

Web Engine Guide

JEUS 9

TMAXSOFT

Copyright

Copyright 2025. TmaxSoft Co., Ltd. All Rights Reserved.

Company Information

TmaxSoft Co., Ltd.

TmaxSoft Tower 10F, 45, Jeongjail-ro, Bundang-gu, Seongnam-si, Gyeonggi-do, South Korea

Website: <https://www.tmaxsoft.com/en/>

Restricted Rights Legend

All TmaxSoft Software (JEUS®) and documents are protected by copyright laws and international convention. TmaxSoft software and documents are made available under the terms of the TmaxSoft License Agreement and this document may only be distributed or copied in accordance with the terms of this agreement. No part of this document may be transmitted, copied, deployed, or reproduced in any form or by any means, electronic, mechanical, or optical, without the prior written consent of TmaxSoft Co., Ltd. Nothing in this software document and agreement constitutes a transfer of intellectual property rights regardless of whether or not such rights are registered) or any rights to TmaxSoft trademarks, logos, or any other brand features.

This document is for information purposes only. The company assumes no direct or indirect responsibilities for the contents of this document, and does not guarantee that the information contained in this document satisfies certain legal or commercial conditions. The information contained in this document is subject to change without prior notice due to product upgrades or updates. The company assumes no liability for any errors in this document.

Trademarks

JEUS® is registered trademark of TmaxSoft Co., Ltd.

Java, Solaris are registered trademarks of Oracle Corporation and its subsidiaries and affiliates.

Microsoft, Windows, Windows NT are registered trademarks or trademarks of Microsoft Corporation.

HP-UX is a registered trademark of Hewlett Packard Enterprise Company.

AIX is a registered trademark of International Business Machines Corporation.

UNIX is a registered trademark of X/Open Company, Ltd.

Linux is a registered trademark of Linus Torvalds.

Noto is a trademark of Google Inc. Noto fonts are open source. All Noto fonts are published under the SIL Open Font License, Version 1.1. (<https://www.google.com/get/noto/>)

Other products and company names are trademarks or registered trademarks of their respective

owners.

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

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

1. Web Engine	1
1.1. Overview	1
1.2. Components	1
1.3. Major Functions	3
1.4. Management Tools	5
1.4.1. Types of Tools	5
1.4.2. Controlling and Monitoring	5
1.5. Directory Structure	6
1.6. Environment Configurations	8
1.6.1. XML Configuration Files	8
1.6.2. Monitoring	10
1.6.3. Default Error Page	11
1.6.4. Stack Trace Attachment	11
1.6.5. Encoding	12
1.6.6. JSP Engines	18
1.6.7. Response Headers	18
1.6.8. Cookie Policy	19
1.6.9. Sessions	20
1.6.10. Logs	20
1.6.11. Async Servlet Timeout Processing	26
1.6.12. Web Engine Level Properties	27
1.6.13. Preventing Web Attacks	27
1.6.14. Blocking HTTP Requests with a Specific URL Pattern	28
1.7. Tuning the Web Engine	29
2. Web Connection Management	30
2.1. Overview	30
2.2. Components	31
2.2.1. Listeners	31
2.2.2. Connectors	32
2.2.3. Worker Thread Pool	33
2.3. Configuring Web Connections	34
2.3.1. Common Listener Settings	34
2.3.2. AJP Listeners	38
2.3.3. HTTP Listeners	39
2.3.4. TCP Listeners	39
2.3.5. WebtoB Connectors	39
2.3.6. Tmax Connectors	41
2.4. Configuring Web Server Load Balancing	42

2.4.1. Load Balancing Structure	43
2.4.2. Apache	44
2.4.3. IIS and Iplanet Web Servers	47
2.4.4. Load Balancing with WebtoB	47
2.5. Using TCP Listeners	49
2.5.1. Defining Custom Communication Protocols	50
2.5.2. Implementing Dispatcher Configuration Classes	51
2.5.3. Implementing TCP Servlets	53
2.5.4. Configuring TCP Listeners for Custom Protocols	55
2.5.5. Implementing TCP Clients	56
2.5.6. TCP Client Compilation and Execution	58
2.6. Using HTTP/2	58
2.6.1. Configuring HTTP/2	59
2.6.2. Server Push	60
2.7. Tuning Listeners	61
3. Web Contexts	62
3.1. Overview	62
3.2. Basic Structure	62
3.2.1. Web Context Contents	62
3.2.2. Web Context Structure (WAR File Structure)	62
3.3. Deploying Web Contexts	64
3.3.1. Configuring jeus-web-dd.xml	64
3.3.2. Web Context Redeployment (Graceful Redeployment)	68
3.3.3. Adding Shared Library References	68
3.3.4. Configuring Web Contexts for Compatibility	71
3.3.5. Configuring Additional Features	72
3.3.6. Configuring Web Security	73
3.3.7. Mapping Security Roles	74
3.3.8. Mapping Symbolic References	76
3.4. Monitoring Web Contexts	77
3.5. Controlling Web Contexts	78
3.5.1. Reloading Web Contexts	78
3.5.2. Monitoring a Thread Pool for Asynchronous Requests in a Web Context	78
3.5.3. Suspending Web Contexts	79
3.5.4. Resuming Web Contexts	80
3.6. Tuning Web Contexts	80
3.7. Asynchronous Processing Programming	80
3.7.1. Precautions for Using Servlet Standards	80
3.7.2. Other Considerations	84
4. Thread Pool	86
4.1. Overview	86

4.2. Basic Structure	86
4.3. Configuration	87
4.4. Configuring Automatic Thread Pool State Management (Thread State Notify)	89
4.5. Rules	91
5. JSP Engine	93
5.1. Overview	93
5.2. Apache Tomcat Jasper	94
5.3. JSP Engine Functions	95
5.3.1. JSP Graceful Reloading	95
5.3.2. JSP Precompilation	96
5.3.3. JSP Compilation and Execution in Memory	96
5.4. Configuring JSP Engines	97
5.4.1. Web Engine-Level Configuration	97
5.4.2. Configuring jeus-web-dd.xml	98
5.4.3. Web Context Options for JSP Backward Compatibility	99
5.4.4. Configuring jarscan.properties	100
6. Virtual host	101
6.1. Overview	101
6.2. Web Engine and Virtual Hosts	101
6.2.1. ServletContext Object and Virtual Hosts	103
6.3. Web Context Requests to Virtual Host	103
6.3.1. URL Matching Examples	103
6.3.2. URL Matching Order	105
6.4. Configuring Virtual Hosts	105
6.4.1. Add	105
6.4.2. Modify	106
6.4.3. Delete	106
7. WebSocket Container	107
7.1. Overview	107
7.2. Constraints	107
7.3. WebSocket Container Functions	108
7.3.1. WebSocket UserProperties Failover	108
7.3.2. Other Functions	110
7.4. Configuring WebSocket Container	110
7.5. Using Spring WebSocket	112
8. JEUS WebCache	114
8.1. Overview	114
8.2. JSP Caching	114
8.2.1. General Information	114
8.2.2. <cache> Tag	115
8.2.3. Example of Using <jeus:cache>	118

8.2.4. Using Flush	119
8.2.5. Using Refresh	120
8.3. HTTP Response Caching	120
8.3.1. Configuring Filters	120
9. Reloading Classes at Runtime	123
9.1. Overview	123
9.2. Basic Configurations and Operation	123
9.2.1. Configuring Servers	123
9.2.2. Configuring Applications	124
9.2.3. Applications and Modifications Supported by JEUS HotSwap	125
9.2.4. JEUS HotSwap Constraints	129
10. Valves and Filters	130
10.1. Overview	130
10.2. Concept of Valves	130
10.3. Configuring JEUS-implemented Valves	131
10.3.1. Restricting the Use of HTTP Methods	131
10.3.2. Allowing or Blocking Remote Access	132
10.3.3. URL Rewriting	133
10.4. User Custom Valves	134
10.5. Configuring JEUS-implemented Filters	135
10.5.1. SameSite Restriction	135
10.5.2. Forwarded Header	136
11. Practice	139

1. Web Engine

This chapter describes the components and major functions of JEUS web engine and how to configure it.

1.1. Overview

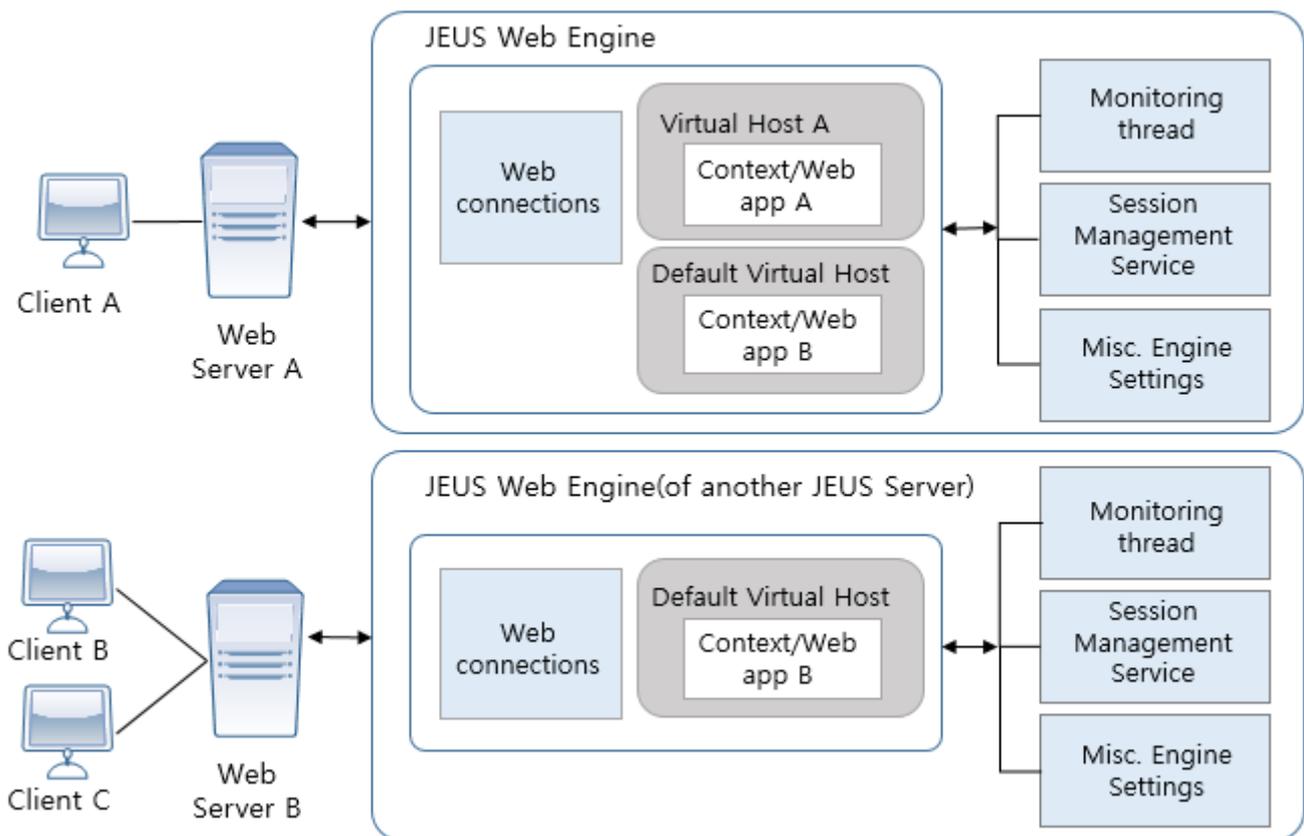
Jakarta EE web applications (hereafter web applications) consist of dynamic web content such as servlets and JSPs, and static resources such as HTML files. TmaxSoft provides JEUS web engine to efficiently handle these web applications.

This chapter introduces the JEUS web engine and describes how to control and monitor it in a production environment.

1.2. Components

This section explains the components of JEUS web engine and the external resources that interact with the engine.

Web engine is an object that manages and runs web applications that conform to the Jakarta EE, servlet, JSP, and EL standards. It can handle dynamic resources like servlets and JSPs, and static resources like HTML.



Web Engine Components



A web engine is also known as a web container or a servlet engine.

The following are the major web engine components.

- **Web Connections**

Web connections allow web engines to connect to client programs, web servers, and other servers.

A large cluster of web servers and web engines is configured by using web connections to improve performance and reliability. For more information, refer to [Web Connection Management](#).

- **Virtual Host**

A virtual host allows client programs to call web applications by host name. A web application can be deployed to a particular virtual host. For more information, refer to [Virtual Hosts](#).

- **Web Context(Web Application)**

When a web application is deployed to a web engine, a web context is created. Web applications are typically packaged as a web archive (WAR) file and deployed to a virtual host in the web engine. For more information, refer to [Web Contexts](#).

- **Monitoring Thread**

Monitoring threads monitor multiple components of the web engine. For more information, refer to [Major Functions](#).

- **Session Management Service**

The session management service provides an environment to manage client sessions in a distributed environment and to use the client sessions in multiple web engines. For detailed information, refer to "Session Tracking" in *JEUS Session Management Guide*.

- **Other Web Engine Configurations**

These include configurations that apply to all web engines such as default error page configurations, encoding configurations, JSP engine configurations, and response header configurations.



The context group that was used in JEUS 6 and earlier is no longer used from JEUS 7 onwards. Most of the context group functions are provided by the web engine.

The following are the external resources that interact with web engines.

- **Clients**

Clients send service requests to JEUS web engines, typically through an HTTP-based web browsers.

- **Web Servers**

Web servers send client HTTP requests to a JEUS web engine. JEUS web engines typically work with web servers to handle the HTTP client requests. For more information, refer to [Web Connection Management](#).

