJB Recovery): if a method is idempotent, we can safely re-invoke it on a different bean even if the first attempt to

invoke the method failed during execution. However, if a method is not idempotent, there is nothing we can do in such a scenario. It is better to throw an exception than re-executing the method when a runtime error occurs. Hence, EJB failover performs better when more Idempotent methods are used. The status of the business method must be identified before configuring it.

6.2.4. Session Replication

A stateful session bean uses JEUS session manager to back up sessions. Generally, session state changes according to each business method invocation. Thus the session manager requests for a back up of sessions to JEUS session manager whenever a method is called and the result is returned in JEUS. This backup process is also called **Session replication**.

JEUS session manger can copy sessions either synchronously or asynchronously, which is called a replication mode in JEUS.

- Sync replication mode

  Bean has to wait until session backup is completed. It is beneficial for failover because the most recent session is always copied. On the other hand, a bean has to wait for the session manager when the session manager does not proceed because of a network failure timeout.

- Async replication mode

  There are no performance issues in this mode, but since replication does not occur immediately, if failover occurs during that waiting time it may not get the latest session data.

Both methods have pros and cons, thus JEUS lets the user select the mode according to the bean and business method characteristics. The user also can specify not to use session replication. For more information, refer to Configuring EJB Clustering.

> In JEUS, by default, session replication uses the Sync mode for stateful session beans that are clustered. But this configuration can be change by the user.

6.3. Configuring EJB Clustering

EJB clustering can be configured either by using annotation in the bean class or the jeus-ejb-dd.xml file. Configuration items for clustering include the Bean to be configured, idempotent methods of the bean, and session replication mode of each bean or method.

This section shows how to configure annotation and descriptor (xml) for clustering through examples.

6.3.1. Configuring Clustering through Annotation

Configure classes or methods of a clustered bean by using annotation as follows.

```java
package ejb.basic.statelessSession;

import jakarta.ejb.Stateful;
import jeus.ejb.Clustered;
import jeus.ejb.Replication;
import jeus.ejb.ReplicationMode;

@Stateful(name="counter", mappedName="COUNTER")
@Clustered(useDlr = true)
@Replication(ReplicationMode.SYNC)
@CreateIdempotent
public class CounterEJB implements Counter, CounterLocal {
    private int count = 0;

    public int increaseAndGet() {
        return ++count;
    }

    @Replication(ReplicationMode.NONE)
    public void doNothing(int a, String b) {
    }

    @Idempotent
    public int getResult() {
        return count;
    }

    @Idempotent
    @jeus.ejb.Replication(ReplicationMode.ASYNC)
    public int getResultAnother() {
        return count;
    }
...
}
```

Descriptions for each class are as follows:

Class	Description
@jeus.ejb.Clustered	Enables or disables clustering of all beans. The configuration useDlr=true indicates that the high-speed clustering method introduced from JEUS 8 is disabled. This configuration is designed to provide compatibility that might be compromised by the new clustering method that entails a different failover method than before. For JEUS 8, this option is only available for EJB 3.x stateless session beans. (Default value: false)

Class	Description
@jeus.ejb.Idempotent	◦ Indicates whether a bean or business method is idempotent. ◦ If the value is false, it is not idempotent. ◦ When an exception occurs while creating a remote bean object, always perform retries using idempotent methods for stateless session beans, and throw an exception to the application using a non-idempotent operation for stateful session beans.
@jeus.ejb.CreateIdempotent	◦ Indicates whether to set methods as idempotent when creating session bean. ◦ If the value is false, methods are not idempotent. If this annotation is not described, stateless session bean methods are always assumed to be idempotent.
@jeus.ejb.Replication	◦ Indicates session replication mode for each bean or each business method. ◦ This class option takes as input an enum type class called **jeus.ejb.ReplicationMode**. For more information about this class, refer to JEUS API documents (javadoc). This annotation can be used in the form, @Replication(Replication- Mode.SYNC). Users can use the annotation for each bean, but if it is not configured, the default value is **ReplicationMode.SYNC**. In general, if a method is not configured, it follows the bean configuration. But for @Idempotent, ReplicationMode.NONE is the default value, and the create() method, a home business method, follows the configuration of the bean.

6.3.2. Configuring Clustering Using XML

The following configuration for each clustered bean can be configured inside the **<clustering>** tag in the JEUS EJB module DD file (jeus-ejb-dd.xml).

Configuring Clustering by using XML: <jeus-ejb-dd.xml>

```
<jeus-ejb-dd>
    . . .
    <beanlist>
        . . .
        <jeus-bean>
            <ejb-name>counter</ejb-name>
            <export-name>COUNTER</export-name>
            . . .
            <clustering>
                <enable-clustering>true</enable-clustering>
                <use-dlr>true</use-dlr>
                <ejb-remote-idempotent-method>
                    <method-name>getResult</method-name>
                </ejb-remote-idempotent-method>
                <ejb-remote-idempotent-method>
                    <method-name>getResultAnother</method-name>
                </ejb-remote-idempotent-method>
```

```
                <create-idempotent>true</create-idempotent>
                <replication>
                    <bean-mode>sync</bean-mode>
                    <methods>
                        <method>
                            <method-name>doNothing</method-name>
                            <method-params>
                                <method-param>int</method-param>
                                <method-param>java.lang.String</method-param>
                            </method-params>
                            <mode>none</mode>
                        </method>
                        <method>
                            <method-name>getResultAnother</method-name>
                            <method-params>
                                <method-param>void</method-param>
                            </method-params>
                            <mode>async</mode>
                        </method>
                    </methods>
                </replication>
            </clustering>
            . . .
        </jeus-bean>
        . . .
    </beanlist>
    . . .
</jeus-ejb-dd>
```

Descriptions for each tag are as follows:

Tag	Description
<enable-clustering>	Indicates whether to enable clustering of all beans.
<use-dlr>	Indicates whether to enable DynamicLinkRef when EJB clustering is enabled. ◦ true: The new clustering method starting from JEUS 8 is not enabled. This configuration is designed to provide compatibility that might be compromised by the new clustering method that entails a different failover method than before. For JEUS 8, this option is only available for EJB 3.x stateless session beans. ◦ false: The new clustering method starting from JEUS 8 is enabled. (Default value)
<ejb-remote-idempotent-method>	Defines idempotent methods among bean methods. (Refer to @jeus.ejb.Idempotent in Configuring Clustering through Annotation).
<ejb-remote-idempotent-exclude-method>	Defines methods that need to be excluded from methods that are defined as idempotent. This has precedence over the <ejb-remote-idempotent-method> setting, but is used in the same way.

Tag	Description
<ejb-home-idempotent-method>	Defines idempotent methods among methods defined in 2.x style home interface (Refer to @jeus.ejb.CreateIdempotent of Configuring Clustering through Annotation). This is used in the same way as <ejb-remote-idempotent-method>.
<ejb-home-idempotent-exclude-method>	Defines methods that need to be exclude from idempotent methods defined in 2.x style home interface. This has higher priority over <ejb-home-idempotent-method>, but is used the same way as <ejb-remote-idempotent-method>.
<create-idempotent>	Defines whether to declare a bean as idempotent when creating a session bean.
<replication>	Specifies the session replication mode at the bean or method level. For more information, refer to Session Replication and Configuring Clustering by using XML: <jeus-ejb-dd.xml>.

In order for the previously defined bean clustering to work, you must consider these issues.

- All beans participating in clustering should have the same settings inside the <clustering> element.
- In order to create a cluster, beans should have the same <export-name> value.

6.4. EJB Failover Restrictions

A client may use the EJB reference (Stub) by caching it or using injection, instead of doing a look up each time. In this case, failover may not be performed even though there is another alive end-point. This may occur when MSs, that were alive in the EJB end-point list at the time of the lookup, are not alive, and only new MSs that started after the lookup are alive.

EJB references are looked up internally at the time of the failover, and a new EJB reference is sent to the client. However, if all deployed MSs are terminated when you are doing a lookup of an EJB reference that is in use, failover cannot be performed even though there is a new EJB end-point that is deployed after the lookup and is able to provide the service.

In the previous case, the termination of existing MSs means that the nodes are abnormally terminated or shut down, or an EJB end-point is undeployed and cannot provide the service. The deployment of a new EJB end-point can mean that the time when the new MS was included to a cluster was too late (when EJB reference is already looked up and used), downed MS was restarted too late, or the undeployed EJB end-point was redeployed too late.

This may occur when two MSs are used for active/backup clustering. The following is an example scenario.

A single EJB engine exists in each MSs: A, B, and C. A client looks up EJB references for the first time and caches them.

1. EJBs are deployed to A, B, and C, and EJB on A is obtained as a result of a lookup.

2. EJB on A is abnormally terminated during use. EJB on B is obtained through lookup internally and it continues to provide the service.

3. EJB on B is undeployed, and EJB on C is obtained through lookup.

4. At this time, EJB on B is redeployed, and C is abnormally terminated.

In this case, when EJB reference of C is looked up, EJB end-point of B was undeployed and was redeployed after EJB on C was looked up. When C was terminated, B was still alive, but failover was not performed. However, if B was not deployed or undeployed, but was abnormally started or terminated, failover is performed. This is because the status of the engine for rebooting is checked for abnormal terminations.

It is possible to avoid the previous situation, if a new EJB Reference is looked up again and a new end-point list is obtained.

7. Session Beans

This chapter describes stateless session beans and stateful session beans in detail. The stateless session bean requires no additional configurations other than the ones that were already explained in Common Characteristics of EJB. Therefore, this chapter will mainly talk about stateful session beans.

7.1. Stateless Session Bean

This section describes about the Thread Ticket Pool (hereafter TTP) of session beans.

7.1.1. Thread Ticket Pool (TTP) and Object Management

Stateless session bean does not need to specify a connection pool because it does not have state information related to client requests and all clients share connections. However, because it is stateless and can reuse bean instances, it can use a bean pool. The following figure shows the relationship of TTP to a bean pool of a stateless session bean.
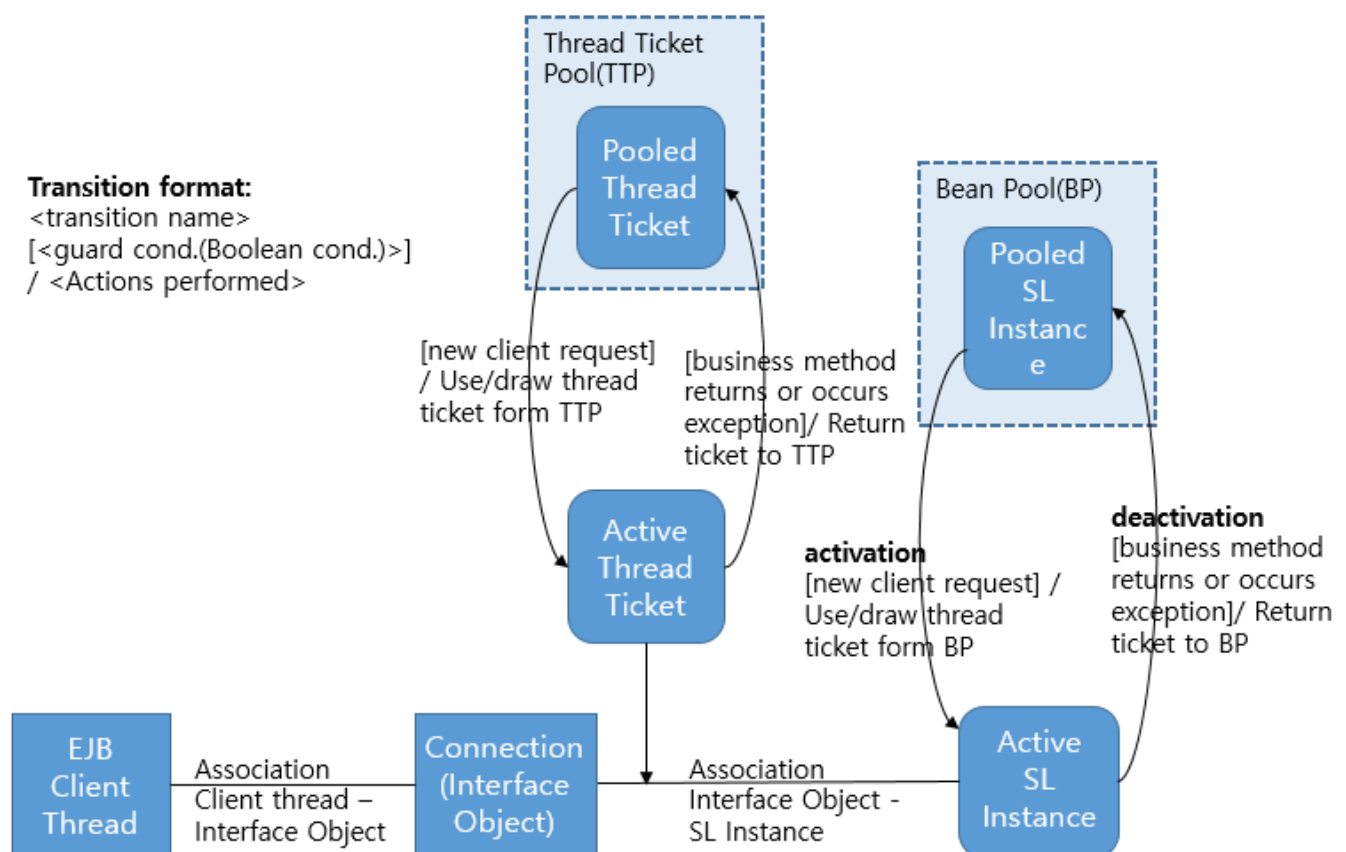
TTP and Bean Pool of Stateless Session Bean

1. If TTP receives a request from a connection to a client at the time of creation, it obtains a Thread Ticket. If TTP cannot obtain the TT, it will wait until it receives the TT as described in Common Characteristics of EJB. If the wait time exceeds 10 minutes, RemoteException will occur and the request cannot be performed.

2. TTP gets an actual bean instance from the bean pool and makes a connection to handle the request.

3. When a stateless session bean instance completes handling the request, the TT is returned to TTP and the bean instance is returned to the bean pool, and waits for the next request. For each request, a TT and a bean instance is allocated and returned.

The size of a bean pool differs from the size of TTP in that it is related to the number of local and remote client requests that can be simultaneously handled. For a remote call, TT is obtained from TTP and then an instance can be obtained from the bean pool. For a local call, since it is performed directly by a client thread, there is no need for a TT and only an instance is obtained from the bean pool.

Therefore, while the number of remote calls that can be handled cannot exceed the value of <thread-max> in jeus-ejb-dd.xml, any number of local calls can be handled because the maximum bean pool size is unlimited.

7.1.2. Web Service Endpoint

From EJB 2.1, a stateless session bean can be exported in the form of a web service.

> For more information about this, refer to "JEUS Web Service Guide".

7.2. Stateful Session Beans

This section describes additional configurations for stateful session beans.

- Object management configuration: the bean instance pool and the connection pool
- Bean instance pooling option
- State persistence mechanisms through session manager

7.2.1. Thread Ticket Pool (TTP) and Object Management

Since a stateful session bean needs to persistently store state information between client requests, we need to configure and utilize a connection pool (hereafter CP). The same also applies to entity beans. For more information, refer to Entity Bean.

The following figure shows the relationship among a bean pool, connection pool, and TTP of a stateful session bean.

Transition format:
<transition name>
[<guard cond.(Boolean cond.)>]
/ <Actions performed>
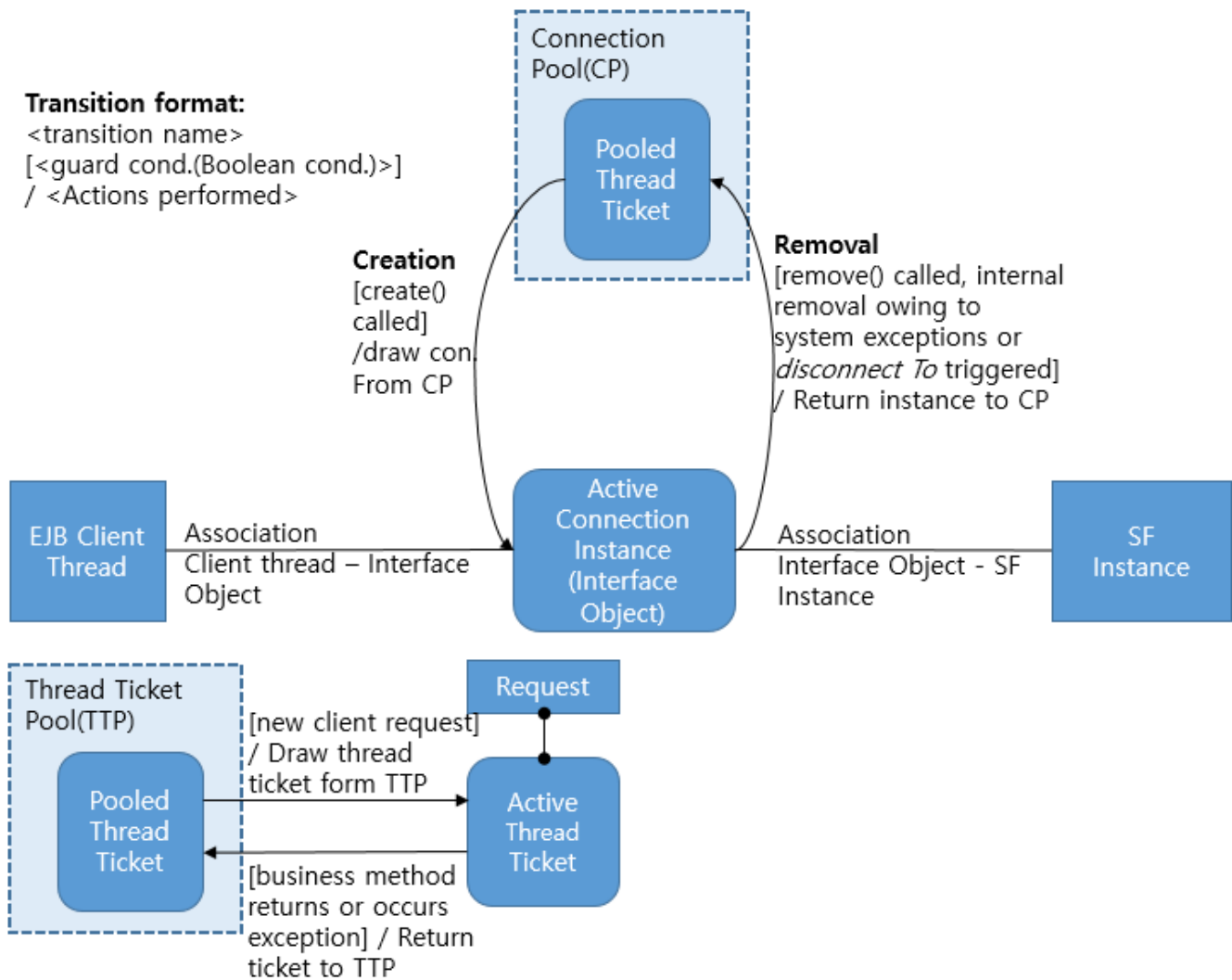
Connection Pool, Thread Ticket Pool, and Bean Pool of a Stateful Session Bean

If a client creates a stateful session bean, a new bean instance (SF instance) is created and a connection is made between the two through the Connection Pool. If the connection to the bean instance is given to a client, the connection is allocated only to the current client and is not shared among other clients. Therefore, unless the client discards it, it will not be returned to the Connection Pool. When it is returned to the Connection Pool, the connection also gets disconnected from the bean instance.

If a request is sent through the relevant connection, a TT is sent from the TTP. Only the requests that have received a TT are handled. Unlike stateless session beans, a bean instance is not allocated from a bean pool for each request, but a bean instance is assigned for a specific client.

As already mentioned, the bean instance is deleted when it is disconnected because the connection is returned to the Connection Pool when a client discards a bean. Since a stateful session bean contains state information, a bean pool that reuses a bean instance is not used.

7.2.2. Pooling Session Bean

According to the J2EE EJB specification, the actual instance of a stateful session bean may not be reused by different client sessions. It appears that this requirement, in most cases, is not really useful. JEUS therefore offers a way to go around the specification in this respect to achieve higher

performance and reduce resource waste.

If a bean pool is used, the bean pool instance is returned to the bean pool when the bean is deleted. A bean is deleted when a client calls remove() or a timeout occurs. At that time, PRE DESTROY callback is invoked, but this is only recommended when the initialization of the bean has been well implemented.

7.2.3. Configuring the Bean Pool

To turn a stateful session bean into a bean that is pooled, you simply set the <pooling-bean> element in jeusejb- dd.xml to "true" as follows:

Configuring the Bean Pool: <jeus-ejb-dd.xml>

```
<jeus-ejb-dd>
    . . .
    <beanlist>
        <jeus-bean>
            <ejb-name>teller</ejb-name>
            <export-name>TELLEREJB</export-name>
            <local-export-name>LOCALTELLEREJB</local-export-name>
            <export-port>7654</export-port>
            <export-iiop>true</export-iiop>
            <object-management>
                <pooling-bean>true</pooling-bean>
                <bean-pool>
                    <pool-min>10</pool-min>
                    <pool-max>200</pool-max>
                    <resizing-period>1800000</resizing-period>
                </bean-pool>
                . . .
            </object-management>
            . . .
        </jeus-bean>
        . . .
    </beanlist>
    . . .
</jeus-ejb-dd>
```

7.2.4. Configuring the Session Data Persistence Mechanism

As the name clearly implies, a stateful session bean contains state information. This information must be kept during a client session (i.e., the state must be preserved over a number of separate client requests coming from the same client). This is the basic property of stateful session beans.

Normally, during runtime, the state is naturally stored as a set of instance variables. However, after the stateful session bean enters the inactivated state as previously described, we need to remove these instance variables from the runtime environment in order to preserve system resources. This is why we need a secondary storage to where we can safely write the variables (state) to be retrieved later into memory when the bean is re-activated.

The distributed session manager included in JEUS is used as the secondary storage. Normally, when the bean is passivated, clustered stateful session bean information is sent to the session manager for failover when the bean successfully commits the transaction. The session manager is configured in MASTER.

7.3. Commonly Used Configuration

This section describes commonly used configuration for stateless and stateful session beans. The configuration is specified in the **<jeus-bean>** element of the JEUS EJB module DD.

7.3.1. Configuring Object Management

Since stateless session beans do not have state information and a single connection can be used, they can just use the bean pool. However, since stateful session beans have state information, they can use both a connection pool and a bean pool. Therefore, the following settings can be configured for a pool to reduce the load of creating a new instance each time.

The following is an example of XML section related to object management configuration.

Object Management Configuration: <jeus-ejb-dd.xml>

```
<jeus-ejb-dd>
    . . .
    <beanlist>
        <jeus-bean>
            . . .
            <object-management>
                <bean-pool>
                    <pool-min>10</pool-min>
                    <pool-max>200</pool-max>
                </bean-pool>
                <connect-pool>
                    <pool-min>10</pool-min>
                    <pool-max>200</pool-max>
                </connect-pool>
                <passivation-timeout>10000</passivation-timeout>
                <disconnect-timeout>1800000</disconnect-timeout>
            </object-management>
        <jeus-bean>
        . . .
    <beanlist>
    . . .
<jeus-ejb-dd>
```

Details for sub elements of the <object-management> element.

- **<bean-pool>**
    - Determines the behavior of the bean instance pool.
    - The following describes sub-tags.

Tag	Description
<pool-min>	Initial number of bean instances created when a pool is initialized (Default value: 0)
<pool-max>	Maximum number of bean instances that can be stored a pool after use. (Default value: 100)
<resizing-period>	Time interval for resizing a pool. Resizing means that unused bean instances are removed, and only the minimum number of bean instances are left in the pool. (Default value: 5 minutes, Unit: ms)

- **<connect-pool>**

    ◦ Determines the number of connections, that connect clients and bean instances, to keep (For JEUS 6 Fix#8 and later, this option is not supported for session beans).

    ◦ The following describes sub-tags.

Tag	Description
<pool-min>	Initial number of connections created when a pool is initialized. (Default value: 0)
<pool-max>	Maximum number of connections for a pool after the connection is used. (Default value: 100)
<resizing-period>	Time interval for resizing a pool. Resizing means that unused connections are removed, and only the minimum number of connections are left in the pool. (Default value: 5 minutes, Unit: ms)

- **<passivation-timeout>**

    ◦ Used for deactivating a bean in the EJB engine for which there are no client request for a specified period of time. If a bean receives no client requests during the specified time period set in this element, the bean is subject to passivation.

    The time interval for checking passivation targets depends on the <resolution> value in EJBMain.xml. The relevant bean instance is deleted from memory, the state of the instance is saved to a file using a distributed session server internally.

    ◦ These settings can be configured at several locations and their priorities are as follows:

    1. Applied to only certain session beans: <passivation-timeout> of jeus-ejb-dd.xml.

    2. Applied to all stateful session beans: the system property jeus.ejb.stateful.passivate

    3. Applied to all EJB beans (all entity beans and session beans): the system property jeus.ejb.all.passivate

    4. Applied to all EJB beans (all entity beans and session beans): <node>/<session-router-config>/<session- router>/<file-db>/<passivation-to> of MASTER

    If no setting exists, default value is 300,000ms (5 minutes).

- **<disconnect-timeout>**

- Used for disconnecting a connection between the client and a stateful session bean instance if no client request is received during the amount of time specified here. This means that the connections are disconnected from each client and instance, and returned to a connection pool.

  Therefore, a client can no longer make a request through this connection. Bean instances in use are removed, or returned to a bean pool if the bean pool is used.

- This setting can be configured from several locations as shown in the following example, and their priorities are as follows:

  1. Applied to only a certain session bean (Priority in the following order)

     a. <stateful-timeout> of ejb-jar.xml

     b. @StatefulTimeout

     c. **(Deprecated)** <disconnect-timeout> of jeus-ejb-dd.xml

  2. Applied to all stateful session beans: the system property jeus.ejb.stateful.disconnect

  3. Applied to all EJB beans (all entity beans and session beans): system property jeus.ejb.all.disconnect.

     If all of these settings do not exist, the value is set to 3,600,000 ms (1 hour), the default value of system property jeus.ejb.all.disconnect.

     (Default value: 3,600,000 (1 hour), unit: ms)

> The time period used for <passivation-timeout> and <disconnect-timeout> are both measured from the time of the last access of the bean instance. This means that you should set the <disconnect-timeout> period considerably larger than <passivation-timeout>. And <passivation-timeout> should be larger than the resolution value specified in the EJB engine.
>
> If timeout value is too large (more than 10 minutes or when timeout is stopped), a large number of inactivated instances remain in the memory for the long duration, wasting system resources. Too short of a timeout period (several seconds), however, may undermine the performance due to frequent inactivations and activations, and has a risk of losing the session values.

8. Entity Bean

From EJB 3.0, entity bean is replaced by JPA. Therefore, it is recommended to use JPA by referencing the JEUS JPA guide when creating a new bean. However, JEUS EJB engine supports entity beans for existing users.

This chapter describes everythijng about configuring and tuning entity beans in JEUS EJB engine.

8.1. Overview

In compliance with the EJB specifications, the following three entity bean types are fully supported in the JEUS EJB engine.

- Bean Managed Persistence Entity Bean (BMP)
- Container Managed Persistence Entity Beans according to the EJB 1.1 specification (CMP 1.1)
- Container Managed Persistence Entity Beans according to the EJB 2.0 specification (CMP 2.0)

Regardless of types, entity beans in JEUS share configuration functions and components described in Common Characteristics of EJB. Therefore, in order to configure and use entity beans in JEUS EJB engines, understanding of the relevant chapters is required.

There are three kinds of configuration settings related to these three bean-types:

- Shared configurations that apply to all three types of entity beans
- Configurations applied to **CMP 1.1** and **CMP 2.0** only
- Configurations applied to **CMP 2.0** only

The following table shows the configurable properties by entity bean type.

Configurable Entity Property	BMP	CMP 1.1	CMP 2.0
1. EJB name	√	√	√
2. Export name	√	√	√
3. Local export name	√	√	√
4. Export port	√	√	√
5. Export IIOP switch	√	√	√
6. use-access-control	√	√	√
7. Run-as identity	√	√	√
8. Security CSI Interop.	√	√	√
9. Env. refs	√	√	√
10. EJB refs	√	√	√
11. Resource Refs	√	√	√

Configurable Entity Property	BMP	CMP 1.1	CMP 2.0
12. Resource env. Refs	√	√	√
13. Thread ticket pool settings	√	√	√
14. Clustering settings	√	√	√
15. HTTP invoke	√	√	√
16. Object management	√	√	√
17. Persistence Optimize	√	√	√
18. CM persistence opt.		√	√
19. Schema info		√	√
20. Relationship map			√

As we can see from the previous table, items 1 through 15 apply to all entity types. These items were previously described in Common Characteristics of EJB.

Item 16 is for both session beans and entity beans, but in a different way. The difference will be explained in this chapter. Items 17 through 20 apply only to entity beans, and will be fully described in this chapter.

Apart from these, we shall also discuss the following topics, pertaining only to JEUS Entity EJB:

- Default primary key class
- JEUS-specific items added to the EJB QL language
- JEUS Instant EJB QL API

> Performance tuning is a major part of EJB entity bean configuration in JEUS. Therefore, this chapter will discuss many performance-related issues such as engine modes and sub-modes. The entity bean tuning section (Tuning Entity EJBs) later in this chapter is a concise summary of what will be covered in this section.
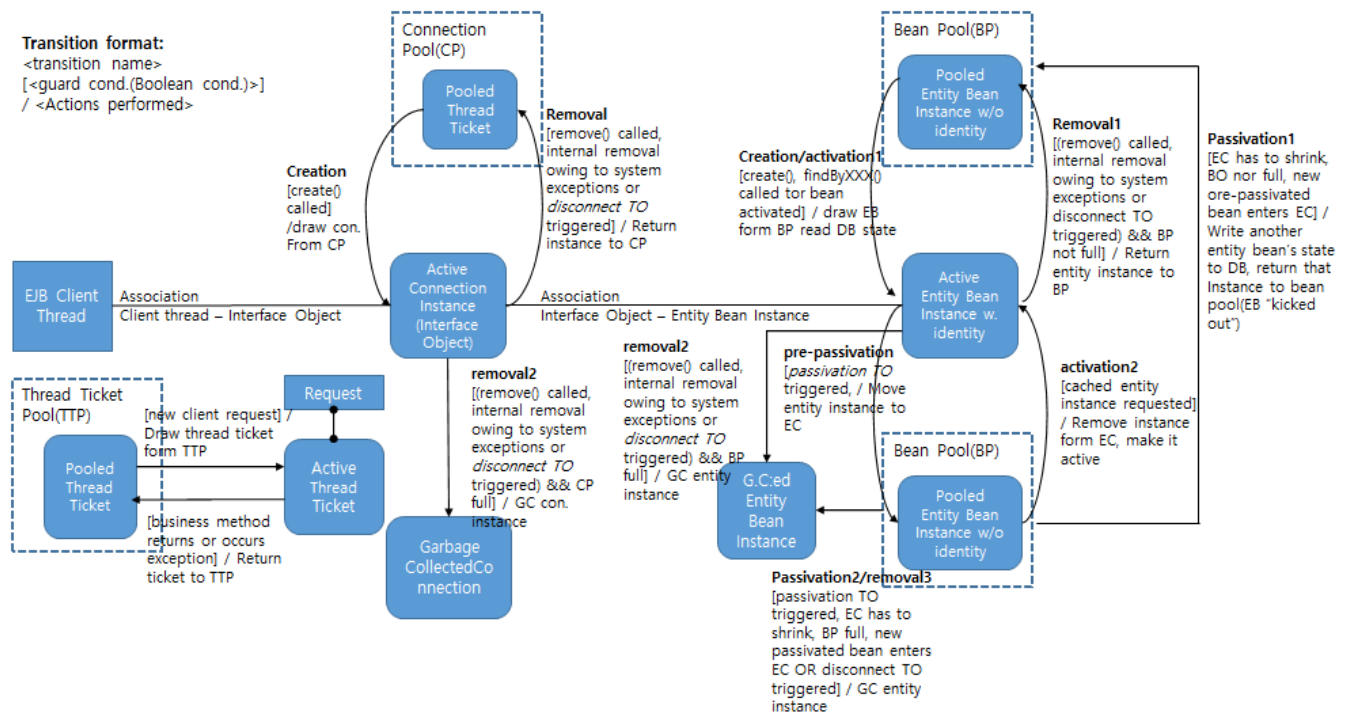
8.2. Key Features

This section describes common functions of entity beans and the main functions of each bean.

8.2.1. Common features

The JEUS entity bean functions are shared by all three types of entity beans in JEUS.

8.2.1.1. Entity Bean Object Management and Entity Cache

The following figure shows how entity bean objects are managed in the EJB engine.

Entity Bean Object and Instance Management in the JEUS EJB Engine

The term "object management" refers to the internal bean instance pool and connection pool of the EJB engine. These pools are configured for each bean, and helps improve the performance and resource management of the system. Basically, it acts like a stateful session bean. This section will only discuss the differences. For other details, refer to Session Bean.

An entity bean also uses the Entity-Cache along with the Connection Pool. When an entity bean is about to be passivated (when the passivation timeout expires), the bean instance is not actually passivated(Stateful Session Bean gets passivated). Instead, it is sent to the entity cache. All the entity beans that have been passivated are stored in the Cache in the activated state (they maintain an identifiable and valid status).

Entity beans are actually passivated in the following case.

An entity cache is passivated when it needs to reduce its size. If a new entity bean is about to enter the cache, but the cache is found to already be filled with "inactivated" beans, an entity bean that is already in the cache will be selected and removed from the cache. The removed entity bean will now be inactivated, and its state is written to the back-end database and its identity-less instance is now either returned to the bean pool or discarded (if the bean pool is full). The bean that was waiting to enter the cache replaces the removed bean instance in the cache.

The size of the cache may be adjusted with the <persistence-optimize> / <entity-cache-size> option. It does not reduce the cache size, but the setting is used as a hint. The removed bean instances are not selected in the order of how long they have remained in the cache.

The advantage of using such a cache is that we can improve the time-consuming inactivation management by using the cache. We do not always need to go back to the EJB database when we re-activate an inactivated entity bean; we can simply use the instance found in the cache (if present).

While the standard inactivation timeout value for object management allows the deployer to set time constraints for inactivation, the entity cache allows him or her to also set up memory-constraints for inactivation, which is the main purpose for using an entity cache.

> The entity cache configuration is not actually a part of the object management configuration but rather the general entity bean persistence optimization settings (see next sub-section). This will be clearly shown in Configuring Entity EJBs.

8.2.1.2. ejbLoad() Persistence Optimization through Engine Mode Selection

It is up to the EJB engine (container) to decide when to invoke the ejbLoad() method on an entity bean. This method is used both in BMP and CMP beans to synchronize the runtime state of the bean instance with the state of the DB. Essentially, the ejbLoad() method is used to read in a row from the database and to populate the internal instance fields of the bean with the current row values.

The ejbLoad() method will be invoked at least once during the life-cycle of an entity bean: when it is first instantiated. In some cases, however, it will also be necessary to invoke the ejbLoad() method before every invocation of the bean instance.

There are basically two such scenarios when the state of the bean must be repeatedly read from the DB:

- When a bean is clustered across several EJB engines, any bean in the cluster may modify a row in the DB. This means that we can never be sure whether the state stored in a particular entity bean instance is the most recent one.
- When some external entity (user or system) modifies the EJB connected to the DB.

  In this scenario, the state of a particular EJB instance on the EJB engine may not reflect the correct information.