1.3. Major Functions

The following are the major functions of a web engine.

- **Managing web applications**

Web engines can deploy, undeploy, and suspend web applications that conform to the Jakarta EE standards. Web engines can also redeploy web applications while the applications are running to provide seamless service. For more information, refer to [Web Contexts](#).

- **Monitoring**

Web engines monitor resource states to respond if problems occur. There are three types of monitoring. For more information about configuring each monitoring type, refer to [Monitoring](#).

Term	Description
Thread pool monitoring	Monitors thread pools at different levels, such as Web Connection, VirtualHost, Context. For more information, see Web Connection Management .
Class loader monitoring	Monitors modifications made to servlet classes in web applications. For more information, see Web Contexts .
Session service monitoring	Checks and removes expired client sessions. For more information about session service monitoring, refer to "Session Tracking" in <i>JEUS Session Management Guide</i> .

- **Default error page**

Web engines can set the default error page that will be shown if the context is not ready. For more information about this configuration, refer to [Default Error Page](#).

- **Attaching the stack trace**

Web engines can configure the option to attach the stack trace when an error occurs. For more information about this configuration, refer to [Stack Trace Attachment](#).

- **Encoding**

Web engines configure the encoding that applies to all registered contexts. For more information about the configuration, refer to [Encoding](#).

- **JSP engines**

A JSP engine is created for each web application. When clients request JSP resources, the JSP engine converts the JSP pages to Java files, and then compiles the Java files to create the servlet byte code. This process creates Java files, used for debugging jsp files, and class files for each JSP. For more information about this configuration, refer to [Configuring JSP Engines](#).

- **Response headers**

Web engines can configure name-value pairs for user defined HTTP response headers. If a response header is set, the defined header will be included in all responses from the web engine. For more information about this configuration, refer to [Response Headers](#).

- **Managing sessions**

Web engines can configure session management. All configurations related to sessions such as clustered session participation, session object sharing, session cookie configuration, and timeouts can be set. For more information about the configurations, refer to [Sessions](#) and "Session Tracking" in *JEUS Session Management Guide*.

- **Event Logging**

Web engines create server logs and access logs. Web applications create user logs using servlet context log method.

Term	Description
Server logs	Web engine operation logs.
Access logs	Request and result logs.
User logs	Logs that were created in web applications through the <code>jakarta.servlet.ServletContext.log(String msg)</code> or <code>jakarta.servlet.ServletContext.log(String msg, Throwable t)</code> methods.

For information about the default paths to JEUS logs, refer to [Directory Structure](#). For more information about the log configuration, refer to [Logs](#).

- **Graceful Shutdown**

This function guarantees the completion of running services if an administrator shuts down a JEUS server. If the administrator executes the shutdown command, the following two actions occur:

1. The server will no longer receive requests.
2. Services that are already running will complete and then shutdown will continue.

If the services that are already running take too long to complete, JEUS can set the timeout option when running the shutdown command. For more information about graceful shutdown, refer to "Shutting Down a Managed Server" in *JEUS Server Guide*.



The shutdown-timeout configuration that was provided by JEUS 6 is no longer available. Instead, the same function is provided if the shutdown timeout option is configured through the console tool (jeusadmin).

- **Handling web attacks**

Web engines provide settings that prevent attacks like denial of service (DoS).

The limits for the POST request size, the number of parameters included in GET and POST requests, the number of headers included in a request, the size of headers included in a request, and the query string size of a GET request can be configured. For more information about these settings, refer to [Preventing Web Attacks](#).

1.4. Management Tools

This section describes the tools that control and monitor web engines.

1.4.1. Types of Tools

The following are the web engine operating tools.

- **Console tool (jeusadmin)**

A tool for controlling web engines through the console. This tool provides basic control and monitoring functions. For more information about the console tool, refer to "Controlling and Monitoring JEUS Servers" in *JEUS Server Guide* and "Web Engine Commands" in *JEUS Reference Guide*.

1.4.2. Controlling and Monitoring

Tools can be used to control and monitor the web engine.

Controlling Web Engine

Controlling a web engine includes starting and shutting down the engine. This can be accomplished by using the console tool.

- **Using the console tool**

Web engines are part of JEUS servers and there is no way to control web engines independently from the servers. For more information, refer to "Controlling and Monitoring JEUS Servers" in *JEUS Server Guide*.

Monitoring Web Engine

The console tool can be used to monitor web engines.

```
show-web-statistics [-server <server-name>]
                   [-ctx,--context <context-name>]
                   [-r,--request | -s,--session | -t,--thread |
                   -m,--memory]
```

You can use `jeusadmin` to view basic information about web engines. For more information about `jeusadmin`, refer to "JEUS Server Guide" and "Web Engine Commands" in *JEUS Reference Guide*.

1.5. Directory Structure

The following is the directory structure of JEUS web engines.

```
{JEUS_HOME}
|--domains
    |--domains1
        |--config
            |--[X]Domain.xml
            |--Servlet
                |--server1
                    |--[X]web.xml
                    |--[X]webcommon.xml
                    |--[X]web.xml
                    |--[X]webcommon.xml
        |--servers
            |--server1
                |--logs
                    |--[T]jeusServer.log
                    |--servlet
                        |--vhost
                            |--[T]access.log
                            |--[T]access.log
```

* 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

JEUS installation home directory.

domains/*domain1*/config/

The `domains` directory in `JEUS_HOME` contains a directory for each domain name. The web engine

configuration files are in the config directory, and *domain1* is an example for a domain name. **From JEUS 7 onwards, WEBMain.xml is no longer used. There is a separate web engine for each server JVM, so a configuration directory for each engine is not needed.**

Directory / File	Description
domain.xml	Integrated configuration file that contains all server configurations for the domain. Web engines can also be configured by each server. For more information about domain.xml, refer to "JEUS Server Guide".
servlet/	<p>This directory contains the XML configuration file that is used by all servers in the domain when deploying web applications. When servers in the domain start, they receive web.xml and webcommon.xml from the Domain Administration Server to maintain synchronization.</p> <ul style="list-style-type: none"> ◦ web.xml: For Servlet 3.0 and later, web.xml is not needed. But because Tomcat Jasper, a JSP parser, checks the application version based on the file, the file must be kept. Web applications that do not contain web.xml will also use this file. ◦ webcommon.xml: Shared configuration file that applies to all web applications. The file type is same as web.xml.
servlet/server 1/	<p>If webcommon.xml and web.xml exist in the servlet configuration directory, the files only apply to that server. This configuration has a higher precedence than those in the parent directories.</p> <p>If web.xml or webcommon.xml exists in a server directory, they are not managed automatically by JEUS, so the server administrator must manage the files manually.</p>

domains/domain1/servers/server1/logs

All files created while each server in the domain is running are located in the directory created with the server's name, under the **servers** directory. For more information, refer to "Logging" in JEUS Server Guide.

Log files are created in the **logs** directory. Logs created by web engines are stored in JeusServer.log. The file name can change due to log rotation, but it is initially set to JeusServer.

Directory / File	Description
servlet/	<p>This directory contains the web application access log, access.log. If access logs are created by each virtual host in this directory, directories will be created with the virtual host names and the access logs will be stored in the corresponding directory. For more information, refer to Virtual Hosts.</p> <p>This directory also contains the user.log or user_appname.log file that contains logs created through the log method of the ServletContext API. User logs can be stored in user.log or in a separate file, user_appname.log, depending on the configuration.</p>

1.6. Environment Configurations

This section describes environment and web engine configuration files.

Settings for the web engine can be configured using domain.xml.

- **Web Engine Configuration**

You can configure the web engine in domain.xml.

```
<web-engine>
  ...
  <jsp-engine>...</jsp-engine>
  <web-connections>
    <http-listener>
      <name>SERVER-HTTP</name>
      <thread-pool>
        <min>10</min>
        <max>20</max>
        <max-idle-time>300000</max-idle-time>
        <max-queue>-1</max-queue>
      </thread-pool>
      <postdata-read-timeout>600000</postdata-read-timeout>
      <max-post-size>-1</max-post-size>
      <max-parameter-count>-1</max-parameter-count>
      <max-header-count>-1</max-header-count>
      <max-header-size>8192</max-header-size>
      <max-querystring-size>8192</max-querystring-size>
      <server-access-control>>false</server-access-control>
      <allowed-server>...</allowed-server>
      <server-listener-ref>http-server</server-listener-ref>
    </http-listener>
  </web-connections>
  <monitoring>...</monitoring>
  <access-log>...</access-log>
  <ejb-engine>...</ejb-engine>
  <jms-engine>...</jms-engine>
  <system-logging>...</system-logging>
  ...
</web-engine>
```

1.6.1. XML Configuration Files

The following XML configuration files are related to web engines.

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

Location	JEUS_HOME/domains/<domain-name>/config
Purpose	JEUS web engine configurations
Details	"JEUS Server Guide"

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

Location	<packaged web application>/WEB-INF/
Purpose	JEUS web module deployment descriptor (DD)
Details	Described in this guide

- **web.xml** (web-app_5_0.xsd)

Location	<packaged web application>/WEB-INF/
Purpose	Jakarta EE standard web module DD
Details	Servlet 5.0 specification

- **web.xml** (web-app_5.0.xsd)

Location	JEUS_HOME/domains/<domain-name>/config/servlet
Purpose	Web.xml for web modules without a separate copy of web.xml
Details	Servlet 5.0 specification

- **webcommon.xml** (web-app_5_0.xsd)

Location	JEUS_HOME/domains/<domain-name>/config/servlet
Purpose	Shared configuration file that applies to all web modules in the web engine.
Details	Servlet 5.0 specification

XML schema files are located in the JEUS_HOME/lib/schemas/jeus/supportLocale/ko/ directory. These files must start with the XML header defined by JEUS. The root element must specify the namespace of the JEUS XML schema with the existing namespace.

The following are the headers used by each file. The XML header in web.xml is included in the servlet specification.

- XML header in domain.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<domain xmlns="http://www.tmaxsoft.com/xml/ns/jeus" version="9">
```

- XML header in jeus-web-dd.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<jeus-web-dd xmlns="http://www.tmaxsoft.com/xml/ns/jeus" version="9">
```

- XML header in web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="5.0" xmlns="https://jakarta.ee/xml/ns/jakartaee">
```



All tags used in this guide are written in the same order as the XML schema. Since it is not easy to maintain the order, JEUS provides a sorting function that automatically orders the tags. Users do not need to maintain the order when editing an XML configuration file. For more information about the configuration order, refer to "JEUS XML Reference".

In the previous examples, the standard header is omitted for readability. Actual XML configuration files contain the standard header.

1.6.2. Monitoring

Web engines can monitor the changes of threads, classes, and sessions. Web engines can also change the monitoring interval.

The following shows how to configure the monitoring interval using domain.xml.

```
<web-engine>
  ...
  <monitoring>
    <check-thread-pool>300000</check-thread-pool>
    <check-class-reload>300000</check-class-reload>
    <check-session>300000</check-session>
  </monitoring>
  ...
</web-engine>
```

The following describes each option.

Item	Description
Check Thread Pool	Interval for checking worker thread states in the worker thread pool. The default value is 300000 (5 minutes).
Check Class Reload	Interval for checking if each class has been modified for automatic web context reloading. This configuration is used when <auto-reload><enable-reload> in jeus-web-dd.xml is configured. This does not check web applications for which <check-on-demand> is set to true.
Check Session	Interval for checking the timeout state of each session. The timeout states of sessions are defined in the web.xml file of the web engine or context. The default value is 300000 (5 minutes).

Since monitoring configurations except the Min/Max settings for the thread pool cannot be applied dynamically, you need to restart the server to apply the changes.

1.6.3. Default Error Page

When a web context exists for a request, the web context handles any errors that occur. However, if a web context does not exist, the web engine must handle the error by displaying an error page. In this case, since the default error page might not be in a form suitable for the user, the absolute path to the desired error page can be configured on the web engine.

This configuration is only used for HTTP responses. Only HTML or HTM files can be used, and forwarding is not allowed.

The following explains how to configure the default error page using domain.xml.

Specify the absolute path to the default error page in the '**Default Error Page**' field.

```
<web-engine>
  <default-error-page>/jeus/user/error.html</default-error-page>
</web-engine>
```

Since the default error page setting cannot be applied dynamically, you need to restart the server to apply the setting.

1.6.4. Stack Trace Attachment

This configuration determines whether to display details about the error on the default error page provided by JEUS if an error occurs while a web engine is processing a request. The stack trace details are useful in a development environment, but it is not recommended to display them to the user in the production environment. The configuration can be set in jeus-web-dd.xml of each application. For more information about configuring jeus-web-dd.xml, refer to [jeus-web-dd.xml Configuration](#).

The following explains how to configure the option to attach the stack trace in case of an error using domain.xml.

The default value is false. When set to true, Stack Trace is displayed in the default error page. However, if the '**Default Error Page**' is specified, the error page is printed as configured. For more information about the default error page configuration, refer to [Default Error Page](#).