The following figure shows that the ejbLoad() method is called periodically when entity beans are clustered or when an external entity modifies the bean's DB row.
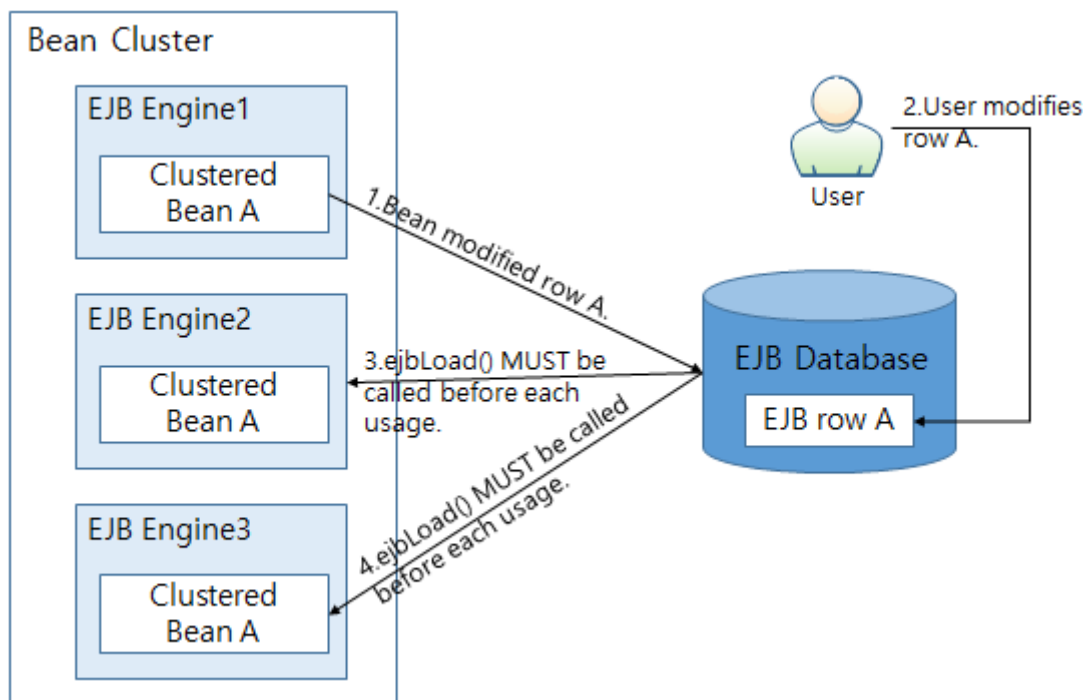
Scenarios where ejbLoad() is Called Periodically

If the previous two scenarios can be avoided (under the condition that you do not use clustering and you can guarantee that no external entity modifies the EJB's back-end data), you may optimize the behavior of the ejbLoad() method for the entity bean.

This is done by specifying what is known as the EXCLUSIVE_ACCESS engine mode for the bean. In this mode, the ejbLoad() method will only be called once, when the bean is instantiated. This will effectively remove 50% of database accesses.

The following figure depicts a scenario when the EXCLUJSIVE_ACCESS mode can be used. With EXCLUSIVE_ACCESS, only one bean accesses the database row, and most of the ejbLoad() method calls are optimized.
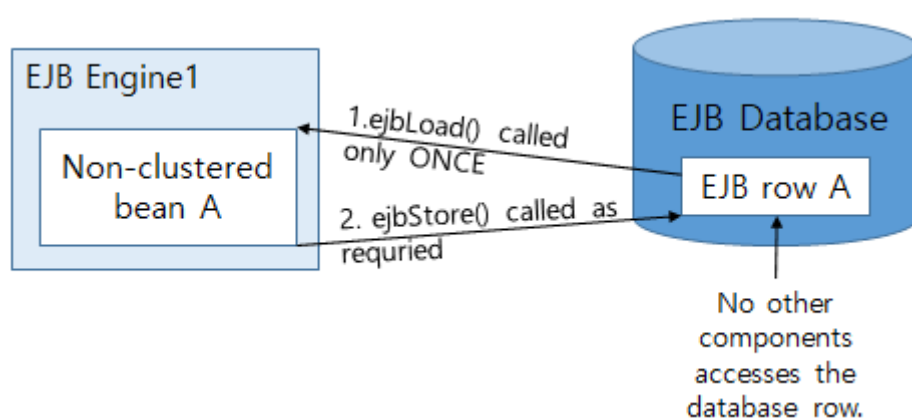
EXCLUSIVE_ACCESS Mode

If you do need to cluster your beans or if some external entity needs to modify the EJB data, you must use either the SINGLE_OBJECT or MULTIPLE_OBJECT engine modes. When either of these modes is used, the EJB engine will invoke the ejbLoad() method each time an EJB client request is made. This will degrade the performance of individual entity bean deployments. However, this will allow you to distribute the load across many different beans and engines, since the state will now be

kept and managed entirely by the DB.

The difference between the SINGLE_OBJECT and MULTIPLE_OBJECT modes is as follows:

- SINGLE_OBJECT engine mode

  With SINGLE_OBJECT, only one bean instance in each EJB engine will handle all client requests, implying that other requests that arrive during the processing of the initial request will have to wait
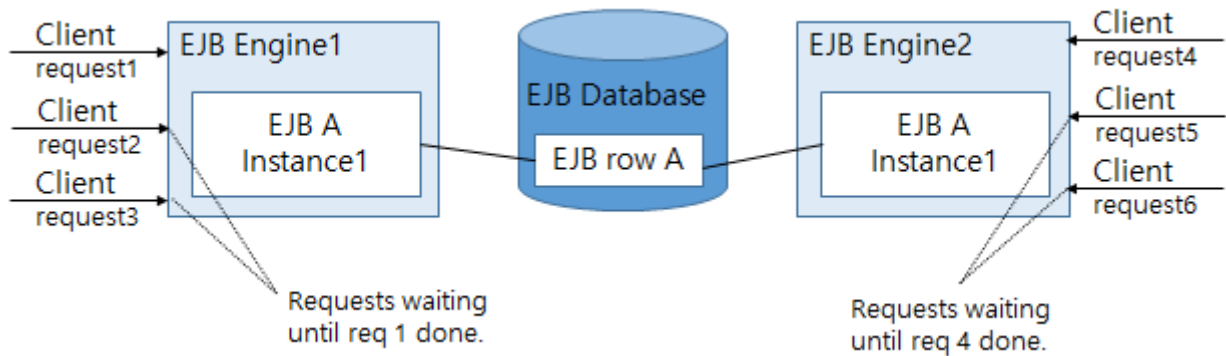
SINGLE_OBJECT Engine Mode

- MULTIPLE_OBJECT engine mode

  With MULTIPLE_OBJECT, however, the previous restriction does not apply. The EJB engine will spawn a new entity bean instance for handling each new EJB client request. In other words, a new EJB instance gets allocated in this mode.
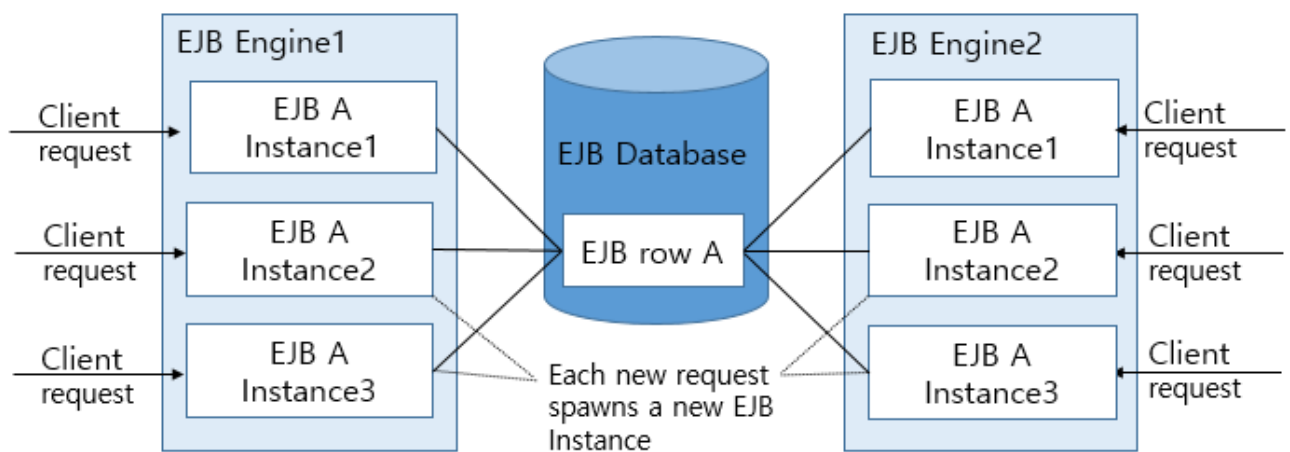
MULTIPLE_OBJECT Engine Mode

The following table lists all the advantages and disadvantages of selecting an engine mode.

Criterion	EXCLUSIVE	SINGLE	MULTIPLE
Can be used with clustering?	No	Yes	Yes
Can be used if external entity accesses the DB?	No	Yes	Yes
Efficiently uses database connections (ejbLoad() called rarely)?	Yes	No	No
Provides good options for time-consuming transactions?	No	No	Yes

Criterion	EXCLUSIVE	SINGLE	MULTIPLE
Efficient mode for one engine to handle a single request?	Not applicable	Yes	No
Efficient for one engine to handle multiple, concurrent EJB requests?	Not applicable	No	Yes

The following is the guidelines for choosing a mode.

- If you expect a large number of requests for your entity bean, and if you also have a large number of EJB engines set up, use the SINGLE_OBJECT mode and cluster the beans across the engines.

- If you expect a large number of requests for your entity bean, but if you only have a limited number of EJB engines set up, use the MULTIPLE_OBJECT mode and cluster the beans across the engines.

- Another very important consideration when choosing a mode is the total time needed for a single transaction to complete. If this time is long, it is recommended to use the MULTIPLE_OBJECT mode.

- If you expect a very moderate number of concurrent requests for your entity bean, use the EXCLUSIVE_ ACCESS mode and deploy the bean to only one EJB engine.

For information about EJB clustering, refer to EJB Clustering.

8.2.2. BMP & CMP 1.1

This subsection will show ways to optimize the ejbStore() method invocations.

> The discussion in this subsection applies only to BMP and CMP 1.1 beans. CMP 2.0 uses a different approach to solve these problems.

8.2.2.1. ejbStore() Persistence Optimization through Non-modifying Methods

The ejbStore() method is called by the EJB engine whenever it is decided that the bean's state must be persisted to the back-end DB. Usually, saving to the back-end storage is performed after each transaction commitment or before the bean instance is inactivated.

If, however, during the course of a transaction, a bean's state does not change, there is no reason to call the ejbStore() method.

Currently, the JEUS EJB engine cannot automatically detect whether a bean's state has changed or not. Therefore, you as the developer or deployer must provide hints to aid the EJB engine decide whether to invoke ejbStore() or not. These hints are provided as a list of so-called non-modifying methods. A non-modifying method is essentially a read-only method (a getter method) that does not change the entity bean's internal database-bound state in any way.

By looking at the list of non-modifying methods, the EJB engine can decide whether to invoke the ejb Store() method. If only non-modifying methods were executed during a transaction or when a bean is an activated state, the EJB engine skips the method call by assuming that it is not needed. This implies the improvement of the overall performance.

8.2.3. CMP 1.1/2.0

The following describes major features of CMP 1.1/2.0.

8.2.3.1. ejbLoad(), ejbFind(), CM Persistence Optimization

ejbLoad() Persistence Optimization through Engine Mode Selection and ejbStore() Persistence Optimization through Non-modifying Methods talked about when the ejbLoad() and ejbStore() methods should be invoked. That discussion applied to both BMP and CMP type entity beans since both types use these methods.

In this sub-section, we will discuss how the engine-generated implementation of the ejbLoad() and ejbFind() methods can be optimized declaratively to enhance the overall performance.

This discussion only applies to CMP type beans (both the 1.1 and the 2.0 versions) since with this bean type, it is up to the EJB engine to actually define how to implement the ejbLoad() and ejbFind() methods. With BMP, on the other hand, the implementation and behavior of the methods are completely up to the developer.

When the EJB engine makes a call to the database to execute the ejbLoad() and ejbFind() methods of CMP, it can optimize its behavior by choosing among three possible engine sub-modes:

Mode	Description
ReadLocking	When ejbLoad() is invoked in this mode, the EJB engine has a so-called shared lock on the database. This mode allows other beans to read from the same row in the database, but prevents them from writing to the row.
WriteLocking	When ejbLoad() is invoked in this mode, the EJB engine has a so-called exclusive lock on the database. Both read and write operations from other beans and threads are prohibited.
WriteLockingFind	When ejbLoad() and ejbFind() are invoked in this mode,the EJB engine has exclusive lock on the database. Other beans are prevented from reading and writing to the database.

We only describe which mode should be used for a given situation. The technical details of the modes will not be covered here.

- If your CMP bean is expected to do more reading than writing operations, always use the ReadLocking mode.
- If your CMP bean is expected to do more writing than reading operations, select the WriteLocking or WriteLockingFind mode.

For CMP, the engine mode, the use of non-modifying methods described in the previous sub-sections, and the engine sub-mode described in this sub-section should be combined according to the set rules. See Tuning Entity EJBs for a summary on configuring these parameters.

For CMP beans, you may also specify java.sql.ResultSet fetch size and a setting called "init caching" that determines whether EJB bean instances should be pre-instantiated and cached when the EJB engine boots.

8.2.3.2. Entity Bean Schema Information

When the EJB engine is responsible for managing bean persistence (CMP), it also provides a way to define the mapping between entity bean instance fields and the actual tables and columns in the database. It also requires a data source to be defined for use. This information is encapsulated in the <schema info> element of the EJB XML configuration.

For CMP 1.1, this element also contains SQL fragments used as a base when the EJB engine generates CMP 1.1 finder methods.

8.2.3.3. Automatic Primary Key Generation

In some cases, a developer may need to create an EJB instance without specifying a primary key. In these cases, the JEUS EJB engine can be configured to take responsibility for generating a unique primary key for the new EJB instance. The JEUS EJB engine provides this functionality through cooperation with the Database.

In JEUS, automatic primary key generation works for both CMP 1.1 and CMP 2.0 beans. With this feature, EJB may be easily developed regardless of the presence of a primary key.

The following describes how to use and apply the automatic primary key generation function.

1. The developer tends to call the create() method for the CMP1.1/CMP2.0 EJB without parameters. This means that the create() method in question does not receive a parameter that corresponds to the primary key of the EJB.

   ```
   InitialContext ctx = new InitialContext();
   BookHome bHome = (BJookHome) ctx.lookup("bookApp");
   Book b = bHome.create("JEUS EJB Guide", "Park Sungho");
   // Start using "b".
   . . .
   ```

   The above example looks up the BookHome EJB and calls the create() method with two arguments, whose values are "JEUS EJB Guide" and "Park Sungho". The former argument represents the title of the new book, and the latter is the author's name. None of these arguments is appropriate for use as a primary key, as there may be several books with the same title, and several authors may have the same name.

   To support this kind of client code, the EJB engine (let us call it A) must now somehow generate a

unique primary key for the new Book instance, and it must do so under the assumption that several other EJB engines in an EJB cluster (B and C) may also try to achieve the same thing concurrently.

This means that somewhere, there must exist a single central storage, since each engine could try to fetch the cluster-wide unique primary key value. This storage is implemented in the form of a database table with a single row and a single column.

The single value in this row is the primary key counter (or primary key generator), which is constantly incremented after it is fetched by an EJB engine. In this way, the database always holds a new, unique primary key value that may be read and used by any EJB engine in the cluster.

2. After calling the create() method without providing a primary key, the EJB engine A accesses the central primary key storage (the primary key database). From this storage, it fetches the primary key value (which must always be an integer) and assigns it as the primary key of the new Book instance.

3. EJB engine A then increments the primary key field in the storage with a pre-determined value called the "key cache size". The pre-determined default value is 1, but it is recommended to use a value considerably larger than 1, e.g., 20.

   If the key cache size is set to 20, the current EJB engine A will thus allocate (or cache) nineteen (20-1) unique values for its own use. Other EJB engines (B and C), that read the primary key value next, will read a value that is twenty more than the previously fetched primary key value.

   This means that the engine does not need to fetch a new primary key value from the external primary key storage to create the following nineteen EJB instances inside the EJB engine without a primary key. This considerably helps improve the performance. It is simply sufficient for engine A to keep an internal counter to see when it has run out of allocated primary keys, forcing it to read a new value from the primary key storage. The internal counter is of course also used to directly generate a new unique primary key without involving the external storage.

The following describes how to use and apply the automatic primary key generation function.

- Oracle and Microsoft SQL Server databases support automatically generated primary key sequences. For more information about automatic primary key generation for these databases in the sub-sections, refer to Automatic Primary Key Generation Support with Oracle DB and Automatic Primary Key Generation Support with Microsoft SQL Server DB.

- Non-Oracle and non-Microsoft SQL Server databases need to be explicitly configured with a primary key table to hold the primary key value. This is discussed in Automatic Primary Key Generation Support with Oracle DB.

- The database that is to support this feature must support the TRANSACTION_SERIALIZABLE isolation level (configured for the JDBC calls). This guide assumes that it is supported.

- A primary key field must still be defined for the EJB. The primary key field must be of the type java.lang.Integer, it must not be a single key, and it must be declared in the EJB deployment DD.

> The automatic primary key generation functionality must support the isolation level, TRANSACTION_SERIALIZABLE. You must make sure that all

The following describes how to set up the DB properly.

Automatic Primary Key Generation Support for Oracle DB

Automatically generated primary keys in the Oracle DB use an Oracle-specific feature called sequence. To use primary key generation, you must setup a sequence generator for Oracle DB. Refer to the Oracle documentation for information about setting up a sequence generator.

The following is an example of the jeus-ejb-dd.xml file that generates a primary key in Oracle DBs.

Automatic Primary Key Generation Support for Oracle DB: <jeus-ejb-dd.xml>

```
<jeus-ejb-dd>
    . . .
    <beanlist>
        . . .
        <jeus-bean>
            . . .
            <schema-info>
                . . .
                <data-source-name>MYORACLEDB</data-source-name>
                <auto-key-generator>
                    <generator-type>
                        Oracle
                    </generator-type>
                    <generator-name>
                        my_generator
                    </generator-name>
                    <key-cache-size>
                        20
                    </key-cache-size>
                </auto-key-generator>
            </schema-info>
            . . .
        </jeus-bean>
        . . .
    </beanlist>
    . . .
</jeus-ejb-dd>
```

In the previous example, we have specified the primary key generator type as "Oracle", <generator-name> is set to "my_generator" and <key-cache-size> is set to "20".

The <key-cache-size> value must be set equal to the SEQUENCE INCREMENT value that was used to create the Oracle sequence generator.

The <data-source-name> value is used to select a particular Oracle DB. (In the previous example, "MYORACLEDB," a DB connection pool JNDI name defined in

Automatic Primary Key Generation Support for Microsoft SQL Server DB

Setting up automatic primary key generation for Microsoft SQL Server is rather simple. Microsoft SQL Server automatically maintains a unique primary key value in the IDENTITY column.