```
<web-engine>
  ...
  <attach-stacktrace-on-error>false</attach-stacktrace-on-error>
  ...
</web-engine>
```

Since the stack trace attachment cannot be applied dynamically, you need to restart the server to

apply the setting.

1.6.5. Encoding

There are three types of encoding configurations that can be used by all contexts in the engine. They can be applied to each application, virtual host, or server by using `jeus-web-dd.xml`. The XML configurations are applied in the aforementioned order. The administrator can set the encoding in `domain.xml`. **Since encoding configuration can vary depending on each web application, it is recommended to set it in `jeus-web-dd.xml`.** (The following XML is a configuration for `jeus-web-dd.xml` or `domain.xml`.)

• Request Encoding

The encoding that is applied to query strings and cookies in the HTTP request header, and the HTTP body.

- Query strings in HTTP request lines are encoded in the following order. Query strings described to process forward and include in a Servlet or JSP are also encoded in the same order.
 1. The "forced" encoding of `<request-url-encoding>` set in xml.
 2. The "forced" encoding of `<request-encoding>` set in xml.
 3. The "client-override" encoding of `<request-encoding>` set in xml.
 4. The "default" encoding of `<request-url-encoding>` set in xml.
 5. The "encoding" encoding of `<url-mapping>` in `<request-encoding>` set in xml
 6. The "default" encoding of `<request-encoding>` set in xml.
 7. ISO-8859-1
- Cookies in HTTP request headers are encoded in the following order of priority.
 1. The "charset-encoding" value of `<cookie-policy><write-value-on-header-policy>` that is set in xml.
 2. The "forced" encoding of `<request-encoding>` set in xml.
 3. The "client-override" encoding of `<request-encoding>` set in xml.
 4. The "encoding" encoding of `<url-mapping>` in `<request-encoding>` set in xml
 5. The "default" encoding of `<request-encoding>` set in xml.
 6. ISO-8859-1

The same precedence order is applied to the cookies of the HTTP Response header. To consistently configure cookie encoding, it is recommended to set the same value to "charset-encoding" described in 1. for all web engines.

- HTTP Bodies are encoded in the following order of priority.
 1. The "forced" encoding of `<request-encoding>` set in xml.

2. The Servlet/JSP application configuration, which is set through `request.setCharacterEncoding()`.
3. The "client-override" encoding of `<request-encoding>` set in xml.
4. The encoding that is set by `charset` in `Content-Type` of the HTTP request.
5. The "encoding" encoding of `<url-mapping>` in `<request-encoding>` set in xml
6. The "default" encoding of `<request-encoding>` set in xml.
7. The encoding of `<request-character-encoding>` set in `web.xml`.
8. ISO-8859-1

If a request is POST and `Content-Type` is `application/x-www-form-urlencoded`, a web engine will read the body and process parameters when a web application calls `ServletRequest.getParameter()`.

Note that since `ServletRequest.getParameter()` also processes URI query strings, if encoding set in `<request-url-encoding><forced>` is different from that in `<request-encoding>`, String objects to which different encodings are applied can be returned.

Bodies are encoded when `ServletRequest.getReader()` is called.

The request encoding precedence order is different from JEUS 6 and 7 (earlier than Fix#1). Hence, the encoding configuration must be modified according to the previous order or compatibility options must be applied in applications where HTTP request encoding was processed in existing version.



The "forced" option has a higher precedence than the ServletAPI starting from JEUS 7 Fix #2. The client-override configuration must be used to keep the "forced" configuration in Fix #1 or earlier.

• Response Encoding

The encoding that is applied to a body of HTTP response messages.

Response encoding decides which encoding to use for web container responses when converting `ServletOutputStream.println()` or `PrintWriter.println()` to a byte array, setting the "XXX" value of `"Content-Type:text/html;charset=XXX"` in the HTTP header, etc. It can also be set in `jeus-web-dd.xml`.

- The response encoding is applied in the following order of priority.
 1. The "forced" encoding of `<response-encoding>` set in xml.
 2. The encoding by the JSP Response Character Encoding API call.
 3. The "default" encoding of `<response-encoding>` set in xml.
 4. The encoding of `<response-character-encoding>` set in `web.xml`
 5. ISO-8859-1

JSP page encoding is for JSP files. However, `<response-encoding><forced>` is used for JSP file encoding because JSP page encoding and JSP response encoding are rarely used differently. If BOM exists in a JSP file, UTF-8 is used.



`<response-encoding><forced>` replaces JSP page encoding to help JSP developers to correct an invalid `pageEncoding` value described in a page tag only by configuring JEUS. For example, when EUC-KR is set for JSP files and `pageEncoding` is set to UTF-8, this issue can be resolved by setting `<response-encoding><forced>` to EUC-KR.

To put it simply, the following precedence order is used to read a JSP file.

BOM > `<response-encoding><forced>` > JSP Page Character Encoding > JSP Response Character Encoding > default



For detailed information about the JSP Page Character Encoding and Response Character Encoding, refer to the JSP standard.

Using domain.xml

The encoding can be set in domain.xml using the following ways.

In the **<Encoding>** section, enter the settings values for the '**<request-encoding>**', '**<request-url-encoding>**', and '**<response-encoding>**' tags. Except for '**<request-url-encoding>**', the configuration can also be set in jeus-web-dd.xml. For each item, select 'Default', 'Forced', or 'Client Override', and enter the encoding name.

```
<web-engine>
...
<encoding>
  <request-encoding>
    <client-override>...</client-override>
  </request-encoding>
  <request-url-encoding>
    <default>...</default>
  </request-url-encoding>
  <response-encoding>
    <default>...</default>
  </response-encoding>
</encoding>
...
</web-engine>
```

Value	Description
Default	Specifies the default encoding to use if no encoding is specified.

Value	Description
Client Override	Indicates to use the charset of the Content-Type header sent by the client to use if no encoding is specified.
Forced	Indicates to use a specific encoding at all times.

Character Encoding Configuration and Compatibility Guide

JEUS provides encoding configuration in XML to provide application developers with the ease of development and application users or administrators with ease of management. However, in the process of determining and implementing the encoding policy, there arose problems such as the mismatch of "forced" configuration definition between the request and response encodings or the violation of the servlet standard.



Starting from JEUS 7, efforts have been made to improve the consistency and correctness of the encoding policy. This may cause compatibility issues with existing applications. This guide will help modify applications or to use the compatibility option to maintain operation. It is recommended to configure the compatibility option in jeus-web-dd.xml. If it is configured in domain.xml of each server, it affects all applications deployed to the server.

• Request Url Encoding

- Starting from JEUS 7, the Accept-Language Header of an HTTP Request is not referenced, and no compatibility option is provided for this.
- The forced encoding of <request-url-encoding> is used only to process query strings included in a request URL as parameters.
- When a Servlet calls ServletRequest.getParameter() for POST requests (Content-Type: application/x-www-form-urlencoded), it must be determined whether to use the forced encoding of <request-url-encoding> according to whether a query string will be processed after included in an HTTP body or will be recognized as an HTTP header.

To process a query string as an HTTP body, set the forced encoding. To process a query string as an HTTP header, do not set the forced encoding. In the case of GET requests, the forced encoding of <request-url-encoding> must be set because only query strings that are included in a request URL are processed as parameters.

- Query strings become parameters when a Servlet first calls ServletRequest's parameter API.

• Request Encoding

- Applied to query strings, cookies, and request bodies.
- If forced is configured for <request-url-encoding><forced>, query strings are not affected.
- For cookies, top precedence is given to charset-encoding configuration in <cookie-policy><write-value-on-header-policy>. The jeus.servlet.request.cookie.encoding and jeus.servlet.urldecode.cookie settings used in JEUS 6 and earlier versions are no longer

supported.

- If `<request-encoding><forced>` is used in a version earlier than JEUS 7 Fix#2, it is replaced by `<request-encoding><client-override>`.
- The problem with `request.setCharacterEncoding()` in JEUS 6 is that it applies encoding to the HTTP body as well as query strings and cookies. But the servlet standard limits the API to be applied to only the HTTP body. This is no longer an issue in JEUS 7. To use the API in JEUS 6, the `jeus.servlet.request.6CompatibleSetCharacterEncoding` property must be set to true, and it is recommended to set it in `jeus-web-dd.xml` for the relevant application.
- **It is not recommended to configure `<request-encoding><forced>`.**

Web application developers do not need to call `request.setCharacterEncoding()` whenever writing a program if `client-override` of default is set. String objects can be directly created by passing an encoding value to `new String()` as in the following example.

```
String param = request.getParameter();
String realParam = new String(param.getBytes("ISO-8859-1"), "EUC-KR");
```

In the above example, to create the `param` object with ISO-8859-1, the default encoding (ISO-8859-1) must be set in a web engine. The `realParam` String object is created by a web application through JVM without a web engine.

- **From JEUS 7 Fix#4 onwards, `<request-encoding><url-mapping>` is added in `jeus-web-dd.xml`.**

`<request-encoding>` in `jeus-web-dd.xml` replaces `<request-encoding>` in `domain.xml`.

- `<request-encoding>` in `jeus-web-dd.xml` replaces `<request-encoding>` in `domain.xml`.
- If only `<url-mapping>` is set, only requests specified in `<servlet-path>` are encoded. Requests that are not specified in `<servlet-path>` are encoded by ISO-8859-1, the Servlet standard.
- If `<url-mapping>` and `<default>` are set together, requests that are not specified in `<servlet-path>` are encoded by using the setting in `<default>`.
- Setting in `<request-url-encoding><forced>` of `domain.xml` is only applied to POST bodies.
- `<request-encoding><url-mapping>` has lower priority than the charset of a Content-Type header sent by a client.

Request Encoding Configuration in `<jeus-web-dd.xml>`

```
<?xml version="1.0" encoding="UTF-8"?>
<jeus-web-dd>
  <encoding>
    <request-encoding>
      <url-mapping>
        <servlet-path>/test/test1</servlet-path>
        <encoding>EUC-KR</encoding>
      </url-mapping>
      <url-mapping>
        <servlet-path>/test/*</servlet-path>
```

```

        <encoding>UTF-8</encoding>
    </url-mapping>
    <url-mapping>
        <servlet-path>*.jsp</servlet-path>
        <encoding>EUC-KR</encoding>
    </url-mapping>
</request-encoding>
</encoding>
</jeus-web-dd>

```

• Response Encoding

- Applied to the response bodies.
- Cookies are not affected by the response encoding. `charset-encoding` setting in `<cookie-policy><write-value-on-header-policy>` is a top priority.
- The `<response-encoding><forced>` option has a higher precedence than `response.setCharacterEncoding()`, `setContentType()`, and `setLocale()`. However, the precedence in applying the "forced" option is not defined in JEUS 6 up to JEUS 7 Fix#1. This resulted in incorrect implementations, and in order to continue to use the applications that use these operations, the `jeus.servlet.response.6CompatibleForcedEncoding` property must be set to true. It is recommended to set the property in `jeus-web-dd.xml` for a corresponding application.
- Until JEUS 6, if "forced" is configured, `response.setCharacterEncoding()` has the top precedence. If a user still wants to maintain this operation, the `jeus.servlet.response.6CompatibleSetCharacterEncoding` property must be set to true. It is recommended to set the property in `jeus-web-dd.xml` for a corresponding application.
- **It is not recommended to configure `<response-encoding><forced>`.**

Web application developers can avoid having to configure encoding every time by using the default option. If the default configuration fails to work, the response encoding must be configured using the response API.

• JSP

- According to the JSP standard, JSP has two kinds of character encoding, the page character encoding and the response character encoding.
- The page character encoding is used for reading JSP files from the file system. This can be clearly distinguished from the response character encoding. Even though the file's encoding is UTF-8, the HTTP response created from executing the JSP file can be sent as EUC-KR. However, the response character encoding must be referenced if the page character encoding is unknown. If the response character encoding is also unknown, read the file using ISO-8859-1. Refer to the JSP standard for the precedence of the page character encoding. In most cases, it is preferable to clearly set it for each JSP file as in the following.

Page Character Encoding Configuration in `<sample.jsp>`

```
<%@ page pageEncoding="UTF-8"%>
```

- The response character encoding is the charset value written in the contentType property of <page>. If this value does not exist, the page character encoding is used. If the page encoding is also unknown, there is no default value specified in the JSP standard and it is determined by the configuration or operation of the web container.

Response Character Encoding in <sample2.jsp>

```
<%@ page contentType="text/html; charset=UTF-8"%>
```

- It is recommended to set both of the previous values. If a user wants to apply then in a batch, add the following configurations to web.xml.

Page and Response Character Encoding Configurations in <web.xml>

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app 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/web-app_5_0.xsd"
  metadata-complete="false"
  version="5.0">
  <jsp-config>
    <jsp-property-group>
      <url-pattern>*.jsp</url-pattern>
      <page-encoding>UTF-8</page-encoding>
      <default-content-type>text/html; UTF-8</default-content-type>
    </jsp-property-group>
  </jsp-config>
</web-app>
```

- The <response-encoding> is configuration for HTTP response and is clearly separate from the JSP page character encoding configuration. But, it is very rare for a developer to create a JSP file with a full understanding of all this. Thus, **<response-encoding><forced>** has precedence over the **JSP page character encoding** and **Response character encoding**.

1.6.6. JSP Engines

Since web engines contain JSP engines, JSPs must be configured by each web engine or each application. For more information, refer to [JSP Engines](#).

1.6.7. Response Headers

A user-custom HTTP response header can be defined by using name-value pairs.

The following example shows how to set the default user-custom header to include in a response using domain.xml.

```
<web-engine>
```

```
...
```

```

<response-header>
  <custom-header>
    <header-field>
      <field-name>HeaderName1</field-name>
      <field-value>HeaderValue1</field-value>
    </header-field>
  </custom-header>
</response-header>
...
</web-engine>

```

Item	Description
Custom Header	<p>Defines the custom field that will be added to HTTP responses.</p> <ul style="list-style-type: none"> Header Field: Specify the user-custom header. When setting multiple response headers, associate the name and value of each header with '=', and separate each header by a new line.

1. If set successfully, a result message "Saved" is displayed. Since the response header setting cannot be applied dynamically, you need to restart the server to apply the setting.

1.6.8. Cookie Policy

A policy can be configured for reading a cookie from an HTTP request header or sending a cookie that was created by the application as an HTTP response. The cookie policy can be configured in domain.xml or in jeus-web-dd.xml for each application.