The following is an example of jeus-ejb-dd.xml that automatically generates a primary key in the Microsoft SQL Server DB.

Automatic Primary Key Generation Support for Microsoft SQL Server DB: <jeus-ejb-dd.xml>

```
<jeus-ejb-dd>
    <beanlist>
        . . .
        <jeus-bean>
            . . .
            <schema-info>
                . . .
                <data-source-name>MSSQLDB</data-source-name>
                <auto-key-generator>
                    <generator-type>
                        MSSQL
                    </generator-type>
                </auto-key-generator>
            </schema-inboldfo>
        </jeus-bean>
        . . .
    </beanlist>
</jeus-ejb-dd>
```

We thus see that it is sufficient to set the <generator-type> value to "MSSQL". The specific Microsoft SQL Server database is specified in the <data-source-name> element.

Automatic Primary Key Generation Support for Other DB Types

If you want to use automatic primary key generation with databases other than Oracle and Microsoft SQL Server, you must do the following:

1. Start by defining a new table in the Database. The table should have one row and at least one column with an arbitrary name. The value should initially be set to "0".

   The following table manages the primary key values.

	PrimKeyGeneratorColumn (The column to store the Primary Key value.)
Row 1	0 (Initial value of the Primary Key)

   The previous sample table could be created with two SQL statements as follows:

```
CREATE table PrimKeyTable (PrimKeyGeneratorColumn int);
INSERT into PrimKeyTable VALUES (0);
```

2. After completing step 1, modify the jeus-ejb-dd.xml file as follows:

Automatic Primary Key Generation Support for Other DB Types: <jeus-ejb-dd.xml>

```
<jeus-ejb-dd>
    . . .
    <beanlist>
        . . .
        <jeus-bean>
            . . .
            <schema-info>
                . . .
                <data-source-name>MYDB</data-source-name>
                <auto-key-generator>
                    <generator-type>
                        USER_KEY_table
                    </generator-type>
                    <generator-name>
                        PrimKeyTable
                    </generator-name>
                    <sequence-column>
                        PrimKeyGeneratorColumn
                    </sequence-column>
                    <key-cache-size>
                        20
                    </key-cache-size>
                </auto-key-generator>
            </schema-info>
            . . .
        </jeus-bean>
        . . .
    </beanlist>
    . . .
</jeus-ejb-dd>
```

In the previous example, we see how the type is set to 'USER_KEY_table', the key generator table name is set to 'PrimKeyTable', and the column used to store the primary key value is set to 'PrimKeyGeneratorColumn'. The key cache size is then set to 20.

The Database to use is specified in the <data-source-name> element. The value should match a DB connection pool <export-name> specified in the JEUSMain.xml file.

> For more information about settings, refer to "JEUS Server Guide".

8.2.4. CMP 2.0

The following describes major features of CMP 2.0.

8.2.4.1. Entity Bean Relationship Mapping

In EJB 2.0 specification, the concept of CMP relationships is introduced to describe relationships in the underlying Database. The use of such relationships with CMP 2.0 beans requires that the deployer provides additional information in the JEUS-specific EJB DD.

8.2.4.2. JEUS EJB QL Extension

CMP 2.0 type beans must have any findByXXXX() method, that is associated with an EJB-QL statement of the ejb-jar.xml deployment descriptor, declared in the home interface. JEUS supports a number of additions to the standard EJB-QL language. These additions may be used either in the ejb-jar.xml file or in an instant EJB QL request.

Refer to Using JEUS EJB QL Extension for a complete ejb-jar.xml example.

8.2.4.3. Instant EJB QL

When you need to find a set of CMP beans within client code, you are usually forced to rely on the find ByXXXX() methods that were declared in the bean home interface. In some cases, when the search criteria is complex, the findByXXXX() method alone might not be sufficient.

For those cases, JEUS offers a non-standard interface that you may use in order to define your own EJB-QL select query directly inside the client code. This interface is called jeus.ejb.Bean.objectbase.EJBInstanceFinder, and it will be implemented by the entity bean's home interface when instant QL of the bean is enabled in the JEUS EJB DD. The interface contains one method, called findWithInstantQL (String ejbQLString), that allows you to forward the user's EJB-QL to the home interface. The method will return a java.util.Collection object of an EJB interface that matches the query.

See Automatic Primary Key Generation for an example of using this method. Instant EJB QL API Reference contains a formal description of this API.

8.3. Configuring Entity EJBs

The following are the configuration methods for JEUS entity EJB:

- Configuration applied to all entity beans
  - The basic, shared settings
  - Object Pool
  - Persistence optimizations including the entity cache settings
- Configuration applied to CMP types only
  - CM persistence optimization
  - Schema information

- Configuration applied to CMP 2.0 only

    ◦ Relationships mapping

    ◦ Instant EJB QL

All of these properties will be configured inside the **\<beanlist\>** tag using the **\<jeus-bean\>** tag in the JEUS EJB module DD file (jeus-ejb-dd.xml).

8.3.1. Commonly Used Functions

The following explains the basic settings applicable to all entity beans.

8.3.1.1. Configuring the Common Basic Functions

Refer to Common Characteristics of EJB for information about basic EJB settings that apply to every kind of bean in JEUS. Those settings will also apply to all types of entity beans.

The following is an example of basic XML tags for a BMP entity bean.

Configuring the Basic Shared Settings for Entity EJB: \<jeus-ejb-dd.xml\>

```
<jeus-ejb-dd>
    . . .
    <beanlist>
        <jeus-bean>
            <ejb-name>account</ejb-name>
            <export-name>ACCOUNTEJB</export-name>
            <local-export-name>
                LOCALACCOUNTEJB
            </local-export-name>
            <export-port>7654</export-port>
            <export-iiop>true</export-iiop>
            . . .
        </jeus-bean>
        . . .
    </beanlist>
    . . .
</jeus-ejb-dd>
```

8.3.1.2. Configuring Object Management

The following is an example of XML where object management is configured using the **\<jeus-bean\>** element, which is inside the **\<object-management\>** element.

Configuring Object Management: \<jeus-ejb-dd.xml\>

```
<jeus-ejb-dd>
    . . .
    <beanlist>
        <jeus-bean>
            . . .
```

```
            <object-management>
                <bean-pool>
                    <pool-min>10</pool-min>
                    <pool-max>200</pool-max>
                </bean-pool>
                <connect-pool>
                    <pool-min>10</pool-min>
                    <pool-max>200</pool-max>
                </connect-pool>
                <capacity>5000</capacity>
                <passivation-timeout>10000</passivation-timeout>
                <disconnect-timeout>1800000</disconnect-timeout>
            </object-management>
        </jeus-bean>
    . . .
    </beanlist>
    . . .
</jeus-ejb-dd>
```

The following describes each element:

- **<bean-pool>**
  - Determines the behavior of the bean instance pool.
  - The following describes child tags.

Tag	Description
<pool-min>	Initial number of bean instances created when a pool is initialized and the number of minimum bean instances that will be maintained in a pool (number of instances left in the pool during resizing). (Default value: 0)
<pool-max>	Maximum number of bean instances for a pool that determines whether to save an instance in the pool after the instance is used. Since session beans can be generated up to max number, the number of requests that exceeds the max number may induce an EJB exception. However, an entity bean has no limit in generating bean instances, and it restricts the number of instances that will be maintained in a pool. (Default value: 100)
<resizing-period>	Time interval for resizing a pool. Resizing means that unused bean instances are removed, and only minimum number of bean instances are left in the pool. (Default value: 5 minutes, Unit: ms)

- **<connect-pool>**
  - Determines the number of client and bean instance connections to keep (This option is not used in session beans).
  - The following describes child tags.

Tag	Description
<pool-min>	Minimum number of instances to be maintained in a pool during resizing. (Default value: 0)
<pool-max>	Maximum number of connections for a pool that is used to determine whether to save a connection in the pool after the instance is used. (Default value: 100)
<resizing-period>	Time interval for resizing a pool. Resizing means that unused bean instances are removed, and only minimum number of bean instances are left in the pool. (Default value: 5 minutes, Unit: ms)

- **<capacity>**

    ◦ The maximum number of bean instances to be created. This applies only for entity beans. This value is used to efficiently configure connections between internal client session data and EJB. (Default value: 10000)

- **<passivation-timeout>**

    ◦ Timeout interval for inactivating a bean in the EJB engine, for which there are no client requests during this time.

    If no client request is made for a bean during the timeout period, that bean instance is subject to passivation. The number of bean instances in memory that will be subject to passivation can be adjusted with <persistence-optimize> / <entity-cache-size> tags.

    ◦ Since there is no entity cache for a stateful session bean, it is subject to passivation If no client request is made for a bean during this time. An entity bean, however, remains in the entity cache before becoming subject to passivation. The interval for checking for beans to passivate depends on the <resolution> value set in EJBMain.xml (Default value: 5 minutes).

    If passivation is executed, the bean instance is removed from memory, and its state is archived in a file or database.

    ◦ These settings can be configured at several locations, and their priorities are as follows:

        1. Applied to only certain session beans: <passivation-timeout> of jeus-ejb-dd.xml.

        2. Applied to all entity beans: system property jeus.ejb.entity.passivate.

        3. Applied to all EJB beans (all entity beans and session beans): system property jeus.ejb.all.passivate.

    If no setting exists, default value is 300,000ms (5 minutes).

- **<disconnect-timeout>**

    ◦ Used for disconnecting a connection between the client and a bean instance if there are no client requests during the timeout period. This means that connections to clients and instances are disconnected and returned to a connection pool.

    Therefore, a client can no longer make a request through this connection. Bean instances in use are removed, or returned to a bean pool if the bean pool is used.

◦ These settings can be configured at several locations, and their priorities are as follows:

1. Applied to only certain session beans: <disconnect-timeout> of jeus-ejb-dd.xml.

2. Applied to all entity beans: system property jeus.ejb.stateful.disconnect.

3. Applied to all EJB beans (all entity beans and session beans): system property jeus.ejb.all.disconnect.

If none of these setting are configured, the system property 'jeus.ejb.stateful.disconnect' default value is set to 3600000 ms (1 hour).

> The time period used for <passivation-timeout> and <disconnect-timeout> are both measured from the time of last access to the bean instance. This means that you should set the <disconnect-timeout> period to a value considerably larger than the <passivation-timeout> value. The <passivation-timeout> value should be larger than the resolution specified in the EJBMain.xml file.
>
> If the timeout value is too large, a lot of instances, that are inactivated, end up remaining in the memory for a long period (more than 10 minutes or when timeout expires), wasting system resources. Too short of a timeout period (several seconds), however, may undermine the performance due to frequent inactivations/activations, and has the risk of losing the session data.

8.3.1.3. Configuring ejbLoad() and ejbStore() Persistence Optimization

Optimizations for persistence method invocation are configured in DD of each bean of the jeusejb-dd.xml file.

The following is an XML example in which persistence of ejbLoad() and ejbStore() methods is configured in the **<persistence-optimize>** element inside the **<jeus-bean>** element.

Configuring ejbLoad() and ejbStore() Persistence Optimization: <jeus-ejb-dd.xml>

```
<jeus-ejb-dd>
    . . .
    <beanlist>
        . . .
        <jeus-bean>
            . . .
            <persistence-optimize>
                <engine-type>SINGLE_OBJECT</engine-type>
                <non-modifying-method>
                    <method-name>myBusinessmethod</method-name>
                    <method-params>
                        <method-param>
                            java.lang.String
                        </method-param>
                        <method-param>int</method-param>
                        <method-param>double</method-param>
                    </method-params>
                </non-modifying-method>
```

```
            <non-modifying-method>
                <method-name>myBusinessmethod2</method-name>
                <method-params>
                    <method-param>
                        java.lang.String
                    </method-param>
                    <method-param>int</method-param>
                </method-params>
            </non-modifying-method>
            <entity-cache-size>500</entity-cache-size>
            <update-delay-till-tx>
                false
            </update-delay-till-tx>
            <include-update>
                true
            <include-update>
        </persistence-optimize>
    </jeus-bean>
    . . .
  </beanlist>
  . . .
</jeus-ejb-dd>
```

The following describes each element:

Tag	Description
<engine-type>	Set to one of the following. For more information about the differences between these types, refer to Key Features. ◦ EXCLUSIVE_ACCESS (Default value) ◦ SINGLE_OBJECT ◦ MULTIPLE_OBJECT
<non-modifying-method>	Provides hints to the EJB engine when it is trying to decide whether the ejbStore() method should be invoked. This element does not apply to CMP 2.0 bean.
<entity-cache-size>	Determines the size of the internal cache for entity bean instances that are subject to inactivation. The size is the maximum number of entity bean instances that can be held in the cache. The higher the value the better the performance, but more system resources (e.g., main memory) will be used. (Default value: 2,000)
<update-delay-till-tx>	Boolean value that determines whether EJB data updates/ inserts should be postponed until the ongoing transaction is committed. If this value is set to "false", all inserts/updates will be performed immediately, ensuring that any read operations within the same transaction will see the changes dynamically before the commit() call. This, however, also results in a lower performance. (Default value: true)

Tag	Description
<include-update>	Specifies the default value for the <schema-info><jeus-query><include-updates> setting. If this value is set to "true", the updates generated while the finder method is called are committed, ensuring that any read operations will see the updated information while the execution of the finder method. (Default value: false)

8.3.2. CMP 1.1/2.0

The following explains applicable settings for CMP 1.1/2.0.

8.3.2.1. ejbLoad(), ejbFind(), CM Persistence Optimization

For CMP 1.1 and 2.0 beans, we may configure optimizations for ejbLoad() implementation persistence. More detailed information was described in Configuring ejbLoad() and ejbStore() Persistence Optimization.

The following is an XML example in which persistence of ejbLoad() and ejbFind() CM is configured through the **<cm-persistence-optimize>** element inside the **<jeus-bean>** element.

Configuring ejbLoad() and ejbFind() CM Persistence Optimization: <jeus-ejb-dd.xml>

```
<jeus-ejb-dd>
    . . .
<beanlist>
        . . .
      <jeus-bean>
            . . .
         <cm-persistence-optimize>
             <subengine-type>WriteLocking</subengine-type>
             <fetch-size>80</fetch-size>
             <init-caching>true</init-caching>
         </cm-persistence-optimize>
      </jeus-bean>
        . . .
   </beanlist>
    . . .
</jeus-ejb-dd>
```

The following describes each element:

Tag	Description

<subengine-type>	Set to one of the following:
	
	   ◦ ReadLocking (Default value)
	
	   ◦ WriteLocking
	
	   ◦ WriteLockingFind

<fetch-size>	Determines the number of rows to fetch at a time from a ResultSet. (Default value: 10)
<init-caching>	Boolean switch, if enabled, that causes the EJB engine to create pre-instantiated EJB entity instance for each row in the given database table. This pre-instantiation is carried out when the engine is started. If disabled, EJB instances will only be created in response to create(), find-ByXXXX(), or similar method calls. (Default value: false)

8.3.2.2. Configuring DB Schema Information

In CMP 1.1/2.0 you must configure the schema information, which contains mappings from EJB instance fields to database tables and columns.

The following is an XML example of DB schema information configured in the **<schema-info>** element inside the **<jeus-bean>** element.

Configuring DB Schema Information: <jeus-ejb-dd.xml>

```
<jeus-ejb-dd>
    . . .
    <beanlist>
       . . .
      <jeus-bean>
          . . .
          <schema-info>
              <table-name>ACCOUNT</table-name>
              <cm-field>
                  <field>id</field>
                  <column-name>ID</column-name>
                  <type>NUMERIC</type>
                  <exclude-field>false</exclude-field>
              </cm-field>
              <cm-field>
                  <field>customer_addr</field>
                  <column-name>customer_address</column-name>
                  <type>VARCHAR(30)</type>
                  <exclude-field>false</exclude-field>
              </cm-field>
              <creating-table>
                  <use-existing-table/>
              </creating-table>
              <deleting-table>true</deleting-table>
              <prim-key-field>
                  <field>id</field>
```

```
                </prim-key-field>
                <jeus-query>
                    <method>
                        <method-name>findByAddress</method-name>
                        <method-params>
                            <method-param>java.lang.String</method-param>
                        </method-params>
                    </method>
                    <sql>customer_address=?</sql>
                </jeus-query>
                <jeus-query>
                    <query-method>
                        <method-name>findByTitle</method-name>
                        <method-params>
                            <method-param>java.lang.String</method-param>
                        </method-params>
                    </query-method>
                    <jeus-ql>
                        SELECT OBJECT(b) FROM Book b
                        WHERE b.title = ?1 ORDERBY b.price
                    </jeus-ql>
                </jeus-query>
                <db-vendor>oracle</db-vendor>
                <data-source-name>MYDB</data-source-name>
            </schema-info>
            . . .
        </jeus-bean>
        . . .
    </beanlist>
    . . .
</jeus-ejb-dd>
```

The following describes each element:

- **<table-name>**

    ◦ The name of the relational database table to which the current EJB should be mapped to.

    ◦ If not set, an arbitrary table name will be set using the <EJB module name> and <EJB name> elements.

    ◦ Due to certain restrictions on the length of the table names in some DBMSs, only 15 characters are allowed for a table name.

- **<cm-field>**

    ◦ The container managed field mappings exist for each column-to-field relationships.

    ◦ The following describes child tags.

Tag	Description
<field>	Specifies the name of an EJB field to which the column should be mapped to.
<column-name>	Field name as appears in the given database table. (If not set, the name of the EJB field is used.)

Tag	Description
<type>	Data type defined in the database column, such as VARCHAR(25) and NUMERIC. If this element is not specified, the default type for each database will be used. This mapping information is explained in Basic Java Type and Database Field Mappings. Oracle DBMS supports two types used for storing large data such as images. ◦ Character Large Object (CLOB): Used for java.lang.String fields. ◦ Binary Large Object (BLOB): Used for serializable fields.
<exclude-field>	If enabled, this element causes the specified field not to be mapped and included in the persistence management. This option works for CMP 2.0 beans only. It is included for migration purposes. (Default value: false)

- **<creating-table>**

  - Checks whether a table to be used by a bean exists in the Database when the bean starts. An existing table is used, a new table is created, or an error occurs according to the child elements.