The following example shows how to set the cookie policy in domain.xml. This example sets the cookie encoding to UTF-8 as the cookie policy.

```

<web-engine>
  <cookie-policy>
    <write-value-on-header-policy>
      <apply-url-encoding-rule>true</apply-url-encoding-rule>
      <charset-encoding>UTF-8</charset-encoding>
    </write-value-on-header-policy>
  </cookie-policy>
</web-engine>

```

Item	Description
Apply Url Encoding Rule	Option to apply the URL encoding rule.
Charset Encoding	Charset encoding that is used when cookies are used for a response or interpretation regardless of whether the URL encoding rule will be applied. If not set, the request encoding will be used.

Since the cookie policy setting cannot be applied dynamically, you need to restart the server to apply the setting.

1.6.9. Sessions

The web engine session settings configure items such as session object sharing, session cookie configurations, and timeouts that are required to manage sessions.

- Web engine level

You can configure sessions using the console tool. For more information about session configuration, refer to "Session Configuration" in *JEUS Session Management Guide*.

- Context level

Sessions are configured in the <jeus-web-dd> element of jeus-web-dd.xml.

If sessions are configured at the web engine level, the configurations are shared within the web engine even if they are not configured at the context level. If sessions are configured both at the web engine level and the context level, the context level configurations have a higher precedence. The precedence of the configuration values is **context level > web engine level**. If an item is not configured at the context level, the web engine configuration will be used. If neither level contains the configuration, the default value of the web engine will be used.



For more information about sessions in a clustered environment, refer to "Session Tracking" in *JEUS Session Management Guide*.

1.6.10. Logs

This section describes how to configure logs.

- [Configuring Server Logs](#)
- [Configuring Access Logs](#)
- [Configuring Access Logs by Virtual Host](#)
- [Formatting Access Logs](#)
- [Filtering Access Logs by Writing the Code](#)
- [Configuring User Logs](#)

Configuring Server Logs

For more information about server logs, refer to "Logging" in *JEUS Server Guide*.

Configuring Access Logs

Web engine access logs can be configured through the console. Access logs are written to a file by default.



From JEUS 6 Fix #8 onwards, the access logs are set to use log rotation by default.

Log rotation saves the log file with a new name when the file exceeds a specified size or at a specified time, and records new logs in the existing file.

The following shows how to configure access logs in domain.xml.

XML Configuration for Access Logs: <domain.xml>

```
<domain xmlns="...">
  ...
  <servers>
    <server>
      <web-engine>
        <access-log>
          <level>INFO</level>
          <use-parent-handlers>>false</use-parent-handlers>
          <formatter-pattern>[%d{yyyy.MM.dd HH:mm:ss}][%l] [%J-%T] [%M-%N] %m</formatter-
pattern>
          <handler>
            <file-handler>
              <name>accessLogFileHandler</name>
              <level>FINEST</level>
              <enable-rotation>>true</enable-rotation>
              <valid-day>1</valid-day>
              <buffer-size>1024</buffer-size>
              <append>>true</append>
            </file-handler>
          </handler>
          <enable>true</enable>
          <format>default</format>
          <enable-host-name-lookup>>false</enable-host-name-lookup>
        </access-log>
      </web-engine>
    </server>
  </servers>
  ...
</domain>
```

The default settings can be used without any specific configurations. Custom configurations may be necessary, especially when you need to change the access log format or exclude specific file extensions from the log. To use additional handlers other than the default handlers, add them in the **<handler>**. For more information about handlers, refer to "Logging" in *JEUS Server Guide*.

The following example changes the access log format from the default format to a user custom format.

```

<access-log>
  <level>INFO</level>
  <use-parent-handlers>>false</use-parent-handlers>
  <formatter-pattern>[%d{yyyy.MM.dd HH:mm:ss}][%l] [%J-%T] [%M-%N] %m</formatter-pattern>
  ...
  <exclude-ext>ipg,txt</excluded-ext>
  <enable-host-name-lookup>>false</enable-host-name-lookup>
</access-log>

```

The following describes each item of **Advanced options**.

Item	Description
Enable Host Name Lookup	Option to log the host name instead of the IP address when the %h format is used. Note that setting this option to true may cause overhead due to DNS lookup.
Exclude Ext	File extensions to be excluded from access logs. Multiple values can be specified with comma separators.
Filter Class	Filter class of the logger.
Use Parent Handlers	Option to use parent logger handlers. The default value is true, which prints log messages by using parent logger handlers.

In web engine access log configurations, the filter class and formatter pattern settings cannot be applied dynamically. Therefore, you need to restart the server to apply the setting.

Configuring Access Logs by Virtual Host

To create access logs by virtual host, the access logs must be configured in each virtual host. For more information, refer to [Configuring Virtual Hosts](#).

Formatting Access Logs

The access log format can be configured in domain.xml.

```

<access-log>
  <level>INFO</level>
  <use-parent-handlers>>false</use-parent-handlers>
  <formatter-pattern>[%d{yyyy.MM.dd HH:mm:ss}][%l] [%J-%T] [%M-%N] %m</formatter-pattern>
  ...
</access-log>

```

The formats primarily use '%' to express data, but other character strings can also be utilized.



1. JEUS 9 complies with Common Log Format (CLF). For more information about CLF, refer to <http://httpd.apache.org/docs/2.4/logs.html>.
2. For more information about changing the format using the console tool, refer

to "modify-web-engine-configuration" in *JEUS Reference Guide*.

Web engines provide the following format aliases.

- default

Default format provided by JEUS. This format adds the processing time (%D) to the common format that is the most commonly used in CLF. Due to performance reasons, "%l", which always generates "-", and "%I", which reads values from sessions, have been deleted.

```
%h %t \"%r\" %s %b %z %D
```

- common

Most commonly used format in CLF. It is an alias for the following format.

```
%h %l %u %t \"%r\" %s %b
```

- combined

Prints the Referrer and User-agent from the HTTP header. It is an alias for the following format.

```
%h %l %u %t \"%r\" %s %b \"%{Referer}i\" \"%{User-agent}i\"
```

- debug

Adds the session ID (%S) and response processing thread name (%I) to the default format.

```
%h %t \"%r\" %s %b %z %D %u %S \"%I\"
```

The format can be set with an alias from the previous list.

```
default %S
```

In this case, the common alias is replaced by the following format and is registered to the web engine.

```
%h %l %u %t \"%r\" %s %b %z %D %S
```

Format setting can be modified and applied while a service is running. Hence, if the currently running service has a problem, set the debug format to record the session ID and response processing thread name.

Filtering Access Logs by Writing the Code

Access log filtering is provided to record only logs that meet specific conditions. Requests can be filtered by file extension using `<exclude-ext>`.



Starting from JEUS 7, the package has been changed from `jeus.servlet.util` to `jeus.servlet.logger`.

JEUS provides the `jeus.servlet.logger.AccessLoggerFilter` interface and `jeus.servlet.logger.AbstractAccessLoggerFilter` abstract class.

Definition of `jeus.servlet.logger.AccessLoggerFilter` Interface `<AccessLoggerFilter.java>`

```
package jeus.servlet.logger;

import java.util.logging.Filter;
import java.util.logging.LogRecord;

/**
 * This interface provides the information required to filter access logger.
 * This interface inherits {@link Filter}. It can define various filtering policies in
 * {@link Filter#isLoggable(java.util.logging.LogRecord)} using additionally provided interfaces.
 */
public interface AccessLoggerFilter extends Filter {
    /**
     * Returns the address of the remote client that accesses the server.
     *
     * @param record a LogRecord
     * @return remote client's address. Returns null if the address is invalid.
     */
    public String getRemoteAddr( LogRecord record );

    /**
     * Returns the HTTP method of the request. ex) GET, POST, PUT, etc...
     *
     * @param record a LogRecord
     * @return request method. Returns null if the return value is invalid.
     */
    public String getMethod( LogRecord record );

    /**
     * Returns the URI of the request.
     *
     * @param record a LogRecord
     * @return request uri. Returns null if the return value is invalid.
     */
    public String getRequestURI( LogRecord record );

    /**
     * Returns the response status, ex) 200, 404
     *
     * @param record a LogRecord
     * @return status.
     */
    public int getStatus( LogRecord record );
}
```

```

/**
 * Returns the processing duration in milliseconds.
 *
 * @param record a LogRecord
 * @return processing time. Returns -1 if the return value is invalid.
 */
public long getProcessingTimeMillis( LogRecord record );
}

```

To use filters for certain patterns in the access log, the `AccessLoggerFilter` interface and `AbstractAccessLoggerFilter` abstract class can be used to easily implement and apply the filters.

User-created filter classes inherit `jeus.servlet.logger.AbstractAccessLoggerFilter` interface and must implement the `isLoggable()` method of the `java.util.logging.Filter` interface. When implementing the `isLoggable()` method, use the methods from the `jeus.servlet.logger.AccessLoggerFilter` API that are needed by the user.

The following example defines a filter class that disables leaving access logs if the request extension is `'.gif'`.

Example of Defining a Filter Class : `<SimpleAccessLoggerFilter>`

```

package sample;

import jeus.servlet.logger.AbstractAccessLoggerFilter;
import java.util.logging.*;

public class SimpleAccessLoggerFilter extends AbstractAccessLoggerFilter {
    public boolean isLoggable(LogRecord record) {
        String requestURI = getRequestURI(record);
        return requestURI != null && !requestURI.endsWith(".gif");
    }
}

```

- The `sample.SimpleAccessLoggerFilter` is a user-defined class that extends the `jeus.servlet.logger.AbstractAccessLoggerFilter` class.
- It implements the `java.util.logging.Filter#isLoggable()` method, and uses the `jeus.servlet.util.AccessLoggerFilter#getRequestURI()` method to get the client request URI.

Once the class is defined, compile the class. Add the file to the `'JEUS_HOME/domains/DOMAIN_HOME/lib/application'` directory as a JAR file, and configure the filter for the web engine access log. For more information about configuring access logs, refer to [Configuring Access Logs](#).

Configuring User Logs

By default, logs are stored in the server log file (`JeusServer.log`). To use a separate log file, configure a user log. For more information about the default user log file and its location, refer to [Directory Structure](#).

Like server logs, user logs can be registered at the JEUS server level as well as in `jeus-web-dd.xml` for

web applications. User logs can be created by each web application or in a specific log file.

The following explains how to configure user logs using domain.xml.

Configure it in the child tags of **<user-logging>**.

To create logs only for a web application named webapp, set '**<name>**' as in the following example. If 'jeus.systemuser' is entered without the name webapp, the user logs are configured for all web applications. For more information about each item in the Advanced Options section, refer to [Configuring Access Logs](#). Other items, such as '**<level>**', can be optionally set as needed.

You can add handlers in **<handler>**. Enter the tag of the handler to be added as the user log handler. For more information about configuring each handler, refer to "Logging" in JEUS Server Guide. In this example, the **<file-handler>** tag is added to add a file handler.

```
<server>
  ...
  <user-logging>
    <name>jeus.systemuser.webuser.app1</name>
    <level>INFO</level>
    <use-parent-handlers>true</use-parent-handlers>
    <filter-class>...</filter-class>
    <formatter-pattern></formatter-pattern>
    <handler>
      <file-handler>
        <name>fileHandler</name>
        <level>FINEST</level>
      </file-handler>
    </handler>
  </user-logging>
  ...
</server>
```

1.6.11. Async Servlet Timeout Processing

As of Servlet 3.0, the number of threads that is required to process an async servlet timeout must be configured.

The following shows how to set the async servlet timeout processing using domain.xml.

```
<web-engine>
  <async-timeout-min-threads>10</async-timeout-min-threads>
</web-engine>
```

Since the async servlet timeout processing setting cannot be applied dynamically, you need to restart the server to apply the setting.

1.6.12. Web Engine Level Properties

You can set properties defined in JEUS using domain.xml. For more information about web engine properties, refer to "Web Engine Properties" in *JEUS Reference Guide*.

The following shows how to set properties using domain.xml.

Set the '**<properties>**' tag. Enter the property name as <key> and the value as <value>.

```
<web-engine>
  <properties>
    <property>
      <key>jeus.servlet.request.enableDns=false</key>
      <value>>false</value>
    </property>
  </properties>
</web-engine>
```



It is recommended to configure properties through the '**<properties>**' tag. You can also configure properties using the '**<jvm-option>**' tag in domain.xml.

1.6.13. Preventing Web Attacks

Malicious users can attack JEUS by sending a large volume of requests with a large amount of data. When this happens, normal requests can be delayed or fail to receive a response.

To prevent these kinds of attacks, the JEUS administrator can set conditions to determine which requests are malicious and deny them. By default, this is configured in the web engine of the server but can also be set for each listener or connector that receives the request from the web engine. They can be configured in the HTTP listener, WebtoB connector, and AJP13. For more information about configuring each listener or connector, refer to [Common Listener Settings](#).

Properly configure the following items to prevent web attacks.

The following shows how to set the web attack prevention using domain.xml.

```
<domain xmlns="..." version="...">
  <servers>
    <server>
      <web-engine>
        <web-connections>
          <http-listener>
            <name>SERVER-HTTP</name>
            <thread-pool>
              <min>10</min>
              <max>20</max>
              <max-idle-time>300000</max-idle-time>
              <max-queue>-1</max-queue>
            </thread-pool>
          </http-listener>
        </web-connections>
      </web-engine>
    </server>
  </servers>
</domain>
```

```

        <postdata-read-timeout>600000</postdata-read-timeout>
        <max-post-size>-1</max-post-size>
        <max-parameter-count>-1</max-parameter-count>
        <max-header-count>-1</max-header-count>
        <max-header-size>8192</max-header-size>
        <max-querystring-size>8192</max-querystring-size>
        <server-access-control>>false</server-access-control>
        <allowed-server>...</allowed-server>
        <server-listener-ref>http-server</server-listener-ref>
    </http-listener>
</web-connections>
<web-engine>
    <server>
</servers>
</domain>

```

Configure each item in domain.xml. Each listener can use these settings to prevent web attacks. For more information about each configuration item, refer to [Common Listener Settings](#).