  - Checks whether a bean contains a table with the name specified in <schema-info>/<table-name>, if the fields specified in <schema-info>/<cm-field> exist, and if foreign keys specified in <ejb-relation-map>/<jeus-relationship-role>/<column-map> exist for a bean that manages relationships.

  - The following describes child tags.

Tag	Description
<use-existing-table>	Indicates that a table is created only when it does not exist. Otherwise, the existing table is used.
<force-creating-table>	Indicates that the existing table is deleted and a new table is created.

  When these child elements are not specified and if a table already exists, an error will occur. If a table does not exist, a new table will be generated.

  - If there are no child elements configured, the "-Djeus.ejb.checktable" setting in an MS JVM parameter is used to determine whether to check for the table.

Value	Description
true	Indicates that an error occurs if the table is not found during the check. (Default)

Value	Description
false	Indicates that since table check is not performed when a bean is started, an error may occur when trying to use the table.

- **\<deleting-table\>**
  - If enabled, causes the given table to be removed from the Database when the EJB module is undeployed.
  - Since this option should be used with caution, it has to be separately configured as a JEUS system property to prevent any configuration errors. It only works when the Command setting in the JEUSMain.xml file is set to the value, "-Djeus.ejb.enable.configDeleteOption=true".

    In addition, the \<deleting-table\> setting is hardly used without the \<creating-table\> setting. Hence it is implemented so that the \<deleting-table\> setting does not execute without the \<creating-table\> setting.

- **\<prim-key-field\>**
  - This is optionally used for when a composite key is used as the primary key. It can only be used if a primary key class implemented by a user for a compound key is specified in ejb-jar.xml/\<prim-key-class\>. Basically, all public fields defined in \<prim-key-class\> are configured as a primary key, but, if the user wants to configure it by selecting a specific field, this list must be configured.
  - Make sure not to confuse the following elements.
    - Composite field key

Element	Description
ejb-jar.xml/\<prim-key-class\>	Primary key class implemented by the user. (Required)
ejb-jar.xml/\<primkey-field\>	Cannot be declared. The type defined in \<prim-key-class\> cannot be mapped to a single field name, because it includes multiple fields. (Optional)
jeus-ejb-dd.xml/\<prim-key-field\>	Declares a list of fields to be configured as a primary key, instead of configuring all public fields of a primary key class as a primary key. (Optional)

    - Single field key

Element	Description
ejb-jar.xml/\<prim-key-class\>	Ttype of ejb-jar.xml/\<primkey-field\>, such as java.lang.String, java.lang.Integer, etc. (Required)

Element	Description
ejb-jar.xml/<primkey-field>	Field name of an entity bean for a primary key. A field name defined in <primkey-field> and a type defined in <prim-key-class> must be mapped to a name and type defined by an entity bean class, respectively. (Optional)
jeus-ejb-dd.xml/<prim-key-field>	Only declared when selecting one or more among multiple fields. (Optional)

- **<jeus-query>**

  - For CMP 1.1, you must specify SQL statements for any finder methods that should be implemented by the EJB engine. In CMP 2.0, this element overrides the EJB QL configured in ejb-jar.xml.

    The <jeus-query> setting enables query methods(findXXX()) to use the standard EJB QL and JEUS EJB QL extensions. This is similar to the <query> element of ejb-jar.xml. For more information about the <query> element, refer to Using JEUS EJB QL Extension. This element mainly exists for easier migration from BEA WebLogic application server to JEUS 4.2.

  - SQL statement and the name and arguments of the finder method, that will use the SQL statement, must be specified to define SQL. This SQL statement is used when generating the finder method.

    The specified text must only include the part after the "where" keyword. The rest is given implicitly. In the SQL string you may specify "?" characters that will be substituted by the values of the finder method's arguments. You can use the WHERE clause syntax of ANSI SQL, which includes the "where" keyword.

  - If <include-updates> is set to "true", any commitments done up until invoking the finder method will be written to the database so that the finder method can see the changes.

- **<db-vendor>**

  - Specifies the vendor of the Database you use for this bean.

    The vendor name is used by the EJB engine to make some vendor-specific optimizations. The value you specify here, however, might or might not have an actual impact on the performance. A list of vendor names that you can use are specified in Basic Java Type and Database Field Mappings. Oracle is a commonly used name.

  - This value is also used to determine the correct mapping between Java objects and DB field types defined by the vendor in question.

- **<data-source-name>**

  - Specifies the JNDI name of the DB connection pool used to connect to the Database. This connection pool is usually configured in JEUSMain.xml and is running on the JEUS MS JVM.

- **<auto-key-generator>**

- ◦ Contains child elements for setting up automatic primary key generation for CMP 2.0 beans. For more information about configuring these elements, refer to Automatic Primary Key Generation.

8.3.3. CMP 2.0

The following explains settings applicable to CMP 2.0.

8.3.3.1. Configuring Relationship Mapping

Configuring relationship mapping is done for the following cases.

- One-to-one/One-to-many Relationship Mapping

- Many-to-many Relationship Mapping

One-to-one/One-to-many Relationship Mapping

When setting up a One-to-one/one-to-many relationship between two beans, you must configure the **<ejb-relation-map>** tag.

The following is an example of a many-to-one (= one-to-many) relationship between two beans in a relationship called employee-manager. It is configured in the **<ejb-relation-map>** tag.

Configuring One-to-one/One-to-many Relationship Mapping: <jeus-ejb-dd.xml>

```
<jeus-ejb-dd>
    . . .
    <ejb-relation-map>
        <relation-name>employee-manager</relation-name>
        <jeus-relationship-role>
            <relationship-role-name>employee-to-manager</relationship-role-name>
            <column-map>
                <foreign-key-column>manager-id</foreign-key-column>
                <target-primary-key-column>man-id</target-primary-key-column>
            </column-map>
        </jeus-relationship-role>
    </ejb-relation-map>
    . . .
</jeus-ejb-dd>
```

The following describes each element:

- **<relation-name>**

  Defined in the ejb-jar.xml file.

- **<jeus-relationship-role>**

  Configure two JEUS relationship roles for each direction. The information required for each

relationship role is:

- **<relationship-role-name>**

  Defined in the <ejb-relationship-role-name> element.

- **<column-map>**

  Contains two child tags: <foreign-key-column> and <target-primary-key-column>.

Tag	Description
<foreign-key-column>	Name of the column that acts as a foreign key in a relationship.
<target-primary-key-column>	Primary key column of a table to which the foreign key column is to be mapped. Several column-to-column mappings may be specified if the keys are composed of multiple columns.

In the previous XML example, a foreign key column called "manager-id" in the employee table is mapped to the primary key column of the manager table in order to establish many-to-one employee-to-manager relationship.

One-to-one mappings would work in a completely analogous manner. Just remember that for one-to-one, the primary key column and the foreign key column may exist in either table. For one-to-many, the table on the many side must contain the foreign key column.

Many-to-many Relationship Mapping

In order to establish a many-to-many relationship between two beans, you must provide the <table-name> and <jeus-relationship-role> elements in addition to the information given for one-to-one/one-to-many relationships.

The following is an example of configuring a many-to-many relationship named student-course:

Configuring Many-to-many Relationship Mapping: <jeus-ejb-dd.xml>

```
<jeus-ejb-dd>
    . . .
    <ejb-relation-map>
        <relation-name>student-course</relation-name>
        <table-name>STUDENTCOURSEJOIN</table-name>
        <jeus-relationship-role>
            <relationship-role-name>student</relationship-role-name>
            <column-map>
                <foreign-key-column>
                    student-id <!--This column in join table-->
                </foreign-key-column>
                <target-primary-key-column>
                    stu-id <!--This column in student table-->
                </target-primary-key-column>
            </column-map>
```

```
        </jeus-relationship-role>
        <jeus-relationship-role>
            <relationship-role-name>course</relationship-role-name>
            <column-map>
                <foreign-key-column>
                    course-id <!--This column in join table-->
                </foreign-key-column>
                <target-primary-key-column>
                    cou-id <!--This column in course table-->
                </target-primary-key-column>
            </column-map>
        </jeus-relationship-role>
    </ejb-relation-map>
    . . .
</jeus-ejb-dd>
```

The following describes each element:

Tag	Description
<table-name>	Identifies the join table in the many-to-many relationship.
<jeus-relationship-role>	Two **JEUS relationship roles** that map the two foreign key columns in the join table to the primary key columns of two other tables used by two beans.

In the above example, notice how two JEUS relationship roles are used. Each role declares the mapping between one of two columns of the join table and the primary key column of a table that is used by an EJB.

8.3.3.2. Configuring Instant EJB QL

In order to enable instant EJB QL queries on the client-side, it is sufficient to simply set the **<enable-instant-ql>** element in the JEUS EJB DD to "true." For more information, refer to Instant EJB QL API Reference.

Configuring Instant EJB QL: <jeus-ejb-dd.xml>

```
<jeus-ejb-dd>
    . . .
    <beanlist>
        . . .
        <jeus-bean>
            . . .
            <enable-instant-ql>true</enable-instant-ql>
        </jeus-bean>
        . . .
    </beanlist>
    . . .
</jeus-ejb-dd
```

8.3.3.3. Using JEUS EJB QL Extension

This section describes EJB QL extension related to JEUS CMP.

The following are EJB QL extensions related to JEUS CMP.

- JEUS EJB QL Extension keyword
- JEUS EJB QL Extension Subquery
- JEUS EJB QL Extension ResultSet
- JEUS EJB QL Extension Dynamic Query
- JEUS EJB QL Extension GROUP BY keyword

The home interface must be associated with an EJB-QL statement of the ejb-jar.xml in the findByXXX() and ejbSelectXXX() methods of CMP 2.0 beans. JEUS includes a couple of additions to the standard EJB-QL language. These additions may be used either in the ejb-jar.xml file or in an instant EJB QL request. Refer to CMP 2.0 Entity Bean Example for a complete ejb-jar.xml example which provides a reference for this functionality.

> The EJB QL extensions presented in this section are not supported by standard EJB 2.0 QL, and using them makes your EJB applications non-portable according to the Jakarta EE standard. This means that you can not seamlessly deploy an EJB component, which uses any of these extensions, to another Jakarta EE server. Since most of EJB QL extensions in JEUS work like the standard SQL, refer to SQL for more details.

JEUS EJB QL Extension Keyword Additions

The following non-standard keywords may be used in any EJB QL statements within JEUS:

- **ORACLEHINT** keyword

  Used in the form of "oraclehint<standard Oracle hint string>".

  ```
  SELECT OBJECT(b) ORACLEHINT '/+ALL_ROWS/'
  FROM Book b
  ```

  Refer to the Oracle documentation for detailed information about the keyword.

JEUS EJB QL Extension Subquery

A JEUS EJB QL subquery is an EJB QL query that is nested within the WHERE clause of a parent EJB QL query. This is similar to the relationship between SQL queries and subqueries.

The following example illustrates this concept by constructing a nested EJB QL query that selects all employees with an above-average salary:

```
SELECT OBJECT(e)
FROM EmployeeBean AS e
WHERE e.salary >
    (SELECT AVG(e2.salary)
     FROM EmployeeBean AS e2)
```

- Return Type

  A return type from a JEUS EJB QL query can be:

  - A single value or a collection of values corresponding to the <cmp-field> setting (e.g., e.salary in the previous example).

  - A single return value generated through an aggregate function (e.g., MAX(e.salary)).

  - A cmp-bean with a simple primary key (not compound primary key). Example: SELECT OBJECT(emp).

- The Operands of Comparison Operators.

  JEUS EJB QL supports the use of subqueries as the operands of comparison operators.

  JEUS EJB QL supports the following comparison operators:

  - [NOT] IN, [NOT] EXISTS, [NOT] UNIQUE

  - <, >, <=, >=, =, <>

  - Combination of the previous operators and the following three operators: ANY, ALL, and SOME.

The following example would select all employees who are not managers.

```
SELECT OBJECT(employee)
FROM EmployeeBean AS employee
WHERE employee.id NOT IN
    (SELECT employee2.id
     FROM ManagerBean AS employee2)
```

Following is an example of the **NOT EXISTS** operator that tests for the non-existence of a subquery result. ("NOT" may be removed to test for the existence of the result)

```
SELECT OBJECT(cust)
FROM CustomerBean AS cust
WHERE NOT EXISTS
    (SELECT order.cust_num
     FROM OrderBean AS order
     WHERE cust.num = order.cust_num)
```

The previous query retrieves all customers that have not placed an order.

The last query of the previous example is called a correlated query, since the evaluation of the nested

subquery requires access to a variable defined by the parent query. Queries without such dependencies are called uncorrelated.

The following example shows the use of the **UNIQUE** operator. This operator is used for testing for duplicate rows in a result set. It returns TRUE if the result set has zero or one row, or if every row is unique. Adding NOT to this operator achieves the opposite.

```
SELECT OBJECT(cust)
FROM CustomerBean AS cust
WHERE UNIQUE
    (SELECT cust2.id FROM CustomerBean AS cust2)
```

The following is an example of the **>** operator, which, like all simple arithmetic operators, may be used with a single value result obtained from the subquery to the right of the operator:

```
SELECT OBJECT(employee)
FROM EmployeeBean AS employee
WHERE employee.salary >
    (SELECT AVG(employee2.salary)
     FROM EmployeeBean AS employee2)
```

The previous JEUS EJB QL query would return all employees that earn more than the average salary.

Note that if ANY, ALL, and SOME operators are not used, the nested subquery must produce a single value result (the average salary in this case). The operators ANY, ALL, and SOME are used with subqueries that return more than one value. These operators follow the following rules:

- <Operand> <arithmetic operator> **ANY** <subquery>

  Yields "true" if at least one value in the subquery return set yields "true" for the given operand and arithmetic operator.

- <Operand> <arithmetic operator> **SOME** <subquery>

  Synonym to "ANY" (see above).

- <Operand> <arithmetic operator> **ALL** <subquery>

  Yields "true" if all values in the subquery return set yield "true" for the given operand and arithmetic operator.

> When you use the characters ">" and "<" inside an XML DD file, you must enclose them within CDATA constructs. Otherwise, the XML parser will mistaken these characters for XML tag symbols.

Typing ">" and "<" inside an XML DD File: <ejb-jar.xml>

```
. . .
<query>
```

```
    <description>method finds large orders</description>
    <query-method>
        <method-name>findLargeOrders</method-name>
        <method-params></method-params>
    </query-method>
    <ejb-ql>
  <![CDATA[SELECT OBJECT(o) FROM Order o WHERE o.amount > 1000]]>
    </ejb-ql>
</query>
. . .
```

JEUS EJB QL Extension ResultSet

JEUS EJB engine supports ejbSelect() queries that return a multi-column java.sql.ResultSet object. This means that you may specify a comma-separated list of target fields within the SELECT clause of the ejbSelect() method.

```
SELECT employee.name, employee.id, employee.salary
FROM EmployeeBean AS employee
```

The previous query would create a java.sql.ResultSet object that consists of the name, ID, and salary of all EmployeeBean instances. The ResultSet would contain a set of rows with the columns "name," "ID," and "salary." Each row in the ResultSet would represent a matching entity EJB instance. ResultSets can only be used in this way to return cmp-field values, and not beans or relationship fields.

> When you retrieve an object of the type java.sql.ResultSet through the ejbSelect() method call and complete all necessary tasks, you must call the ResultSet.close() method. The close() method will not be called by the EJB engine.

JEUS EJB QL Extension Dynamic Query

JEUS EJB QL supports creating EJB QL queries programmatically. This is accomplished through the Instant EJB QL function. This short section explains the fundamentals of programming instant EJB QL with JEUS CMP 2.0.

The following code-snippet shows the use of instant EJB QL of the CMP 2.0 Home Interface. The code is purely on the client-side. For this code to work, instant EJB QL has to be correctly configured on the CMP bean in question. This was described in Configuring Entity EJBs.

```
import jeus.ejb.Bean.objectbase.EJBInstanceFinder;
import java.util.Collection;
. . .
Context initial = new InitialContext();
Object objref = initial.lookup("emp");
EmpHome home = (EmpHome)
    PortableRemoteObject.narrow(objref, EmpHome.class);
```

```
EJBInstanceFinder finder = (EJBInstanceFinder) home;
java.util.Collection foundBeans =
    finder.findWithInstantQL ("<EJB QL query sentence>");
// Use foundBeans collection…
. . .
```

The EJB QL query statement in the example should be substituted with a complete EJB QL statement without any arguments (i.e., "?" is not allowed). See Instant EJB QL API Reference for full information about this API.

JEUS EJB QL Extension GROUP BY Keyword

JEUS EJB QL supports the **GROUP BY** and **GROUP BY..HAVING** keywords. They work as follows:

- **<SELECT statement> GROUP BY <grouping column(s)>**

  The SELECT statement is executed, producing a set of rows. The GROUP BY statement then inspects this set for the column(s) included in the <grouping column(s)> list. All rows of this column(s) with the same value are merged into one row, essentially throwing away duplicate values of the column.

  ```
  SELECT employee.departmentName
  FROM EmployeeBean AS employee
  GROUP BY employee.departmentName
  ```

  The previous query would yield a list with all unique departmentNames from the EmployeeBean table.

- **<SELECT statement> GROUP BY <grouping column(s)> HAVING <condition>**

  Statements of this form will work the same way as described in the previous example, but with one addition: only the column groups that satisfy the <condition> setting are included.

  ```
  SELECT employee.departmentName
  FROM EmployeeBean AS employee
  GROUP BY employee.departmentName
  HAVING COUNT(employee.departmentName) >= 3
  ```

  The previous query would thus return a list of all unique departmentNames with more than 2 EmployeeBean instances (i.e., all departments with 3 or more employees).

For example, if you run the EJB QL query on the data in **[Table 1]**, you would get **[Table 2]** as a result.

- **[Table 1]**

id	departmentName
johnsmith	R&D

id	departmentName
peterwright	R&D
lydiajobs	R&D
catherinepeters	Marketing

- **[Table 2]**

departmentName
R&D

This is the result of the HAVING condition that includes only column groups with at least three rows. Refer to SQL reference material for complete information about the GROUP BY and GROUP BY HAVING keywords.

> In JEUS EJB QL, you may use the GROUP BY and GROUP BY HAVING statements in subqueries or top-level queries. In the latter case, a collection of java.lang.String objects are returned. Top-level GROUP BY and GROUP BY HAVING statements cannot return bean objects.