1.6.14. Blocking HTTP Requests with a Specific URL Pattern

For an example of blocking HTTP requests with a specific URL pattern, let's assume that the HTTP client sent the following request.

```
GET /examples/%2e%2e%2fdb.txt HTTP/1.1
```

The web container must decode the URL in the request URI first.

```
/examples/../db.txt
```

By doing this, the user can access files that are not related to the actual HTTP service. Typically, clients with a malicious intent send the request URI in this way.

Since the web container cannot detect all requests with a malicious intent, the user must directly provide settings to prevent malicious patterns. The settings are applied only to HTTP requests and if any match is found within the URI except for the query string, the request is handled as a '404 Not Found' unconditionally.



This setting was configured in the property file from JEUS 6 to JEUS 7 Fix#1, but starting from JEUS 7 Fix#2, it is configured in domain.xml.

The following shows configuration through domain.xml.

Configure the child tags of **<blocked-url-patterns>**.

```
<web-engine>
```

```
...
<blocked-url-patterns>
  <encoded-pattern>%00</encoded-pattern>
  <decoded-pattern>#</decoded-pattern>
  <deny-last-space-character>true</deny-last-space-character>
  <deny-null-character>true</deny-null-character>
</blocked-url-patterns>
...
</web-engine>
```

'**Encoded Pattern**' is used to check if the URI sent from the HTTP client contains the specified strings. The URI from the client is used after it is URL decoded by the web container. Here, it is checked whether the value specified in '**Decoded Pattern**' is included in the URL. If any '**Encoded Pattern**' is set, the basic pattern will not be processed. (Same for '**Decoded Pattern**')



If no configuration exists, the web container processes %2e, %2f, %5c, %23, and %00 as 404 in case of an Encoded Pattern and # and \ as 404 in case of a Decoded Pattern. Patterns are not case-sensitive. They are always converted to lower case letters before being processed.

1.7. Tuning the Web Engine

Consider the following items for the best web engine performance.

- Set <output-buffer-size> to an appropriate size. If using with WebtoB, set appropriate values for <send-buffer-size> and <receive-buffer-size> inside <webtob-connector><webtob-connector>.
- Set <check-included-jspfile> to 'false' unless the included JSPs have been modified. If it is set to false, the included JSP files will not be checked for modification to improve performance.
- If the JSP files have not been modified, the jeus.servlet.jsp.reload property is set to false. This prevents querying the metadata from the file system on every JSP call.

2. Web Connection Management

This chapter describes how to manage and set listeners and connectors provided by the web engine.

2.1. Overview

In JEUS, web connectors and web listeners are both referred to as web connections. The connections provided by a web engine includes connections to WebtoB and other web servers, direct connections to HTTP/TCP clients, and connections to Tmax. The web server receives an HTTP request from a client and delivers it to the corresponding web engine.

WebtoB and Apache are the typical web servers that are used. They provide the following connectors.

- **WebtoB connector**

WebtoB provides the WebtoB connector since JEUS acts as the client when a connection is created.

- **AJP listener**

Apache provides the AJP listener since JEUS acts as the server when a connection is created. Other commercial web servers such as IIS and SunOne(Iplanet) also support AJP, and therefore the AJP listener can be used to connect to other web servers.

The web engine provides the following listeners and connectors to directly create and manage connections for each client.

- **HTTP listener**

Directly manages connections to HTTP clients. **This listener must be used to use the functions of Async Servlet, Servlet NIO, and WebSocket.**

- **TCP listener**

Manages connections to TCP clients.

- **Tmax connector**

Manages connections to Tmax. When a connection is created, JEUS acts as a client, like the WebtoB connector.

Since listeners operate according to the JEUS server configurations, configure JEUS server and its listeners to use secure listeners (SSL).



For more information about configuring JEUS server, refer to "Listener Configuration" in *JEUS Server Guide*.

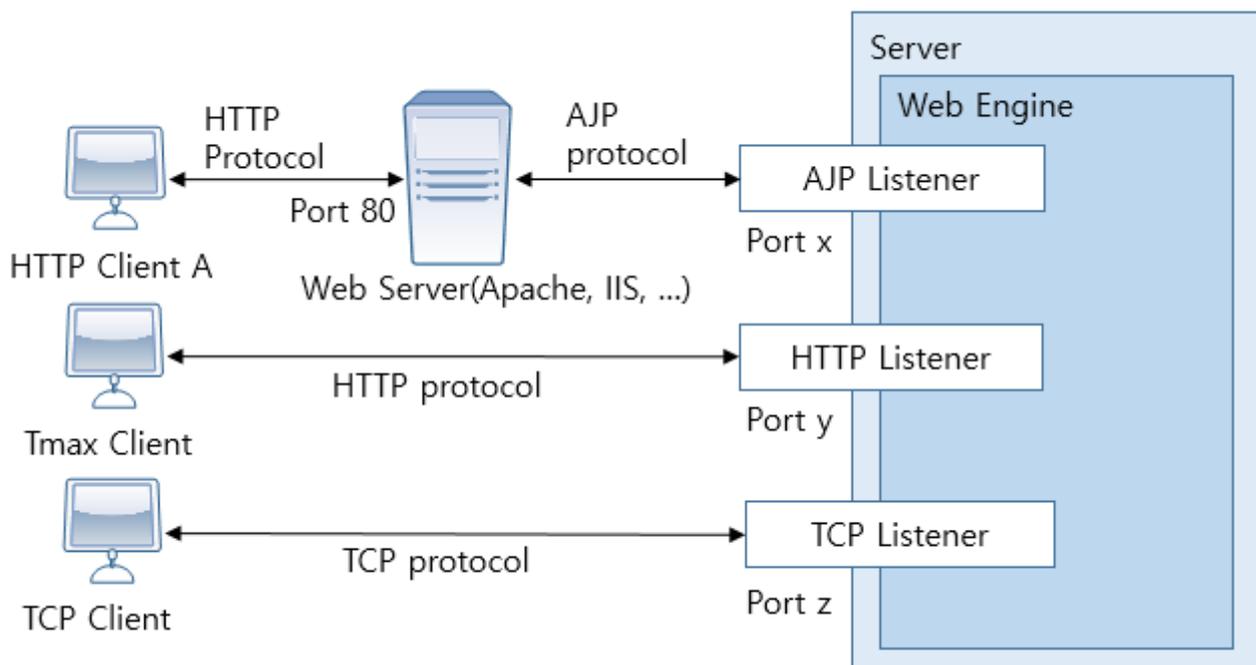
2.2. Components

This section describes listeners and connectors provided by the web engine.

2.2.1. Listeners

Listeners are web engine channels that can be accessed by HTTP/TCP clients and web servers that follow the AJP protocol.

The following figure show the connections between the web engine connectors, each clients, and protocols.



Web Engine Listeners

The following describes each listener.

- AJP listener

AJP listener allows web servers other than WebtoB, such as Apache, IIS, and SunOne(Iplanet), to interact with JEUS web applications. It is supported by the mod_jk module, uses the AJP1.3 protocol, and supports SSL. For more information about AJP listeners, refer to [AJP Listeners](#) and [Configuring Web Server Load Balancing](#). For more information about the configuration of SSL server listeners, refer to "JEUS Server Guide".

- HTTP listener

HTTP listener is used when the web engine directly receives HTTP requests. It supports SSL. For more information, refer to [HTTP Listeners](#).

For more information about configuring SSL for HTTP listeners, refer to "JEUS Server Guide".

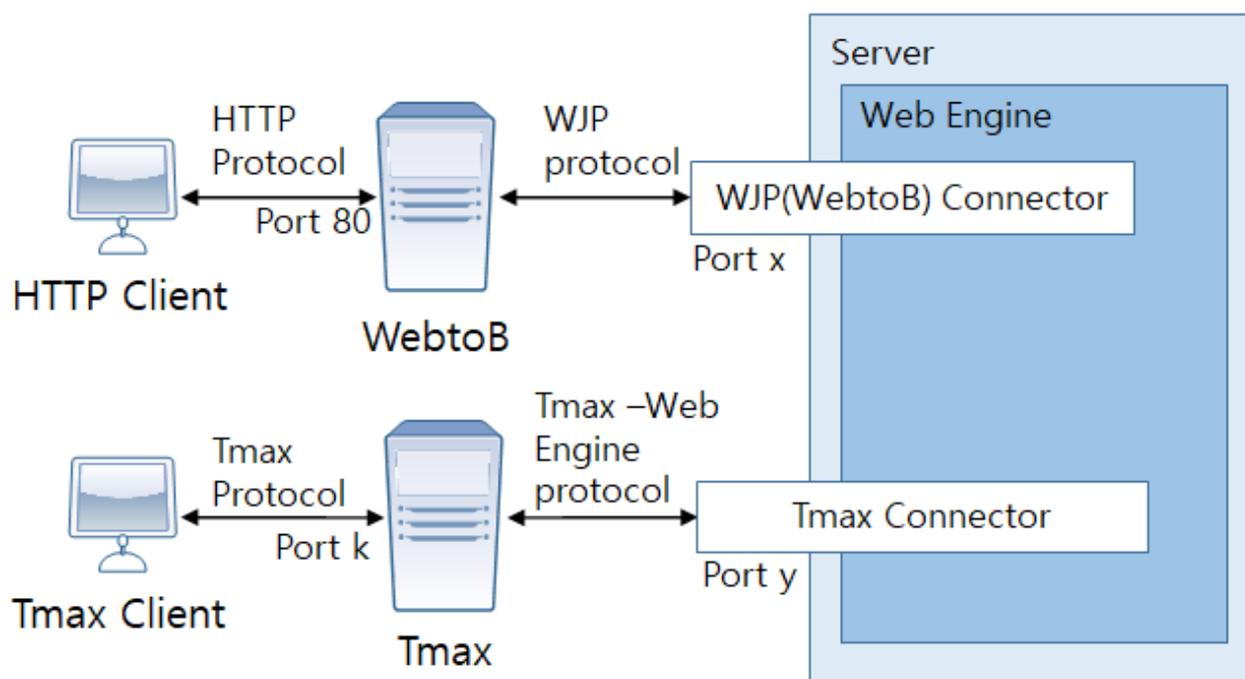
- TCP listener

TCP listener is used for clients that use a custom protocol instead of the HTTP protocol. For more information about TCP listeners, refer to [TCP Listeners](#) and [Using TCP Listeners](#).

2.2.2. Connectors

Connectors are channels that connect to WebtoB or Tmax from the web engine.

The following figure show the connections between the web engine connectors, each clients, and protocols.



Web Engine Connectors

The following describes each connector.

- WJP (WebtoB) connector

WJP stands for WebtoB-JEUS-Protocol. WebtoB is a web server provided by TmaxSoft. It can be used as a standalone installation as well.

WebtoB connector can directly connect to WebtoB since JEUS acts as the client when a connection is created. This feature allows connections to be made to WebtoB without configuring the firewall. For more information about the WebtoB connector, refer to [Configuring Web Server Load Balancing](#).

- Tmax connector

Tmax is a system software that manages XA transactions in a distributed environment. Like WebtoB connectors, JEUS acts as a client for Tmax connectors. The Tmax connector can be used to exchange information between JEUS and Tmax, or to receive HTTP requests through the Tmax gateway in order to integrate communication channels. For more information about the Tmax connector, refer to [Tmax Connectors](#).



JEUS only acts as a client when the connection is created. JEUS actually receives requests from external clients through WebtoB or Tmax, but it does not send requests to WebtoB or Tmax. JEUS functions as a server during service processing.

2.2.3. Worker Thread Pool

Listeners and connectors have a worker thread pool that is required to process client requests. The worker thread pool manages worker threads.

In versions prior to JEUS 21, listeners and connectors managed the worker thread pool. From JEUS 21 fix#1 onwards, web connections, virtual hosts, and contexts manage the worker thread pool. For more information, refer to [Thread Pool](#).

When a listener receives a request, a worker thread is allocated to process the request. When a WebtoB connector has HTTP, AJP13, and TCP listeners with `<use-nio>` being set to true, it looks up the contexts and hosts by parsing the request before passing the task to the worker thread, so it can redirect the request to the context or virtual host level thread pool. When a connector establishes a connection between WebtoB or Tmax and JEUS, JEUS acts as the client. If `<use-nio>` is set to false for a WebtoB or Tmax connector, requests cannot be read before assigning the worker thread pool because there is no server listener. In this case, only a connector-dependent thread pool can read and handle the requests, thus the number of connections must match the number of threads within the connector. Therefore, when configuring the thread pool for a web connection, ensure that the connection and thread pool have the same Min/Max values.

Since the Min/Max values for the worker thread pool have significant impact on the service processing performance, ensure that the worker thread pool is properly configured for a listener or connector.



Each thread pool manages its own status and the monitoring thread logs the result periodically. Hence, the changes in the log can be different from the actual changes of the thread pool.

Active-Management and Status Notification

The worker thread pool contains configurations for active management. The administrator can configure active management to allow the web engine to send a warning message via e-mail or to recommend restarting the server of the engine. To requires configuring the number of blocked worker threads that will trigger an action like sending a warning message via e-mail or a recommendation to restart the server.



If the recommendation to restart is displayed, the administrator must determine whether to restart the server because it is likely that the server cannot process requests normally.

2.3. Configuring Web Connections

Separate vendor-specific configurations are required to use an AJP listener, a WebtoB connector, or a Tmax connector. Since all web connections are managed by the engines, this section only describes the web engine configurations. For more information about configuring web servers, refer to [Configuring Web Server Load Balancing](#).



1. Starting from JEUS 6, context groups are no longer supported. Most items that were managed by the context group are now managed by the web engine.
2. A unified service listener provided by the server is used as a web listener, so AJP, HTTP, and TCP listeners are configured on the server and they are referenced for use. '**Server Listener Ref**' has been added to the configuration for this purpose. However, JEUS uses the current configuration for WebtoB and Tmax connectors because JEUS functions as a client when using these connectors.

You can add, modify or delete connectors using the console tool. For more information about using the console tool, refer to "Web Engine Commands" in *JEUS Reference Guide*.

2.3.1. Common Listener Settings

This section describes the common listener settings.

- **Name**

- Unique name that identifies the web connection. This is a required option and must be unique within the web engine.

- **Server Listener Ref**

- Represents the server listener that the listener references.
- HTTP and AJP listeners can reference the same server listener, but if the HTTP listener is configured for management, the HTTP listener cannot share the server listener. If not set, the default server listener will be used.
- TCP listener cannot reference a server listener that is referenced by another listener.
- 2 or more protocol listeners cannot reference the same server listener. For example, if two AJP listeners are configured, but neither listener is configured with a server listener, an error will occur because both reference the default server listener. One must refer to a different server listener.
- Idle client connections can be closed by using the server listener's `<keep-alive-timeout>` setting. (Unit: ms)

- **Connection Type**

- Forcibly sets the "Connection" header for responses that are sent through each listener or

connector.

- Set one of the following.

Value	Description
keep-alive	Keep the connection after sending the response.
close	Terminate the connection after sending the response.
none	Set the "Connection" response header based on the properties defined in the request header. If not set, the default action is same as 'none'.



The '**Connection Type**' cannot be configured for AJP listeners. It is recommended to follow the configurations of the web server, like Apache, for AJP listeners.

• Thread Pool

- Web connection-level worker thread pool configurations.
- The following describes each setting item.

Item	Description
Min	Minimum number of worker threads managed by the pool. If the <use-nio> setting is false for the WebtoB connector, this value must be the same as in <Connection-Count>.
Max	Maximum number of worker threads managed by the pool. If the <use-nio> setting is false for the WebtoB connector, this value must be the same as in <Connection-Count>.
Maximum Idle Time	Amount of time during which a thread remains unused before being removed from the pool. This causes increased resource usage.
Max Queue	Maximum number of requests that can wait in queue. If set to -1, there is no limit in the queue size.
Thread State Notify	Notification for a blocked worker thread. Each thread pool contains the ' Thread State Notify ' setting which defines the action to be taken when a failure occurs. For more information about this setting, refer to Configuring Automatic Thread Pool State Management .

• Output Buffer Size

- The size of the output buffer that is used by applications. If the buffer is full, its data will automatically be flushed by the web engine. (Unit: Byte)
- For AJP listeners, it is recommended that this value match the value of max_packet_size in the

workers.properties file of mod_jk. For more information about configuring mod_jk, refer to [Configuring Web Server Load Balancing](#).

- **Postdata Read Timeout**

- The maximum amount of waiting time allowed when reading request bodies. Applied in the `ServletInputStream.read()` method. (Unit: ms)
- The point in time when a timeout is applied is changed for HTTP and TCP listeners.

Since I/O control threads that exist in each listener read all request bodies, it is determined whether timeout occurred or not before the bodies are sent to a Servlet. If timeout occurs while reading request bodies, the 500 error is returned. If an HTTP request is chunked, the setting is applied when the servlet calls `ServletInputStream.read()`.

- **Max Post Size**

- A POST data request that is too large and too many resources are required to read and analyze them may prevent other requests from getting processed. This configuration is used to prevent this by blocking requests that are too large and burdensome for processing. (Default value: -1)
- Limits the maximum data size in bytes of POST requests based on the request Content-type.

- If Content-Type is x-www-form-urlencoded

Stops processing the request and send a "413 Request Entity Too Large" response if the request size or the total size of chunked data is greater than the specified value.

- If Content-Type is multipart/form-data

Limits the size of uploaded files and the total size of a POST request. To limit the size of uploaded files, it is recommended to use the multi-part configuration of Servlet.

- Servlet web application deployment descriptor affects this configuration in the following ways.

- `<multipart-config>` in web.xml.

If the `<multipart-config>` setting exists in web.xml, its child settings, `<max-file-size>` and `<max-request-size>`, are used. Otherwise, the web engine does not set a limit, so it is recommended to configure `<multipart-config>` for each application.

- `<content-length>`

If the `<content-length>` setting exists and the content length exceeds the specified value, the request will not be processed. If `<content-length>` is not set and the sum of the bytes of the name/value pairs of parameters exceeds the specified value, the request will not be processed.

- A negative value is considered as the default value, which means there is no limit on the data size.
- Applies to listeners and connectors that receive HTTP requests. It does not apply to TCP listeners or Tmax connectors.

- **Max Parameter Count**

- A request with too many parameters increases the resources required to analyze and manage the parameters. This configuration can be used to prevent such a problem. (Default value: -1)
- Limits the total number of parameters (name-value pairs) that can be included in GET and POST requests.
- If the number of parameters in a multipart/form request, a POST request, or query string of a GET request exceeds the specified value, a "413 Request Entity Too Large" response will be sent and the request will not be processed.
- A negative value is considered as the default value, which means there is no limit on the number of parameters.
- Applies to listeners and connectors that receive HTTP requests. It does not apply to TCP listeners or Tmax connectors.

- **Max Header Count**

- A request with too many headers increases the resources required to read the request. This problem can be avoided by blocking the request. (Default value: -1)
- If a request contains more headers than the specified value, a "400 Bad Request" response will be sent and the connection will be terminated.
- A negative value is considered as the default value, which means there is no limit on the number of headers.
- Applies to listeners and connectors that receive HTTP requests. It does not apply to TCP listeners or Tmax connectors.

- **Max Header Size**

- A request with too large headers increases the resources required to read the request. This problem can be avoided by blocking the request. (Default value: -1)
- If a request contains a header, which includes the header name, delimiter, and header value, with a total byte size that is greater than the specified value, a "400 Bad Request" response will be sent and the connection will be terminated.
- A negative value is considered as the default value, which means there is no limit on the header size.
- Applies to listeners and connectors that receive HTTP requests. It does not apply to TCP listeners or Tmax connectors.

- **Max Query String Size**

- A GET request with a long query string increases the resources required to read the request. This problem can be avoided by limiting the query string size. (Default value: 8192, Unit: bytes)
- If a request contains a query string whose size is greater than the specified number of bytes, a "400 Bad Request" will be sent and request will not be processed.
- If the value is negative, there is no limit on the query string size. Internally, JEUS limits the Request Line to 64 KB, so if the greater number is set, it will not be used.
- Applies to listeners and connectors that receive HTTP requests. It does not apply to TCP

listeners or Tmax connectors.

2.3.2. AJP Listeners

AJP listeners can be added, modified, or deleted using the console tool. AJP listeners comply with AJP 1.3.

Using the console tool

The following shows how to add, modify, and delete AJP listeners using the console tool.

- **Adding a listener**

To add an AJP listener using the console tool, execute the **add-ajp-listener** command. For more information about the command, refer to "add-ajp-listener" in *JEUS Reference Guide*.

```
add-ajp-listener [-cluster <cluster-name> | -server <server-name>]
                 [-f, --forceLock]
                 -name <web-connection-name>
                 -tmin <minimum-thread-num>
                 [-tmax <maximum-thread-num>]
                 [-tidle <max-idle-time>]
                 [-qs <max-queue-size>]
                 -slref <server-listener-ref-name>
```

- **Modifying a Listener**

To modify an AJP listener using the console tool, execute the **modify-web-listener** command. For more information about the command, refer to "modify-web-listener" in *JEUS Reference Guide*.

```
modify-web-listener [-cluster <cluster-name> | -server <server-name>]
                   [-f, --forceLock]
                   -name <web-connection-name>
                   [-tmin <minimum-thread-num>]
                   [-tmax <maximum-thread-num>]
                   [-tidle <max-idle-time>]
                   [-http2 <enable-http2>]
                   [-obuf <output-buffer-size>]
```

- **Deleting a Listener**

To delete an AJP listener using the console tool, execute the **remove-web-listener** command. For more information about the command, refer to "remove-web-listener" in *JEUS Reference Guide*.

```
remove-web-listener [-cluster <cluster-name> | -server <server-name>]
```

```
[-f, --forceLock]
<web-connection-name>
```

2.3.3. HTTP Listeners

HTTP listeners can be added, modified, and deleted using the console tool. Using HTTP listeners is only recommended for internal management.



JEUS 9 supports HTTP/2. For information about how to use HTTP/2, see [Using HTTP/2](#).

Using the console tool

You can use the console tool to add, modify, and delete an HTTP listener, following the same process as for an AJP listener. For HTTP listeners, however, an additional option to use HTTP/2 is provided. For more information about this, refer to [Using the console tool in AJP Listeners](#).

2.3.4. TCP Listeners

A TCP listener is a special listener that allows custom protocols to communicate with each other. TCP listener cannot share server listeners with other web listeners, so a dedicated listener for the TCP listener must be added to the server. TCP listeners can be added, modified or deleted using the console tool.

Using the console tool

You can use the console tool to add, modify, and delete a TCP listener, following the same process as for an AJP listener. For more information, refer to [Using the console tool in AJP Listeners](#).

2.3.5. WebtoB Connectors

Since JEUS acts as a client when a WebtoB connector creates a connection, the address and port of WebtoB are required. WebtoB connectors can be added, modified, and deleted in the console tool.



The version of the WJP, the communication protocol between WebtoB and JEUS, has been upgraded from 1 to 2 (hereafter WJPv1 and WJPv2). WJPv2 uses a smaller packet sizes than WJPv1 and provides various additional functions. If the WebtoB that JEUS will connect to does not provide WJPv2, WJPv1 is used instead. Since WebtoB cannot obtain the WJP version information, it must be configured in the <wjp-version> option of JEUS. WJP stands for WebtoB-JEUS Protocol.

Using the console tool

The following shows how to add, modify, and delete WebtoB connectors using a console tool.

- **Adding a listener**

To add a WebtoB connector using the console tool, execute the **add-webtob-connector** command. For more information about the command, refer to "add-webtob-connector" in *JEUS Reference Guide*.

```
add-webtob-connector [-cluster <cluster-name> | -server <server-name>]
                    [-f,--forceLock]
                    -name <web-connection-name>
                    -tmin <minimum-thread-num>
                    [-tmax <maximum-thread-num>]
                    [-tidle <max-idle-time>]
                    [-qs <max-queue-size>]
                    -conn <thread-number>
                    -regid <registration-id>
                    [-ver <wjp-version>]
                    [-useNio <use-nio>]
                    [-addr <WebtoB address>]
                    [-dsocket | -port <WebtoB port>]
                    [-wbhome <webtob-home>]
                    [-ipcport <ipc-base-port>]
                    [-sndbuf <send-buffer-size>]
                    [-rcvbuf <receive-buffer-size>]
                    [-hth <set-hth-count.>]
```

- **Modifying a Listener**

To modify the WebtoB connector using the console tool, execute the **modify-webtob-connector** command. For more information about the command, refer to "modify-webtob-connector" in *JEUS Reference Guide*.

```
modify-webtob-connector [-cluster <cluster-name> | -server
                        <server-name>]
                        [-f,--forceLock]
                        -name <web-connection-name>
                        [-tmin <minimum-thread-num>]
                        [-tmax <maximum-thread-num>]
                        [-tidle <max-idle-time>]
                        [-conn <thread-number>]
                        [-obuf <output-buffer-size>]
                        [-ver <wjp-version>]
                        [-addr <WebtoB address>]
                        [-dsocket | -port <WebtoB port>]
                        [-wbhome <webtob-home>]
                        [-cloud]
```

```
[-ipcport <ipc-base-port>]
[-regid <registration-id>]
[-sndbuf <send-buffer-size>]
[-rcvbuf <receive-buffer-size>]
[-hth <set-hth-count.>]
```

- **Deleting a Listener**

To delete a WebtoB connector using the console tool, execute the **remove-webtob-connector** command. For more information about the command, refer to "remove-webtob-connector" in *JEUS Reference Guide*.

```
remove-webtob-connector [-cluster <cluster-name> | -server
<server-name>]
[-f,--forceLock]
<web-connection-name>
```

2.3.6. Tmax Connectors

Just like with WebtoB connectors, JEUS operates as a client when connected through a Tmax connector, necessitating the specification of Tmax's address and port. Tmax connectors can be added, modified, or deleted using the console tool.

Using the console tool

The following shows how to add, modify, and delete a Tmax connector using the console tool.

- **Adding a listener**

To add a Tmax connector using the console tool, execute the **add-tmax-connector** command. For more information about the command, refer to "add-tmax-connector" in *JEUS Reference Guide*.

```
add-tmax-connector [-cluster <cluster-name> | -server <server-name>]
[-f,--forceLock]
-name <web-connection-name>
-tmin <minimum-thread-num>
[-tmax <maximum-thread-num>]
[-tidle <max-idle-time>]
[-qs <max-queue-size>]
-addr <server-address>
-port <server-port>
-svrg <server-group-name>
-svr <server-name>
-dcc <dispatcher-config-class>
```

- **Modifying a Listener**

To modify the Tmax connector using the console tool, execute the **modify-tmax-connector** command. For more information about the command, refer to "modify-tmax-connector" in *JEUS Reference Guide*

```
modify-tmax-connector [-cluster <cluster-name> | -server <server-name>]
                        [-f, --forceLock]
                        -name <web-connection-name>
                        -tmin <minimum-thread-num>
                        [-tmax <maximum-thread-num>]
                        [-tidle <max-idle-time>]
                        [-qs <max-queue-size>]
                        -addr <server-address>
                        -port <server-port>
                        -svrg <server-group-name>
                        -svr <server-name>
                        -dcc <dispatcher-config-class>
```

- **Deleting a Listener**

To delete a Tmax connector using the console tool, execute the **remove-tmax-connector** command. For more information about the command, refer to "remove-tmax-connector" in *JEUS Reference Guide*.

```
remove-tmax-connector [-cluster <cluster-name> | -server <server-name> |
                        -f, --forceLock]
                        <web-connection-name>
```

2.4. Configuring Web Server Load Balancing

This section describes how to configure web servers and how to connect web connections to the web engine.