8.3.4. Configuring the DB Insert Delay (CMP Only)

It is possible to specify at which point in time the EJB data should be written to the backend Database when EJB is created in CMP.

Method	Description
ejbCreate	Insert new EJB data after completing the ejbCreate() call but before calling the ejbPostCreate() method.
ejbPostCreate	Insert new EJB data after completing the ejbCreate() and ejbPostCreate() calls. This is used as the default value.

The following Database insert delay setting is configured in the **<database-insert-delay>** element, which is inside the **<jeus-bean>** element of jeus-ejb-dd.xml.

Configuring DB Insert Delay: <jeus-ejb-dd.xml>

```
<jeus-ejb-dd>
    . . .
    <beanlist>
        . . .
        <jeus-bean>
            . . .
            <database-insert-delay>ejbCreate</database-insert-delay>
            . . .
        </jeus-bean>
        . . .
    </beanlist>
```

```
        . . .
    </jeus-ejb-dd>
```

8.4. Tuning Entity EJBs

This chapter summarizes and elaborates further on what has already been mentioned about performance tuning for entity EJBs in JEUS. The tuning tips offered here can be used in addition to the entity bean tuning tips outlined in Common Characteristics of EJB and the Object Management section in Session Bean.

8.4.1. Common

The following explains performance tuning for all entity beans.

- Selecting the Main Engine Mode and Whether or not to Cluster

- Configuring the Entity Cache Size

- Configuring the DB Vendor

Choosing Main Engine Mode and Whether or not to Cluster

The following table outlines the rules that are used to determine the <engine-type> setting and deciding whether to cluster the EJB across several EJB engines.

Request Amount per Bean	Many EJB engines on multiple CPUs	A small number of EJB engines on multiple CPUs	Single EJB engine on single CPU
Frequent	Cluster EJB and use SINGLE_OBJECT mode.	Cluster EJB and use MULTIPLE_ OBJECT mode.	Try to use more EJB engines. Otherwise, do not cluster but use MULITPLE_OBJECT mode.
Moderate	Cluster EJB and use SINGLE_OBJECT mode.	Cluster EJB and use MULTIPLE_ OBJECT mode.	Do not cluster, use EXCLUSIVE_ ACCESS mode.
Occasional	Do not cluster, use EXCLUSIVE_ACCESS mode.	Do not cluster, use EXCLUSIVE_ACCESS mode.	Do not cluster, use EXCLUSIVE_ ACCESS mode.

Of course, if the entity bean's back-end data in the Database is modified by an external non-WAS component, you will be forced to use the SINGLE_OBJECT or MULTIPLE_OBJECT mode. Also, remember that for fail over to work, you need to cluster your beans.

The final consideration that needs to be made when choosing either the SINGLE_OBJECT or MULTIPLE_ OBJECT mode is whether the bean(s) in question have a long transaction processing time. This refers to the average time period between transaction start and commit for a bean. If this time is

very long, it is highly recommended to use the MULTIPLE_OBJECT mode for the concurrency feature. On the other hand, consider using the SINGLE_OBJECT mode for short transactions.

If the previous rules are followed, the EJB engine will invoke the ejbLoad() method in an optimal way.

> The terms "frequent," "moderate," and "occasional" in the previous table are not clearly defined. Some intuition and a lot of testing might be needed to find the optimal setting for a given scenario.

Setting the Entity Cache Size

As we have seen, the entity cache acts as a storage for entity bean instances that have been inactivated. Instances will be drawn from this cache when an entity bean is reactivated.

If you set the **<entity-cache-size>** element to a large value (several hundred instances or so), the performance will be enhanced since new entity bean instances need not be created when inactivated beans are re-activated. However, if memory conservation is the most important issue, be sure to set this value low or disable it altogether (set the size to "0" in order to disable the entity cache).

Setting the DB Vendor

You must properly configure the **<db-vendor>** element. In some cases, the EJB engine can optimize database connections for products from other vendors, like Oracle. For more information about DBMS vendor names that can be used in this tag, refer to "Configuring domain.xml EJB Engine" of the "JEUS XML Reference".

8.4.2. BMP & CMP 1.1

The following explains tuning methods applicable to BMP and CMP 1.1.

Registering Non-modifying Methods

All entity bean business methods, that do not alter the state information in the back-end database, should be registered through the **<non-modifying-method>** element of the JEUS EJB module DD. This allows the EJB engine to optimize the ejbStore() calls. Such tuning method only applies to BMP and CMP 1.1 beans.

8.4.3. CMP 1.1/2.0

The following explains applicable settings for CMP 1.1/2.0.

Choosing Engine Sub-mode

The **<subengine-type>** element enables EJB engine to optimize the ejbLoad() and ejbFInd() methods when using CMP beans.

CMP beans adhere to the following rules when setting the **<subengine-type>** element.

- If your CMP bean is expected to do more reading than writing operations, always use the ReadLocking mode.
- If your CMP bean is expected to do more writing than reading operations, select the WriteLocking or WriteLockingFind mode.

Setting the Fetch Size

The **<fetch-size>** element of CMP beans should be set high if you prioritize high performance at the expense of wasting system memory: a large value would imply a more efficient use of network connections since large amount of data is read in each time. However, this also requires the EJB engine to cache a large amount of data, as wasting system resources, and therefore increases the latency for each individual call.

The reverse is true if you prioritize conservation of system memory and is prepared to sacrifice some (network) performance: in this case, set the value relatively low. The default fetch size is "10". A value of "100" would, in most cases be considered rather high while a value less than "10" is quite low.

Configuring Initial Caching

If you are prepared to sacrifice system memory for higher runtime performance, set the **<init-caching>** value of CMP beans to "true". This will create entity bean instances from all Database rows that are read during the boot-time of the EJB. This will result in a much more time-consuming engine boot sequence but much quicker first-time accesses. The reverse is true if either system memory conservation or engine boot-time is the main concern, in which case init caching should be set to "false". If a Database table that is mapped to an EJB contains a lot of rows (more than several hundreds or thousands of rows, etc.), the <init-caching> setting will need to be set to false. On the other hand, if the EJB is read-only (implying that ejbLoad() will only have to be called once), it is highly recommended to set it to "true".

8.4.4. CMP 2.0

Note that the use of instant EJB QL is generally discouraged since it is both non-portable and slightly inefficient. Use it only in simple situations for tuning CMP 2.0.

8.5. CMP 2.0 Entity Bean Example

In order to show how the EJB code, standard EJB DD, and JEUS-specific DD work together, we will show a complete example of a CMP 2.0 bean called the Book EJB, which models the concept of a

book. This section is uncommented and is only meant to illustrate how a CMP 2.0 bean might be configured in jeus-ejb-dd.xml. In order to deploy this example, package it first and then deploy it as an EJB. For more information, refer to EJB Modules. For more information about using EJB QL in JEUS ejb-jar.xml, refer to Jakarta EE EJB DD.

Remote Interface

Remote Interface : <Book.java>

```
package test.book;

import java.rmi.RemoteException;
import jakarta.ejb.EJBObject;

public interface Book extends EJBObject {
    public String getTitle() throws RemoteException;
    public void setTitle(String title) throws RemoteException;
    public String getAuthor() throws RemoteException;
    public void setAuthor(String author) throws RemoteException;
    public double getPrice() throws RemoteException;
    public void setPrice(double price) throws RemoteException;
    public String getPublisher() throws RemoteException;
    public void setPublisher(String publisher)
        throws RemoteException;
    public String toBookString() throws RemoteException;
}
```

Home Interface

Home Interface : <BookHome.java>

```
package test.book;

import java.rmi.RemoteException;
import jakarta.ejb.CreateException;
import jakarta.ejb.FinderException;
import jakarta.ejb.EJBHome;
import java.util.*;

public interface BookHome extends EJBHome {
    public Book create(String code, String title, String  author,
            double price, String publisher)
        throws CreateException, RemoteException;
    public Book findByPrimaryKey(String code)
        throws FinderException, RemoteException;
    public Collection findByTitle(String title)
        throws FinderException, RemoteException;
    public Collection findInRange(String from, String to)
        throws FinderException, RemoteException;
    public Collection findAll()
        throws FinderException, RemoteException;
}
```

Bean Implementation

Bean Implementation : <BookEJB.java>

```java
package test.book;

import jakarta.ejb.EntityBean;
import jakarta.ejb.EntityContext;

public abstract class BookEJB implements EntityBean {
    public String ejbCreate(String code, String title,
            String author, double price, String publisher) {
        setCode(code);
        setTitle(title);
        setAuthor(author);
        setPrice(price);
        setPublisher(publisher);
        return null;
    }

    public void ejbPostCreate(String code, String title,
        String author, double price, String publisher) {}

    public abstract String getCode();
    public abstract void setCode(String code);
    public abstract String getTitle();
    public abstract void setTitle(String title);
    public abstract String getAuthor();
    public abstract void setAuthor(String author);
    public abstract double getPrice();
    public abstract void setPrice(double price);
    public abstract String getPublisher();
    public abstract void setPublisher(String publisher);

    public String toBookString() {
        return getCode() + "- [" + getTitle() + "] by " +
                getAuthor() + " | " + getPrice() + " | " +
                getPublisher();
    }

    public BookEJB() {}
    public void setEntityContext(EntityContext ctx) {}
    public void unsetEntityContext() {}
    public void ejbLoad() {}
    public void ejbStore() {}
    public void ejbActivate() {}
    public void ejbPassivate() {}
    public void ejbRemove() {}
}
```

Jakarta EE EJB DD

Jakarta EE EJB DD : <ejb-jar.xml>

```xml
<?xml version="1.0"?>
<ejb-jar version="4.0" xmlns="https://jakarta.ee/xml/ns/jakartaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
xsi:schemaLocation="https://jakarta.ee/xml/ns/jakartaee
https://jakarta.ee/xml/ns/jakartaee/ejb-jar_4_0.xsd">
    <enterprise-Beans>
        <entity>
            <ejb-name>BookBean</ejb-name>
            <home>test.book.BookHome</home>
            <remote>test.book.Book</remote>
            <ejb-class>test.book.BookEJB</ejb-class>
            <persistence-type>Container</persistence-type>
            <prim-key-class>java.lang.String</prim-key-class>
            <reentrant>false</reentrant>
            <cmp-version>2.x</cmp-version>
            <abstract-schema-name>Book</abstract-schema-name>
            <cmp-field><field-name>code</field-name></cmp-field>
            <cmp-field><field-name>title</field-name></cmp-field>
            <cmp-field><field-name>author</field-name></cmp-field>
            <cmp-field><field-name>price</field-name></cmp-field>
            <cmp-field><field-name>publisher</field-name></cmp-field>
            <primkey-field>code</primkey-field>
                <query>
                    <query-method>
                        <method-name>findByTitle</method-name>
                        <method-params>
                            <method-param>
                                java.lang.String
                            </method-param>
                        </method-params>
                    </query-method>
                    <ejb-ql>
                        SELECT OBJECT(b) FROM Book b
                        WHERE b.title = ?1 ORDERBY b.price
                    </ejb-ql>
                </query>
                <query>
                    <query-method>
                        <method-name>findInRange</method-name>
                        <method-params>
                            <method-param>
                                java.lang.String
                            </method-param>
                            <method-param>
                                java.lang.String
                            </method-param>
                        </method-params>
                    </query-method>
                    <ejb-ql>
                        SELECT OBJECT(b) FROM Book b
                        WHERE b.title BETWEEN ?1 AND ?2
                    </ejb-ql>
                </query>
                <query>
                    <query-method>
                        <method-name>findAll</method-name>
                        <method-params/>
                    </query-method>
                    <ejb-ql>
                        SELECT OBJECT(b) ORACLEHINT '/+ALL_ROWS/' FROM Book b
                    </ejb-ql>
                </query>
```

```
            </entity>
        </enterprise-Beans>
        <assembly-descriptor>
            <container-transaction>
                <method>
                    <ejb-name>BookBean</ejb-name>
                    <method-name>*</method-name>
                    <method-params/>
                </method>
                <trans-attribute>Required</trans-attribute>
            </container-transaction>
        </assembly-descriptor>
</ejb-jar>
```

> Note the three bold lines in the previous code. They show some additional, non-standard EJB QL keywords that are allowed in JEUS (ORDERBY, BETWEEN, and ORACLEHINT).

JEUS EJB DD

JEUS EJB DD : <jeus-ejb-dd.xml>

```
<?xml version="1.0"?>
<jeus-ejb-dd xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
<module-info/>
    <beanlist>
        <jeus-bean>
            <ejb-name>BookBean</ejb-name>
            <export-name>book</export-name>
            <export-port>0</export-port>
            <export-iiop><only-iiop>false</only-iiop></export-iiop>
            <persistence-optimize>
                <engine-type>EXCLUSIVE_ACCESS</engine-type>
                <non-modifying-method>
                    <method-name>getTitle</method-name>
                </non-modifying-method>
                <non-modifying-method>
                    <method-name>getAuthor</method-name>
                </non-modifying-method>
                <non-modifying-method>
                    <method-name>getPrice</method-name>
                </non-modifying-method>
                <non-modifying-method>
                    <method-name>getPublisher</method-name>
                </non-modifying-method>
                <non-modifying-method>
                    <method-name>toBookString</method-name>
                </non-modifying-method>
            </persistence-optimize>
            <schema-info>
                <table-name>Booktable</table-name>
                <cm-field><field>code</field></cm-field>
                <cm-field><field>title</field></cm-field>
                <cm-field><field>author</field></cm-field>
```

```
            <cm-field><field>price</field></cm-field>
            <cm-field><field>publisher</field></cm-field>
            <creating-table>
                <use-existing-table/>
            </creating-table>
            <deleting-table>true</deleting-table>
            <db-vendor>oracle</db-vendor>
            <data-source-name>datasource1</data-source-name>
        </schema-info>
        <enable-instant-ql>true</enable-instant-ql>
      </jeus-bean>
   </beanlist>
</jeus-ejb-dd>
```

9. Message Driven Bean(MDB)

This chapter describes some precautions for using Message Driven Beans (MDBs) in the JEUS engine.

9.1. Overview

The EJB engine concurrently executes multiple Message Driven Bean (hereafter MDB) instances. It can also process message streams. Since the EJB engine cannot guarantee the accuracy of the order in which messages arrive in the message driven bean class, it does not transmit the messages in the order they are received. Instead, it randomly selects a message to transmit.

MDB also does not guarantee the order of processing the requests. For example, since a message to cancel a reservation may arrive before the message to make a reservation, message processing order should be considered when designing the MDB.

Concurrency is processed when the bean pool obtains an Instance from MDB and delegate it to the request. This is identical to the behavior of the Thread Ticket Pool (TTP) of stateless session beans. For MDB, a <pool-max> setting, instead of the <thread-max> setting, should be used. For more information, refer to Session Bean.

9.2. Configuring MDBs

Some MDB-specific configurations are required in order for the EJB engine to execute MDB beans. Other than the MDB-specific configurations, MDBs share some basic EJB settings with other JEUS bean types.

This section will very briefly discuss:

- The basic settings of MDB in the JEUS EJB DD (a sub-set of the settings described in Common Characteristics of EJB).
- The JMS-specific settings of JEUS MDB that are configured in the JEUS EJB DD file.

> These configurations should be self-explanatory for anyone familiar with MDB.

9.2.1. Configuring Basic Settings

MDB shares a few similar basic settings with the other EJB types. The settings you can set up for each MDB are <ejb-name>, <run-as identity>, and <security-interop> elements.

The following is an example of configuring basic MDB settings.

Configuring Basic Settings: <jeus-ejb-dd.xml>

```
<jeus-ejb-dd>
```

```
        . . .
    <beanlist>
        . . .
        <jeus-bean>
            <ejb-name>order</ejb-name>
            <!--JMS settings goes here (see "Configuring JMS Settings")-->
            <run-as-identity>
                . . .
            </run-as-identity>
            <security-interop>
                . . .
            </security-interop>
            <env>
                . . .
            </env>
            <ejb-ref>
                . . .
            </ejb-ref>
            <res-ref>
                . . .
            </res-ref>
            <res-env-ref>
                . . .
            </res-env-ref>
            <!-- No clustering of MDB -->
        </jeus-bean>
        . . .
    </beanlist>
        . . .
</jeus-ejb-dd>
```

9.2.2. Configuring JMS Settings

Different settings are required for different message types processed by MDB. The JMS connection factory export name is required to process JMS messages, and the resource adaptor is required to process connector messages.

The following is a section of the XML document that shows annotation settings of a MDB class .

Configuring JMS Settings: <MyMDB.class>

```
@MessageDriven(
    activationConfig = {
        @ActivationConfigProperty(propertyName="destinationType",
                                  propertyValue="jakarta.jms.Queue"),
        @ActivationConfigProperty(propertyName="acknowlegeMode",
                                  propertyValue="Auto-acknowledge")
    },
mappedName = "jms/QUEUE1"
)
public class MyMDB implements MessageListener {

...

}
```

Configuring JMS Settings: <jeus-ejb-dd.xml>

```
<jeus-ejb-dd>
        . . .
    <beanlist>
        . . .
        <jeus-bean>
            . . .
            <ejb-name>MyMDB</ejb-name>
            <connection-factory-name>
                QueueConnectionFactory
            </connection-factory-name>
          <destination>jms/QUEUE1</destination>
            <max-message>15</max-message>
            <activation-config>
                <activation-config-property>
                    <activation-config-property-name>
                        destinationType
                    </activation-config-property-name>
                    <activation-config-property-value>
                        jakarta.jms.Queue
                    </activation-config-property-value>
                </activation-config-property>
                <activation-config-property>
                    <activation-config-property-name>
                        acknowlegeMode
                    </activation-config-property-name>
                    <activation-config-property-value>
                        Auto-acknowledge
                    </activation-config-property-value>
                </activation-config-property>
            </activation-config>
            . . .
        </jeus-bean>
            . . .
    </beanlist>
        . . .
</jeus-ejb-dd>
```

The following describes each element:

- **<connection-factory-name>**

    ◦ A JNDI export name of the JMS connection factory.

    ◦ This setting can be configured from several locations as shown in the following example, and their priorities are as follows:

        1. jeus-ejb-dd.xml/<jeus-bean>/<connection-factory-name>

        2. jeus-ejb-dd.xml/<jeus-bean>/<activation-config> property name : jeus.connectionFactoryName

        3. @ActivationConfigProperty property name : jeus.connectionFactoryName

- **<mdb-resource-adapter>**

    ◦ A resource adaptor that is used to connect to the Messaging System through JEUS connector. Use the module id specified in jeus-connector-dd.xml of the resource adapter.