Web servers and web engines (servlet engines) can be organized to improve the system performance when processing HTTP requests. First, web servers distribute the HTTP requests to the web engines. After that, they direct the same connections or session requests to the web engine that processed the first request by using HTTP session clustering services to improve service processing efficiency.

If an error occurs in the web engine that first processed the request, requests that are received after the error occurrence will be sent to another web engine to provide seamless services. A cluster is required to perform this task. For more information about HTTP session clustering, refer to "Distributed Session Servers" in *JEUS Session Management Guide*.

Even if web engines are used without web servers, equipment or software that can deliver further requests to the web engine that processed the first request can be used to increase efficiency like

when using web servers. However, JEUS neither provides this kind of software nor recommends using one.

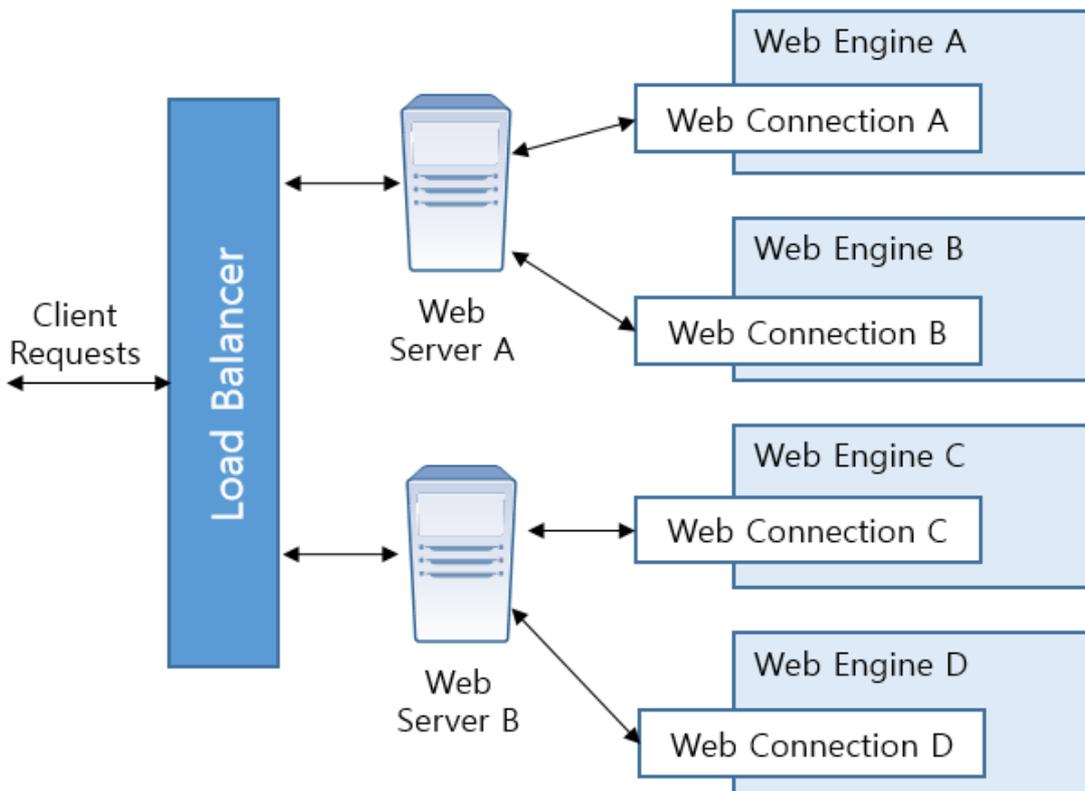


Web server and mod_jk configurations can differ according to the web server version. The configuration method described here is used for enhancing the understanding of JEUS. **When configuring web servers in the production environment, refer to each relevant web server documentations and configuration examples provided by local/overseas communities.**

2.4.1. Load Balancing Structure

When a web site needs to handle a large number of requests, a single web server and web engine are not sufficient in handling the requests. In such cases, multiple servers and web engines are required for load balancing.

The following diagram shows a load balancing structure in which two web servers are connected to four web engines.



Load Balancing in a Small Website

The same web context must be deployed to each engine. In this environment, the most important thing is whether the sessions are shared or not. Web engines must be clustered in order to manage applications more easily or if distributed sessions are required. For more information about session clustering, refer to "Session Tracking" in *JEUS Session Management Guide*.

2.4.2. Apache

Execute the following steps to integrate Apache with the web engines.

1. Install the mod_jk library in Apache.
2. Configure an AJP13 listener in the JEUS web engine.
3. Create and modify the workers.properties file.
4. Insert the integration settings in the httpd.conf file for the Apache web server.
5. Restart the Apache web server.



The description in this guide is based on Apache 2.2.4 and mod_jk 1.2.20.

Installing the mod_jk Library

To use an Apache web server as a front-end server for the web engine, the mod_jk module must be added to the Apache install module. The mod_jk module implements the Apache JServ Protocol 1.3 (AJP 1.3), a communication protocol between servers and engines.



Download the source from [Apache Tomcat Connector Download Page](#), and compile it on the server.

Configuring AJP13 Listeners

After the mod_jk library has been installed, configure the AJP13 listener in the JEUS web engine. For more information about configuring AJP13 listeners, refer to [AJP Listeners](#).

Configuring the workers.properties File

The workers.properties file is a configuration file for mod_jk. For example, there are two JEUS servers named server1 and server2 and the host names are server1 and server2. The name of the web engine for each server is server_servlet and server2_servlet. The AJP listeners configured on each web engine are 9901 and 9902.

The following is the example of a workers.properties file in the described JEUS environment.

mod_jk Configuration File: <workers.properties>

```
worker.list=jeus_load_balancer_workers
worker.jeus_load_balancer_workers.type=lb
worker.jeus_load_balancer_workers.sticky_session=true

#####
# listener specific configuration
#####
worker.jeus_load_balancer_workers.balance_workers=server1
```

```

worker.server1.reference=worker.template
worker.server1.host=192.168.0.101
worker.server1.port=9901
worker.server1.route=ZG9tYWluMS9zZXJ2MQ==

worker.jeus_load_balancer_workers.balance_workers=server2
worker.server1.reference=worker.template
worker.server1.host=192.168.0.102
worker.server1.port=9902
worker.server1.route=ZG9tYWluMS9zZXJ2Mg==

#####
# common config
#####
worker.template.type=ajp13
worker.template.socket_connect_timeout=5000
worker.template.socket_keepalive=true
worker.template.ping_mode=A
worker.template.ping_timeout=10000
worker.template.connection_pool_minsize=0
worker.template.connection_pool_timeout=600
worker.template.reply_timeout=300000
worker.template.recovery_options=3

```

The workers.properties file uses the "worker.<worker_name>.<directive>=<value>" format to define most settings.

The following describe the major configuration items.

- **worker.list**

Worker names. Multiple workers can be defined and each worker name is separated by a comma (",").

The following example defines a worker with the name "jeus_load_balancer_workers".

```
worker.list=jeus_load_balancer_workers
```

- **worker.<worker_name>.type**

Worker types. 'ajp13', 'ajp14', 'jni', 'lb', and 'status' can be used. To support AJP13, setting 'ajp13', and 'lb' are sufficient in JEUS.

The following example defines "jeus_load_balancer_workers" as a load balancer.

```
worker.jeus_load_balancer_workers.type=lb
```

- **worker.<worker_name>.balance_workers**

A comma (',') can be used to list the workers for load balancing. The workers written in worker.list

above cannot be in this list.

When integrating with an AJP13 listener in JEUS, the worker names must be the servlet engines of JEUS for the workers to properly function as load balancers or sticky sessions. JEUS uses the servlet engine names for the session ID routing information, and mod_jk uses the worker names.

- **worker.<worker_name>.sticky_session**

Option to support routing based on session ID. Options are 'true'(or 1) or 'false'(or 0). Since JEUS supports sticky sessions, this must be always set to 'true'. (Default value: true)

The following example sets the "jeus_load_balancer_workers" setting to support sticky sessions.

```
worker.jeus_load_balancer_workers.sticky_session=true
```

- **worker.<worker_name>.host**

Host name or IP address where the AJP13 listener exists.

- **worker.<worker_name>.port**

AJP listener port number in JEUS.

- **worker.<worker_name>.reference**

Useful for configuring multiple load balancer workers. The specified workers can be referenced. For example, if "worker.castor.reference=worker.template" is set, all worker.template configurations will be inherited except configuration values that are explicitly set for castor workers.

- **worker.<worker_name>.route**

A sticky session is a session routing technology used in sessions that are created by the JEUS session manager. By matching the session with a worker, this can improve the availability of local sessions. For more information about session routing, refer to "Session Tracking Structure" in *JEUS Session Management Guide*.

The value domain_name/server_name is encoded in Base16. The encoded value can be retrieved by using the encryption command in the 'JEUS_HOME/bin' directory.

The following is an example of using the encryption command.

```
${JEUS_HOME}/bin/encryption -algorithm base64 -text domain1/server1
```

Note that a "domain name/server" format is used as in "domain1/server1" or "domain1/server2".



Since this configuration information may not be current for the latest version of mod_jk, use the examples in this section only for reference. To configure

the settings, mod_jk related descriptions in the Tomcat site must be followed.

Configuring httpd.conf

To use the mod_jk module, add the following to the httpd.conf file.

Configuring mod_jk in Apache: <httpd.conf>

```
. . .
LoadModule    jk_module    "/usr/local/apache/modules/mod_jk.so"
JkWorkersFile "/usr/local/apache/conf/workers.properties"
JkLogFile     "/usr/local/apache/logs/mod_jk.log"
JkLogLevel    info
JkMount /examples/* jesus_load_balancer_workers
. . .
```



Since this configuration information may not be current for the latest version of Apache, use the examples in this section only for reference. Refer to the Apache manuals.

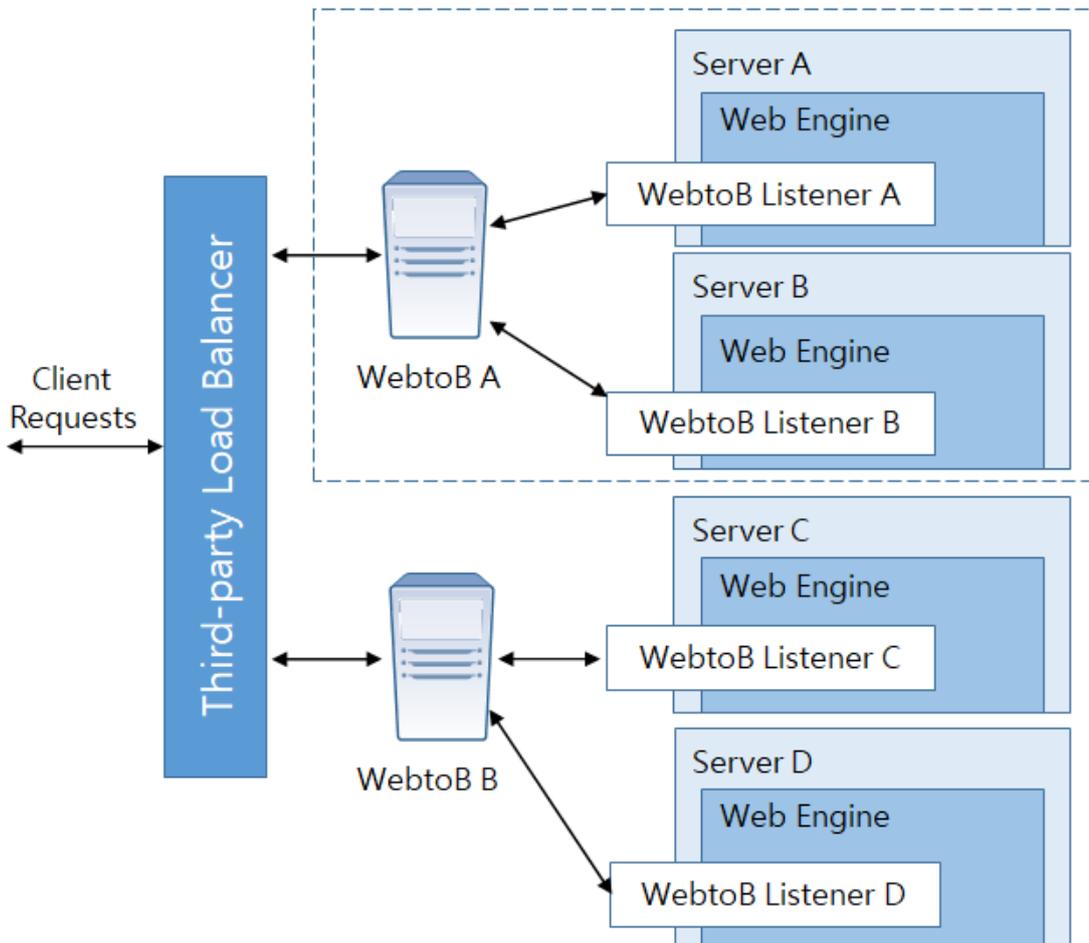
2.4.3. IIS and Iplanet Web Servers

IIS and Iplanet support the AJP13 protocol. The workers.properties file is configured in the same way as JEUS, but mod_jk is installed and configured differently. Refer to the manual of each web server.