- **<destination>**

  - A JNDI name of JMS Destination.

  - This setting can be configured from several locations as shown in the following example, and their priorities are as follows:

    1. jeus-ejb-dd.xml/<jeus-bean>/<destination>

    2. <message-driven>/<mapped-name> of ejb-jar.xml

    3. mappedName of @MessageDriven

    4. <message-driven>/<message-destination-link> of ejb-jar.xml

- **<jndi-spi>**

  - Used by MDB when using a JMS service which is registered to a JNDI naming service other than the default one (jeus.jndi.JNSContextFactory). That is, it is used when connecting JEUS MDB to a service, such as IBM MQ or SONIC MQ, other than JEUS JMS service.

- **<max-messages>**

  - Used to specify the maximum number of messages that can be assigned to each session for registering and processing multiple JMS session.

  - If the number of messages in a queue is smaller than the specified number, one session continues to process the requests. If the number is larger than the specified number, another session is used. When the specified number is 10, one session processes the messages until the number of messages exceeds 10. This helps reduce the load. For details, refer to the JMS specification.

- **<activation-config>**

  - Used to override activation config of ejb-jar.xml with the activation config which will be used for JMS or resource adapter.

  - For JMS MDB, acknowledgeMode, messageSelector, destinationType, and subscriptionDurability properties are recognized by default. You may additionally use the previously mentioned properties, jeus.connectionFactoryName, jeus.clientId, and jeus.subscriptionName. The jeus.clientId and jeus.subscriptionName properties are provided for DurableSubscription of the Topic. If they are not specified, the "Module Name"/"Bean Name" are used for them, respectively.

  - This setting can be configured from several locations as shown in the following example, and their priorities are as follows:

    1. <activation-config> in jeus-ejb-dd.xml

    2. <activation-config> in ejb-jar.xml

    3. @ActivationConfig

- **<ack-mode>**

  - Specifies the acknowledge mode of the JMS session.

  - This setting can be configured from several locations as shown in the following example, and their priorities are as follows:

    1. jeus-ejb-dd.xml/<jeus-bean>/<ack-mode>

2. jeus-ejb-dd.xml/<jeus-bean>/<activation-config> property name : acknowlegeMode

3. ejb-jar.xml/<message-driven-bean>/<activation-config> property name : acknowlegeMode

4. ejb-jar.xml/<message-driven-bean>/<acknowledge-mode>

5. @ActivationConfigProperty property name : acknowledgeMode

- **<durable>**

  ◦ Specifies durable subscriber of JMS.

  ◦ This setting can be configured from several locations as shown in the following example, and their priorities are as follows:

    1. jeus-ejb-dd.xml/<jeus-bean>/<durable>

    2. jeus-ejb-dd.xml/<jeus-bean>/<activation-config> property name : subscriptionDurablity

    3. ejb-jar.xml/<message-driven-bean>/<activation-config> property name : subscriptionDurablity

    4. ejb-jar.xml/<message-driven-bean>/<subscription-durability>

    5. @ActivationConfigProperty property name : subscriptionDurablity

- **<msg-selector>**

  ◦ Specifies the JMS message selector.

  ◦ This setting can be configured from several locations as shown in the following example, and their priorities are as follows:

    1. jeus-ejb-dd.xml/<jeus-bean>/<msg-seletor>

    2. jeus-ejb-dd.xml/<jeus-bean>/<activation-config> property name : messageSelector

    3. ejb-jar.xml/<message-driven-bean>/<activation-config> property name : messageSelector

    4. ejb-jar.xml/<message-driven-bean>/<message-selector>

    5. @ActivationConfigProperty property name : messageSelector

> For more information on the above settings, refer to "JEUS MQ Guide", "JEUS JCA Guide", and the JMS standard and EJB standard.

9.2.3. Configuring JNDI SPI

JEUS MDB basically uses JMS connections by looking them up with a JEUS Naming Service.

If you are using another JMS service (IBM MQ or SONIC MQ), connections may be obtained through the service. In this case, the MDB using an external naming service, including a JMS service, may be configured. You may add a **<jndi-spi>** element for each MDB that requires it to look up a JMS service.

The following is an example of configuring JNDI SPI.

Configuring JNDI SPI: <jeus-ejb-dd.xml>

```
<jeus-ejb-dd>
```

```
        . . .
    <beanlist>
        . . .
        <jeus-bean>
            . . .
            <jndi-spi>
                <mq-vendor>SONICMQ</mq-vendor>
                <initial-context-factory>
                    acme.jndi.ACMEContextFactory
                </initial-context-factory>
                <provider-url>
                    protocol://localhost:2345
                </provider-url>
            </jndi-spi>
            . . .
        </jeus-bean>
            . . .
    </beanlist>
        . . .
</jeus-ejb-dd>
```

Descriptions for each element are as follows:

Tag	Description
<mq-vendor>	Name of the MQ/JMS vendor from which the MDB will obtain a connection (through the JNDI Naming Service configured in this element). Legal values are "IBMMQ" and "SONICMQ".
<initial-context-factory>	Class name of the JNDI InitialContext Factory for the naming server is used to connect with the JMS service.
<provider-url>	URL used to connect to the naming service.

10. EJB Timer Service

EJB timer service is the service which enables you to receive timer callbacks periodically or at a specified time. For basic usage of the timer service, refer to EJB specification.

This chapter describes how to configure and use the timer service provided in JEUS EJB.

10.1. Configuring the Timer Service

In general, the JEUS EJB timer service follows the specification, but the function to manage the timer persistently can be used according to the desired performance and user's need.

The following settings can be configured for the timer service.

- EJB Engine's Timer Service

  Enables the shared properties and persistent timer service that are applied to all the beans using EJB timer service.

- jeus-ejb-dd.xml

  Configures how to manage Persistent Timers when deploying and updeploying each bean.

10.1.1. Configuring the Persistent Timer Service (EJB Engine)

The persistent timer service of an EJB engine can be configured by using domain.xml as follows.

Configuring Persistent Timer Service for an EJB Engine: <domain.xml>

```
<ejb-engine>
    <resolution>1000</resolution>
    <use-dynamic-proxy-for-ejb2>true</use-dynamic-proxy-for-ejb2>
    <enable-user-notify>false</enable-user-notify>
    <timer-service>
        <support-persistence>true</support-persistence>
        <max-retrial-count>1</max-retrial-count>
        <retrial-interval>5000</retrial-interval>
        <thread-pool>
            <min>2</min>
            <max>30</max>
            <period>3600000</period>
        </thread-pool>
        <database-setting>
            <data-source-id>tibero_XA1</data-source-id>
            <db-vendor>tibero</db-vendor>
        </database-setting>
    </timer-service>
</ejb-engine>
```

The following describes each configuration items of Persistent Timer Service in the **Advanced**

Options.

- **Thread Pool**

  The thread pool used for the timer service to execute the timeout() method is configured as follows.

Tag	Description
Min	Minimum number of threads in the pool.
Max	Maximum number of threads in the pool.
Period	Time period for which idle threads can remain in the pool before removal.

- **Database Setting**

  The following table describes the mandatory option settings in **Advanced Options**. Since the persistent timer internally uses CMP bean to access the database, the CMP bean is configured through the following options. Therefore, each item in CMP bean is identical to the schema settings of a CMP bean.

Tag	Description
DB Vendor	Database vendor used by the Timer CMP bean.
Data Source ID	DataSource name used by the Timer CMP bean.

10.1.2. Processing the Persistent Timer (jeus-ejb-dd.xml)

When an EJB Engine is configured to use a persistent timer service, the user can decide whether to use the persistent timer service for each bean, use the persistent timer that remains in the database before deploying it, or discard it. This can be done through the jeus-ejb-dd.xml file. Persistent Timer Service can be configured in domain.xml. For more information, refer to Configuring the Persistent Timer Service (EJB Engine).

The following is an example of jeus-ejb-dd.xml in which persistent timer processing is configured through the **<timer-service>** element inside the **<jeus-bean>**.

Processing Persistent Timer: <jeus-ejb-dd.xml>

```
<jeus-ejb-dd>
    . . .
    <beanlist>
        . . .
        <jeus-bean>
            . . .
            <timer-service>
                <support-persistence>
                    true
                </support-persistence>
            </timer-service>
            . . .
        </jeus-bean>
```

```
        . . .
    </beanlist>
      . . .
</jeus-ejb-dd>
```

The following describes the timer element:

Tag	Description
<support-persistence>	Determines whether to manage the timer of the bean persistently.

10.1.3. Configuring the Cluster-Wide Timer Service

All EJB engines must use the same EJB timer service. The timer service configured in each MS will be ignored.

The following describes how to set a cluster-wide timer service in domain.xml.

Configuring Cluster-Wide Timer Service: <domain.xml>

```
<cluster>
    <cluster-wide-timer-service>
        <database-setting>
            <data-source-id>tibero_XA1</data-source-id>
            <db-vendor>tibero</db-vendor>
        </database-setting>
    </cluster-wide-timer-service>
</cluster>
```

The description of each setting item is the same as the **'Database Setting'** items in Configuring the Persistent Timer Service (EJB Engine).

10.2. Configuring Timer Monitoring

EJB timers can be monitored or cancelled using the console tool.

Using the Console Tool

An EJB timer that runs on the server can be monitored or cancelled using the console tool.

- **Monitoring**

   Information about an EJB timer can be displayed by using the **ejb-timer-info** command.

   ```
   ejb-timer-info -server <server-name>
   ```

- **Cancellation**

  An EJB timer can be cancelled by using the **cancel-ejb-timer** command.

  ```
  cancel-ejb-timer -server <server-name>
  ```

  > For more information about the previous two commands and EJB engine related commands, refer to "EJB Engine Commands" in JEUS Reference Guide.

10.3. Cautions for Timer Service

The following describes cautions to consider when using a timer service.

- **Persistent Timer and JDBC Connection**

  Persistent Timer is stored in the Database, and CMP Bean that manages the Timers uses the Time Service - Database Setting - Data Source Id value of the EJB engine.

  A transaction that includes beans which use the persistent timer also manages the datasource which is used by the Timer CMP bean. Therefore, you need to consider the datasource to determine whether to use LOCALXAResource or XAResource.

  > For detailed information about the datasource, refer to "JEUS Server Guide".

11. EJB Client

This chapter describes an example of EJB client programming and how to configure the InitialContext object.

11.1. Overview

An EJB client is any kind of application that accesses and invokes business logic of EJB components deployed inside the EJB engine. EJB clients thus include servlets, applets, other EJBs, standalone Java programs, etc.

This chapter covers the basic information needed to implement and execute an EJB client through a standalone Java program.

> For more information about client container applications, refer to "JEUS Application Client Guide".

11.2. Programming a Client to Access EJB

The following example illustrates the implementation of a stand-alone Java application client that looks up an EJB with the export name HelloApp and invokes a business method, sayHello(), on it.

Programming a Client to Access EJB: <HelloClient.java>

```java
package hello;
import java.rmi.*;
import jakarta.ejb.*;
import javax.naming.*;
import javax.rmi.PortableRemoteObject;

public class HelloClient {
    private void run()
    {
        try {
            //Get JEUS InitialContext
            InitialContext ctx = new InitialContext();

            //Get EJB
            Hello hello = (Hello)ctx.lookup("HelloApp");
            //Call Business Method
            String s = hello.sayHello();
        }
        catch (Exception e)
        { // Exception handling. }
    }

    public static void main(String args[])
    {
        HelloClient hclient = new HelloClient();
```

```
        hclient.run();
    }
}
```

In the previous example, note the bold line **InitialContext ctx = new InitialContext()** where the InitialContext object is acquired. In order for this to execute correctly, you need to set up the attributes of the InitialContext. For more information about Initial Context settings, refer to Configuring InitialContext.

11.3. Configuring InitialContext

InitialContext is an important component required for using the EJBs bound to the JNDI namespace.

It is used to look up all objects specified in the JNDI namespace. EJB uses it to search for actual bean references of the EJB engine. In order to obtain a JEUS InitialContext instance, you need to specify at least one of the five naming context attributes before attempting to acquire an InitialContext, so that your client application will be able to communicate with the JEUS Naming Server (JNSServer).

The following describes how to set up the JNDI InitialContext object either through the use of JVM "–D" properties (recommended approach) or through a hashtable in the EJB client code. It will also show how to actually invoke the EJB client.

The following are five supported naming context environment attributes that must be configured prior to acquiring a JEUS InitialContext object.

- **INITIAL_CONTEXT_FACTORY** (required)

  Must be set to the value, "jeus.jndi.JNSContextFactory". This is the factory class name used to build the naming context.

- **URL_PKG_PREFIXES**

  Set to jeus.jndi.jns.url. Used to look up objects by using the URL scheme of the JEUS InitialContext.

- **PROVIDER_URL**

  Contains the IP address of the JEUS naming server (JNS server) to be used. Default is 127.0.0.1 ("localhost").

- **SECURITY_PRINCIPAL**

  Sets UserInfo that is used during the authentication process of the JEUS naming service.

  If no UserInfo is registered with the execution thread, the default value, guest, will be used. If it is specified later, the UserInfo with a username defined in the JEUS security realm will be used.

- **SECURITY_CREDENTIALS**

Sets a user password that will be used during the authentication process. If the authentication fails, previous UserInfo will be maintained in its previous state. If it succeeds, the new UserInfo will be used.

> In addition to the basic properties presented above, you can set additional properties. For information on those additional properties, refer to "JNDI Naming Server" in JEUS Server Guide.

11.3.1. Configuring Naming Attributes Using JVM Properties

The attributes in the previous sub-section may be configured by using the "-D" switch of the JVM interpreter. The switch should be appended to the "java" command that is used to start the EJB client application.

The following shows how to configure each naming attribute using the "-D" switch.

- **INITIAL_CONTEXT_FACTORY** (required)

```
-Djava.naming.factory.initial=jeus.jndi.JNSContextFactory
```

- **URL_PKG_PREFIXES**

```
-Djava.naming.factory.url.pkgs=jeus.jndi.jns.url
```

- **PROVIDER_URL**

```
-Djava.naming.provider.url=<host_address>
```

- **SECURITY_PRINCIPAL**

```
-Djava.naming.security.principal=<username>
```

- **SECURITY_CREDENTIALS**

```
-Djava.naming.security.credentials=<password>
```

The following example shows how to use the "-D" option to configure the naming context of an EJB client during its invocation. The example shows how a user named "user1" with the password "password1" is used for an EJB client that looks up EJB stubs through a JEUS naming server with the IP address, 192.168.10.10:

```
$ java -classpath
     ${JEUS_HOME}/lib/client/jclient.jar
     -Djava.naming.factory.initial=jeus.jndi.JNSContextFactory
     -Djava.naming.factory.url.pkgs=jeus.jndi.jns.url
     -Djava.naming.provider.url=192.168.10.10
     -Djava.naming.security.principal=user1
     -Djava.naming.security.credentials=password1
     HelloClient
```

> A client library is needed, if the version of an EJB deployed on JEUS is 2.x. This
> library is usually generated when compiling EJB using an appcompiler.
>
> If a class FTP server is used, clients may automatically download and use the
> necessary classes. None is required when using EJB version 3.0 or later.

11.3.2. Configuring Naming Attributes Using a Hashtable

Another possibility for setting the naming context attributes is to use a hashtable and populate it
with naming attribute data directly inside the client code.

The following code fragment shows how this can be done.

```
Context
     ctx = null; Hashtable ht = new Hashtable();
     ht.put(Context.INITIAL_CONTEXT_FACTORY,jeus.jndi.JNSContextFactory");
     ht.put(Context.URL_PKG_PREFIXES, "jeus.jndi.jns.url");
     ht.put(Context.PROVIDER_URL, "192.168.10.10");
     ht.put(Context.SECURITY_PRINCIPAL, "user1");
     ht.put(Context.SECURITY_CREDENTIALS, "password1");
     try { ctx = new InitialContext(ht); . . .
```

In the example, note how the hashtable is passed as the InitialContext object constructor argument.

The following example shows how to execute EJB client when a naming attribute is configured inside
the client class.

```
$ java -classpath
        ${JEUS_HOME}/lib/client/jclient.jar HelloClient
```

> Using a hashtable is not recommended because the values are hard coded.

12. Additional Functions

This chapter describes additional functions that are used to implement EJB in JEUS.

12.1. WorkArea Service

The Work Area service enables a program to continuously propagate a particular implicit context. Like with a security context or transaction context, even if the implicit context is not propagated as a separate argument, it will automatically be transmitted during a local or remote method invocation. Work Area can be used when it is needed to transmit a context or there are multiple method arguments.

As a kind of user repository, Work Area is stored as a map in the form of a name-value pair. When Work Area is newly started, it gets transmitted with the current thread, and so it can continuously be used inside the same component as that of the invoked method or EJB. It also gets propagated during a remote EJB invocation.

The UserWorkArea interface, jeus.workarea.UserWorkArea, is used to access the Work Area service.

1. For detailed information about the API, refer to the jeus.workarea package in JEUS API JavaDoc.

2. Currently, the remote Work Area propagation function is available both in EJB 2.0 and 3.0 version. It is also available when using IIOP invocation instead of JEUS basic RMI invocation.

12.1.1. UserWorkArea Interface

The UserWorkArea interface defines all methods required to start, close, and operate UserWorkArea as follows.

<UserWorkArea Interface>

```
public interface UserWorkArea {
    public void begin(String name);
    public void complete()
            throws NoWorkAreaException, NotOriginatorException;
    public String getName();
    public String[] retrieveAllKeys();
    public void set(String key, java.io.Serializable value)
            throws NoWorkAreaException, NotOriginatorException,
                    PropertyReadOnlyException;
    public void set(String key, java.io.Serializable value, PropertyModeType mode)
            throws NoWorkAreaException, NotOriginatorException,
                    PropertyReadOnlyException;
    public java.io.Serializable get(String key);
    public PropertyModeType getMode(String key);
    public void remove(String key)
```

```
        throws NoWorkAreaException, NotOriginatorException,
            PropertyReadOnlyException;
}
```

> For more information about method definition, refer to JavaDoc.

12.1.2. PropertyMode Type

Each value stored in the WorkArea has its own PropertyMode setting.

Each PropertyMode Type is as follows:

Type	Description
NORMAL	Enables used to modify and delete values registered in UserWorkArea.
READ_ONLY	Disables user to modify and delete values registered in UserWorkArea.

12.1.3. Exceptions

Exceptions defined for UserWorkArea are as follows:

Exception	Description
WorkAreaException	The top-level exception that can occur in WorkArea.
NotOriginatorException	Occurs when trying to modify or delete a value or close a Work Area from a thread that did not initiate the work area.
PropertyReadOnlyException	Occurs when trying to modify a value whose PropertyMode is set to READ_ONLY.

12.1.4. Nested UserWorkArea

UserWorkArea can be nested. If a new UserWorkArea is started while one already exists, the new one is nested inside the existing UserWorkArea.

The nested UserWorkArea can use values from the parent UserWorkArea, and can also add new values. The values added by the nested UserWorkArea are available only in the nested UserWorkArea, and they will be removed when the nested UserWorkArea is terminated.

Registration information in the nested UserWorkAreas is as follows:

Registration Information in Nested UserWorkArea

12.1.5. Developing Application Programs that Use UserWorkArea

This section shows an example of EJBs using the UserWorkArea interface.

The example consists of two EJBs, the UserWorkAreaSampleSender and UserWorkAreaSampleReceiver. In the example, the sender creates the UserWorkArea and transmits data, and the receiver creates a message using the specified value in UserWorkArea and returns the message to the original sender.

1. **UserWorkArea Access**

   To use the UserWorkArea, JNDI has to look up the UserWorkArea.

   The following example shows how JNDI looks up the UserWorkArea.

   Access to UserWorkArea: <UserWorkAreaSampleSenderBean.java>

   ```
   public class UserWorkAreaSampleSenderBean implements UserWorkAreaSampleSender {
       public String getMessage() {
           InitialContext ic;
           String message = null;
           try {
               ic = new InitialContext();
               //Retrieve UserWorkArea from JNDI.
               UserWorkArea userWorkArea
                 = (UserWorkArea) ic.lookup("java:comp/UserWorkArea");
           } catch (NamingException e) {
               // Do Something...
           }
           return message;
       }
   }
   ```

2. **Starting a New UserWorkArea**

A new UserWorkArea needs to be started, because there was no information available for the first UserWorkArea that was looked up in JNDI. If a UserWorkArea name returned is null, NullPointerException occurs.

Starting New UserWorkArea: <UserWorkAreaSampleSenderBean.java>

```
public class UserWorkAreaSampleSenderBean implements UserWorkAreaSampleSender {
    public String getMessage() {
        InitialContext ic;
        String message = null;
        try {
          ic = new InitialContext();
          UserWorkArea userWorkArea
            = (UserWorkArea) ic.lookup("java:comp/UserWorkArea");
          // Initiate a new UserWorkArea.
          userWorkArea.begin("UserWorkArea1");
        } catch (NamingException e) {
            // Do Something...
        }
        return message;
    }
}
```

3. **Configuring Registration in WorkArea**

Configuring registration information in the new UserWorkArea. The registration information includes <key, value, mode>. The key is String, and the value is a serializable object.

If there is no UserWorkArea when configuring the value, a NoWorkAreaException occurs. If the UserWorkArea has not started yet, a NotOriginatorException occurs, and if a value which is already set to READ_ONLY is tried to be modified, a PropertyReadOnlyException occurs.

Configuring Registration in WorkArea: <UserWorkAreaSampleSenderBean.java>

```
public class UserWorkAreaSampleSenderBean implements UserWorkAreaSampleSender {
    public String getMessage() {
        InitialContext ic;
        String message = null;
        try {
            ic = new InitialContext();
            UserWorkArea userWorkArea
              = (UserWorkArea) ic.lookup("java:comp/UserWorkArea");
            userWorkArea.begin("UserWorkArea1");
            // Set userWorkArea as NORMAL.
            userWorkArea.set("name", "johan");
            // Set userWorkArea as READ_ONLY.
            userWorkArea.set("company", "TmaxSoft", PropertyModeType.READ_ONLY);
            UserWorkAreaSampleReceiver receiver
              = (UserWorkAreaSampleReceiver) ic.lookup("receiver");
            message = receiver.createMessage();
        } catch (NamingException e) {
            // Do Something...
        } catch (NoWorkAreaException e) {
            // Do Something...
```

```
        } catch (PropertyReadOnlyException e) {
            // Do Something...
        } catch (NotOriginatorException e) {
            // Do Something...
        }

        return message;
    }
}
```

4. **Obtaining Registration Information Configured in WorkArea**

   Creates a message by using the registration information configured in UserWorkArea that was
   propagated from the receiver. Null is returned if a key, which does not exist in the UserWorkArea,
   is used when getting the information.

   Obtaining Registration Information Configured in WorkArea: <UserWorkAreaSampleReceiverBean.java>

```
public class UserWorkAreaSampleReceiverBean implements UserWorkAreaSampleReceiver {
    public String createMessage() {
        InitialContext ic;
        String message = null;
        try {
            ic = new InitialContext();
            UserWorkArea userWorkArea
              = (UserWorkArea) ic.lookup("java:comp/UserWorkArea");
            // Retrieve settings from UserWorkArea.
            String name = (String)userWorkArea.get("name");
            String company = (String)userWorkArea.get("company");
            message = "Hello " + name + " from " + company;

        } catch (NamingException e) {
            // Do Something...
        }

        return message;
    }
}
```

5. **Completing UserWorkArea**

   Completes the UserWorkArea that was started. Only the originator who started the
   UserWorkArea can complete it, and if others attempt to complete it, a NotOriginatorException
   occurs.

   Completing UserWorkArea: <UserWorkAreaSampleSenderBean.java>

```
public class UserWorkAreaSampleSenderBean implements UserWorkAreaSampleSender {
    public String getMessage() {
        InitialContext ic;
        String message = null;
        try {
            ic = new InitialContext();
            UserWorkArea userWorkArea
              = (UserWorkArea) ic.lookup("java:comp/UserWorkArea");
```

```
            userWorkArea.begin("UserWorkArea1");
            userWorkArea.set("name", "user1");
            userWorkArea.set("company", "TmaxSoft", PropertyModeType.READ_ONLY);
            UserWorkAreaSampleReceiver receiver
              = (UserWorkAreaSampleReceiver) ic.lookup("receiver");
            message = receiver.createMessage();
            userWorkArea.complete(); // Shut down UserWorkArea.

    } catch (NamingException e) {
        // Do Something...
    } catch (NoWorkAreaException e) {
        // Do Something...
    } catch (PropertyReadOnlyException e) {
        // Do Something...
    } catch (NotOriginatorException e) {
        // Do Something...
    }

    return message;
    }
}
```

Appendix A: Basic Java Type and Database Field Mappings

This appendix describes the basic mapping between a Java field type specified by JEUS and a database column type from the main Database vendors supported by JEUS.

A.1. Overview

This basic mapping is used when <schema-info><cm-field><type> of the CMP bean is not specified in jeus-ejb-dd.xml. The EJB system searches for Java fields among CMP fields of the bean to obtain a Java type, and the <type> is assumed to be the same as what is specified for the basic mapping in the next section.

Each of the following tables shows three types of columns.

- **Java field type**

    EJB CMP field types found by searching.

- **SQL type**

    General SQL types defined in the java.sql.Types class.

- **DB column type**

    Names of Database column types to be used by JEUS when creating a new table, based on the Java field types that were found.

A.2. Tibero Field - Column Type Mapping

The following table describes the mapping of the EJB CMP field and database for Tibero.

Java Field Type	SQL Type (java.sql.Types)	Tibero DB Column Type
java.lang.String	VARCHAR	VARCHAR(255)
java.math.BigDecimal	NUMERIC	NUMERIC(15,5)
java.lang.Boolean	BIT	SMALLINT
Boolean	BIT	SMALLINT
java.lang.Byte	TINYINT	SMALLINT
Byte	TINYINT	SMALLINT
java.lang.Character	CHAR	CHAR(4)
Char	CHAR	CHAR(4)
java.lang.Short	SMALLINT	SMALLINT

Java Field Type	SQL Type (java.sql.Types)	Tibero DB Column Type
Short	SMALLINT	SMALLINT
java.lang.Integer	INTEGER	INTEGER
Int	INTEGER	INTEGER
java.lang.Long	BIGINT	NUMERIC(22,0)
Long	BIGINT	NUMERIC(22,0)
java.lang.Float	REAL	REAL
Float	REAL	REAL
java.lang.Double	DOUBLE	DOUBLE PRECISION
Double	DOUBLE	DOUBLE PRECISION
byte[]	LONGVARBINARY	LONG RAW
java.sql.Date	DATE	DATE
java.sql.Time	TIME	DATE
java.sql.Timestamp	TIMESTAMP	DATE
<Java object>	JAVA OBJECT	LONG RAW

A.3. Oracle Field - Column Type Mapping

The following table describes the mapping of the EJB CMP field and database for Oracle.

Java Field Type	SQL Type (java.sql.Types)	Oracle DB Column Type
java.lang.String	VARCHAR	VARCHAR(255)
java.math.BigDecimal	NUMERIC	NUMERIC(15,5)
java.lang.Boolean	BIT	SMALLINT
Boolean	BIT	SMALLINT
java.lang.Byte	TINYINT	SMALLINT
Byte	TINYINT	SMALLINT
java.lang.Character	CHAR	CHAR(4)
Char	CHAR	CHAR(4)
java.lang.Short	SMALLINT	SMALLINT
Short	SMALLINT	SMALLINT
java.lang.Integer	INTEGER	INTEGER
Int	INTEGER	INTEGER
java.lang.Long	BIGINT	NUMERIC(22,0)
Long	BIGINT	NUMERIC(22,0)

Java Field Type	SQL Type (java.sql.Types)	Oracle DB Column Type
java.lang.Float	REAL	REAL
Float	REAL	REAL
java.lang.Double	DOUBLE	DOUBLE PRECISION
Double	DOUBLE	DOUBLE PRECISION
byte[]	LONGVARBINARY	LONG RAW
java.sql.Date	DATE	DATE
java.sql.Time	TIME	DATE
java.sql.Timestamp	TIMESTAMP	DATE
<Java object>	JAVA OBJECT	LONG RAW

A.4. Sybase Field - Column Type Mapping

The following table describes the mapping of the EJB CMP field and database for Sybase.

Java Field Type	SQL Type (java.sql.Types)	Sybase DB Column Type
java.lang.String	VARCHAR	VARCHAR(255)
java.math.BigDecimal	NUMERIC	NUMERIC(15,5)
java.lang.Boolean	BIT	BIT
Boolean	BIT	BIT
java.lang.Byte	TINYINT	TINYINT
Byte	TINYINT	TINYLINT
java.lang.Character	CHAR	CHAR(4)
Char	CHAR	CHAR(4)
java.lang.Short	SMALLINT	SMALLINT
Short	SMALLINT	SMALLINT
java.lang.Integer	INTEGER	INT
Int	INTEGER	INT
java.lang.Long	BIGINT	NUMERIC(22,0)
Long	BIGINT	NUMERIC(22,0)
java.lang.Float	REAL	REAL
Float	REAL	REAL
java.lang.Double	DOUBLE	DOUBLE PRECISION
Double	DOUBLE	DOUBLE PRECISION
byte[]	LONGVARBINARY	IMAGE

Java Field Type	SQL Type (java.sql.Types)	Sybase DB Column Type
java.sql.Date	DATE	DATETIME
java.sql.Time	TIME	DATETIME
java.sql.Timestamp	TIMESTAMP	Not spported
<Java object>	JAVA OBJECT	IMAGE

A.5. Microsoft SQL Server Field - Column Type Mapping

The following table describes the mapping of the EJB CMP field and database for Microsoft SQL Server.

Java Field Type	SQL Type (java.sql.Types)	Microsoft SQL Server DB Column Type
java.lang.String	VARCHAR	VARCHAR(255)
java.math.BigDecimal	NUMERIC	NUMERIC(15,5)
java.lang.Boolean	BIT	BIT
Boolean	BIT	BIT
java.lang.Byte	TINYINT	TINYINT
Byte	TINYINT	TINYINT
java.lang.Character	CHAR	CHAR(4)
Char	CHAR	CHAR(4)
java.lang.Short	SMALLINT	SMALLINT
Short	SMALLINT	SMALLINT
java.lang.Integer	INTEGER	INT
Int	INTEGER	INT
java.lang.Long	BIGINT	NUMERIC(22,0)
Long	BIGINT	NUMERIC(22,0)
java.lang.Float	REAL	REAL
Float	REAL	REAL
java.lang.Double	DOUBLE	FLOAT
Double	DOUBLE	FLOAT
byte[]	LONGVARBINARY	IMAGE
java.sql.Date	DATE	DATETIME
java.sql.Time	TIME	DATETIME
java.sql.Timestamp	TIMESTAMP	DATETIME
<Java object>	JAVA OBJECT	IMAGE

A.6. DB2 Field - Column Type Mapping

The following table describes the mapping of the EJB CMP field and database for DB2.

Java Field Type	SQL Type (java.sql.Types)	DB2 DB Column Type
java.lang.String	VARCHAR	VARCHAR(255)
java.math.BigDecimal	NUMERIC	NUMERIC(15,5)
java.lang.Boolean	BIT	SMALLINT
Boolean	BIT	SMALLINT
java.lang.Byte	TINYINT	SMALLINT
Byte	TINYINT	SMALLINT
java.lang.Character	CHAR	CHARACTER(4)
Char	CHAR	CHARACTER(4)
java.lang.Short	SMALLINT	SMALLINT
Short	SMALLINT	SMALLINT
java.lang.Integer	INTEGER	INTEGER
Int	INTEGER	INTEGER
java.lang.Long	BIGINT	BIGINT
Long	BIGINT	BIGINT
java.lang.Float	REAL	REAL
Float	REAL	REAL
java.lang.Double	DOUBLE	DOUBLE
Double	DOUBLE	DOUBLE
byte[]	LONGVARBINARY	VARCHAR(255)
java.sql.Date	DATE	DATE
java.sql.Time	TIME	TIME
java.sql.Timestamp	TIMESTAMP	TIMESTAMP
<Java object>	JAVA OBJECT	LONG VARCHAR

A.7. Cloudscape Field - Column Type Mapping

The following table describes the mapping of the EJB CMP field and database for Cloudspace.

Java Field Type	SQL Type (java.sql.Types)	Cloudscape DB Column Type
java.lang.String	VARCHAR	VARCHAR(255)
java.math.BigDecimal	NUMERIC	DECIMAL(15,5)

Java Field Type	SQL Type (java.sql.Types)	Cloudscape DB Column Type
java.lang.Boolean	BIT	BOOLEAN
Boolean	BIT	BOOLEAN
java.lang.Byte	TINYINT	TINYINT
Byte	TINYINT	TINYINT
java.lang.Character	CHAR	CHAR(4)
Char	CHAR	CHAR(4)
java.lang.Short	SMALLINT	SMALLINT
Short	SMALLINT	SMALLINT
java.lang.Integer	INTEGER	INTEGER
Int	INTEGER	INTEGER
java.lang.Long	BIGINT	LONGINT
Long	BIGINT	LONGINT
java.lang.Float	REAL	REAL
Float	REAL	REAL
java.lang.Double	DOUBLE	DOUBLE PRECISION
Double	DOUBLE	DOUBLE PRECISION
byte[]	LONGVARBINARY	LONG BIT VARYING
java.sql.Date	DATE	DATE
java.sql.Time	TIME	TIME
java.sql.Timestamp	TIMESTAMP	TIMESTAMP
<Java object>	JAVA OBJECT	LONG BIT VARYING

A.8. Informix Field - Column Type Mapping

The following table describes the mapping of the EJB CMP field and database for Informix.

Java Field Type	SQL Type (java.sql.Types)	Informix DB Column Type
java.lang.String	VARCHAR	VARCHAR(255)
java.math.BigDecimal	NUMERIC	NUMERIC(15,5)
java.lang.Boolean	BIT	VARCHAR
Boolean	BIT	VARCHAR
java.lang.Byte	TINYINT	SMALLINT
Byte	TINYINT	SMALLINT
java.lang.Character	CHAR	CHAR

Java Field Type	SQL Type (java.sql.Types)	Informix DB Column Type
Char	CHAR	CHAR
java.lang.Short	SMALLINT	SMALLINT
Short	SMALLINT	SMALLINT
java.lang.Integer	INTEGER	INTEGER
Int	INTEGER	INTEGER
java.lang.Long	BIGINT	SERIAL8
Long	BIGINT	SERIAL8
java.lang.Float	REAL	REAL
Float	REAL	REAL
java.lang.Double	DOUBLE	DOUBLE PRECISION
Double	DOUBLE	DOUBLE PRECISION
byte[]	LONGVARBINARY	VARCHAR(255)
java.sql.Date	DATE	DATE
java.sql.Time	TIME	DATETIME HOUR TO SECOND
java.sql.Timestamp	TIMESTAMP	DATETIME YEAR TO SECOND
<Java object>	JAVA OBJECT	BYTE

Appendix B: Instant EJB QL API Reference

This appendix describes how to use the instant EJB QL API methods and interfaces.

B.1. Overview

EJB QL API helps to overcome the shortcomings of an EJB finder method by allowing an EJB client application developer to specify an EJB QL query from client-side code. The API consists of only one interface and one method.

> The API should be used in a very extreme situation. It is more fixed and less efficient than the standard EJB entity finder method.

B.2. EJBInstanceFinder Interface

interface jeus.ejb.bean.objectbase.EJBInstanceFinder

This interface is implemented by the home interface for the CMP 2.0 entity bean, when the enable-instant-ql element of jeus-ejb-dd.xml is set to "true." This interface enables you to specify any EJB QL query from client-side code.

```
public abstract interface EJBInstanceFinder extends Remote
```

B.3. EJBInstanceFinder Method

java.util.Collection findWithInstantQL

- Usage

  Returns the EJB group of the bean which corresponds to an EJB QL query that is specified in the ejbQlQuery parameter.

  ```
  java.util.Collection findWithInstantQL (String ejbQlQuery)
  ```

- Parameter

Parameter	Description
string ejbQlQuery	Valid EJB QL statement without "?". This syntax is one of the three additions to EJB QL defined to JEUS.

- Return Value

Return Value	Description
java.util.Collection	Group of bean interfaces that are part of the query result.

- Exception
  - FinderException
  - RemoteException