2.4.4. Load Balancing with WebtoB

This section describes how to configure load balancing between WebtoB and JEUS with some examples.

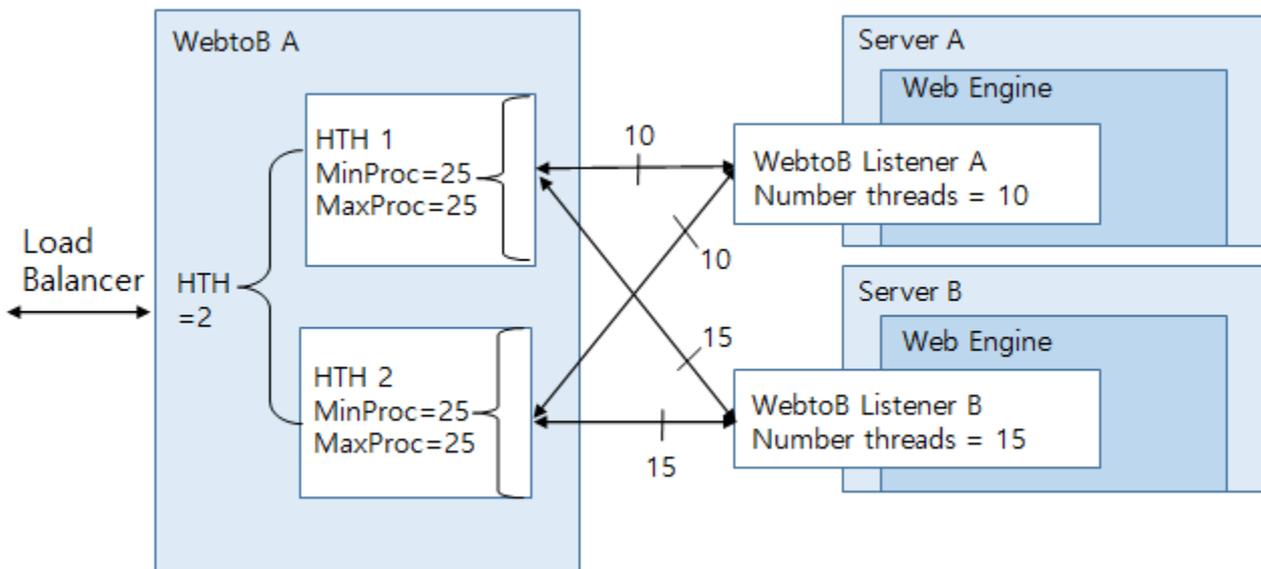
The following shows the server structure of the example. Each engine is connected a WebtoB server. In the following example, there are two WebtoB servers and each one is connected to two web engines.



Load Balancing Server - Each of Two WebtoB Connected to Two Web Engines

The same context must be deployed to each server. In general, in a structure like the previous figure, servers A through D become clustered and distributed session servers are used accordingly. Clustering is not necessary if user sessions are not used.

The following diagram shows the dotted box above in detail and displays each WebtoB connector configurations.



WebtoB Connector Configurations

When configuring WebtoB connectors, the number of WebtoB HTH processes must be considered.

A WebtoB HTH process behaves as an engine having multiple sub-processes. It establishes a 1 to 1 connection with the worker thread in the WebtoB connector. Therefore, it must be configured by considering the '**connection-count**' value for the WebtoB connector and the '**MaxProc**' value in the WebtoB configuration.

The number of HTH processes in the WebtoB configuration can be set in the '**Hth Count**' setting for the WebtoB connector. The '**MaxProc**' value of WebtoB and the '**connection-count**' value of the WebtoB connector can be expressed using the following formulas.

```
MinProc = Listener A connection-counts + Listener B connection-counts + ... + Listener X connection-counts setting.  
MaxProc = Listener A connection-counts + Listener B connection-counts + ... + Listener X connection-counts setting.
```

The '**Hth Count**' of the WebtoB connector must be the same as the number of HTH processes in the *NODE configuration in the WebtoB configuration file. For more information about configuring WebtoB connectors, refer to [WebtoB Connectors](#).

The following example of the http.m file configures WebtoB A from the previous example.

WebtoB Configuration: <http.m>

```
*NODE  
foo    ...  
    HTH = 2,  
    JSVPORT = 9900,  
    ...  
*SVRGROUP  
jsvg   NODENAME = foo, SVRTYPE = JSV  
  
*SERVER  
default   SVGNAME = jsvgl, MinProc = 6, MaxProc = 25  
  
*URI  
uri1   Uri = "/examples/", Svrtype = JSV  
uri2   Uri = "/test/", Svrtype = JSV  
  
*EXT  
jsp    MimeType = "application/jsp", SvrType = JSV
```

The connection count must be between 6 and 25.

2.5. Using TCP Listeners

TCP listeners can define a communication protocol between TCP clients and TCP servlets, but they are only recommended when the HTTP and HTTPS protocols are not sufficient.

Execute the following steps to use a TCP listener.

1. Define a customized communication protocol.
2. Implement the dispatcher config class (Protocol configuration)
3. Implement the TCP handler servlet (Protocol implementation)
4. Set the TCP listener to implement the custom protocol.
5. Implement the TCP client.
6. Compile and run the TCP client.

The following are the terms used in this section.

Term	Description
Protocol(commu nication protocol)	Defines the message structure and contents that are exchanged between the TCP client and TCP handler (TCP listener is a mediator).
Message	Data exchanged between the TCP client and TCP handler. Must follow the structure defined in the protocol.
TCP Client	External application, which is processed by the TCP handler, that communicates and exchanges messages with the TCP listener. A standard socket is used to exchange messages.
TCP Handler	Receives messages from the TCP listener and processes them. The TCP handler is implemented as a subclass of <code>jeus.servlet.tcp.TCPServlet</code> and is registered in the web engine like regular servlets. The TCP handler works as if it is the provider or implementation class of the custom protocol that exist above the TCP protocol.
TCP Dispatcher Configuration Class	Extended class of <code>jeus.servlet.tcp.TCPDispatcherConfig</code> . This class sends the custom protocol information to the TCP listener and processes non-HTTP messages. Place the class in the ' <code>DOMAIN_HOME/lib/application</code> ' directory, and configure ' Dispatcher Config Class ' for the TCP listener in the web engine configuration. (See TCP Listeners)
TCP Listener	Interprets custom messages and provides the routing infrastructure. This acts as a non-HTTP mediator between the TCP client and TCP handler. Similar to other HTTP listeners, this exists in the web engine.

2.5.1. Defining Custom Communication Protocols

The TCP listener divides all messages that are exchanged between TCP client and TCP handler into two parts, the header and body. In general, the header is fixed in size and contains standard information. The body contains user data, such as the HTML code in an HTTP response, to transmit.

This section defines a simple protocol (message structure) to describe how to use the TCP listener.

The custom communication protocol (custom message structure) contains the following contents.

- Header
 - The header starts with a 4 byte **magic** number. This is used to identify the protocol. For the example, set it to '7777'.
 - Next is a 1 byte **type** field. '0' indicates a request and '1' indicates a response.
 - The third item is a 4 byte **body length**. This item contains the body length in bytes. It is fixed at 128 bytes.
 - The last item is a 32 byte string called **service name**. For the example, enter the name of the TCP servlet that handles the request.
- Body
 - The body contains 128 bytes of character data. In this example, enter the 128 byte block as two 64 byte strings.

2.5.2. Implementing Dispatcher Configuration Classes

A dispatcher configuration class is a subclass of the `jeus.servlet.tcp.TCPDispatcherConfig` class. The abstract method of the class must implement the information required for the TCP listener to deliver messages to the appropriate TCP handler. The methods are described in `JEUS_HOME/docs/api/jeus-servlet/index.html`.

From JEUS 7 Fix #1 onwards, the `getBodyLengthLong` method has been added. If the TCP Body is 2 GB or more, the Body length can be set to 8 bytes by using this method.

The following code shows how the dispatcher configuration class is implemented based on the defined protocol. To understand how the methods are used, refer to comments in the following example.

TCPDispatcherConfig Implementation: <SampleConfig.java>

```
package sample.config;

import jakarta.servlet.*;
import jakarta.servlet.http.*;
import jeus.servlet.tcp.*;

/*
 * This class extends the abstract class jeus.servlet.tcp.
 * TCPDispatcherConfig class. This implementation provides
 * routing and handling information to the TCP listener
 * according to our defined communications protocol.
 */
public class SampleConfig extends TCPDispatcherConfig {
    /*
     * Any init code goes into this method. This method will be
     * Called once after starting the TCP listener.
     * We leave this method empty for our simple example.
     */
    public void init() {}
}
```

```

/*
    This method returns the fixed-length of the header so
    that the TCP listener knows when to stop
    reading the header. The header length for
    our example is 41 (bytes): 4 (magic) + 1
    (type) + 4 (body length) + 32 (service name).
*/
public int getHeaderLength() {
    return 41;
}

/*
    This method must return the length of the body.
    For our example, this length is represented as an
    "int" in the header, starting at position "5"
    (see the protocol definition above).
*/
public int getBodyLength(byte[] header) {
    return getInt(header, 5);
}

/**)
 * Returns the long-size length of request body of
 * incoming request packet.
 * If you don't need to support long, you may map to
 * {@link #getBodyLength(byte[])}.
 */
public long getBodyLengthLong(byte[] header) {
    return getBodyLength(header);
}

/*
    This method must return the context path so that the
    request can be routed by the TCP listener to the context
    that contains the TCP handler (TCPServlet implementation).
    For our example, we always use the context path "/tcptest".
*/
public String getContextPath(byte[] header) {
    return "/tcptest";
}

/*
    This method must return the name (path) of the TCP
    handler(TCPServlet) relative to the context path.
    For our example, we fetch this name from the 9th position
    in the header.
*/
public String getServletPath(byte[] header) {
    return "/" + getString(header, 9, 32);
}

/*
    This method returns some path info from the header.
    It is not used in our example and thus returns "null".
*/
public String getPathInfo(byte[] header) {
    return null;
}

```

```

    /*
     * This method returns any session ID embedded in the header.
     * It is not used in our example and thus returns "null".
     */
    public String getSessionId(byte[] header) {
        return null;
    }

    /*
     * This method determines whether the TCP listener
     * should keep the socket connection open after the TCP handler
     * has delivered its response. If it returns "false", the
     * connection will be dropped like in HTTP communications.
     * If it returns "true" the connection will be kept open
     * like in the Telnet or FTP protocols. For our example,
     * we choose to make it persistent (connection not closed
     * by the TCP listener).
     */
    public boolean isPersistentConnection() {
        return true;
    }
}

```

2.5.3. Implementing TCP Servlets

TCP servlets are always a subclass of the `jeus.servlet.tcp.TCPServlet` class. The handler always contains the abstract void service (`TCPServletRequest req`, `TCPServletResponse, res`) method that is overridden.

The service method must be implemented to process messages that follow the custom protocol. The web container reads a header and sends it to the `TCPServletRequest` object, and the TCP servlet writes a response to an output stream of the `TCPServletResponse` object.

The following is an implementation of the TCP servlet that processes the message that follows the custom protocol.

TCPServlet Implementation: `<SampleTCPServlet.java>`

```

package sample.servlet;

import java.io.*;
import jakarta.servlet.*;
import jakarta.servlet.http.*;
import jeus.servlet.tcp.*;

/**
 * Sample TCPServlet implementation
 *
 * Protocol scheme:
 *
 * common header (for request and response) : total 41 byte
 *
 * magic field: length = 4 byte, value = 7777
 * type field : length = 1 byte, 0 : request, 1:response
 * body length field : length = 4, value = 128
 */

```

```

*   service name field : length = 32
*
* request and response body
*
*   message1 field : length = 64
*   message2 field : length = 64
*
*/

public class SampleTCPServlet extends TCPServlet {

    public void init(ServletConfig config) throws ServletException {
    }

    public void service(TCPServletRequest req, TCPServletResponse res)
        throws ServletException, IOException {
        ServletContext context = req.getServletContext();
        byte[] header = req.getHeader();
        byte[] body = new byte[req.getContentLength()];
        int read = req.getInputStream().read(body);
        if (read < body.length) {
            throw new IOException("The client sent the wrong content.");
        }

        String encoding = res.getCharacterEncoding();
        if (encoding == null)
            encoding = "euc-kr";

        DataInputStream in = new DataInputStream(new ByteArrayInputStream(header));
        int magic = in.readInt();
        context.log("[SampleTCPServlet] received magic = " + magic);

        byte type = (byte)in.read();
        context.log("[SampleTCPServlet] received type = " + type);

        int len = in.readInt();
        context.log("[SampleTCPServlet] received body length = " + len);

        byte[] svcname = new byte[32];
        in.readFully(svcname);
        context.log("[SampleTCPServlet] received service name = "
            + (new String(svcname)).trim());

        String rcvmsg = null;
        rcvmsg = (new String(body, 0, 64)).trim();
        context.log("[SampleTCPServlet] received msg1 = " + rcvmsg);

        try {
            rcvmsg = (new String(body, 64, 64, encoding)).trim();
        } catch (Exception e) {}
        context.log("[SampleTCPServlet] received msg2 = " + rcvmsg);

        String msg1 = "test response";
        String msg2 = "test response2";

        byte[] result1 = null;
        byte[] result2 = null;
        if (encoding != null) {
            try {

```

```

        result1 = msg1.getBytes(encoding);
        result2 = msg2.getBytes(encoding);
    } catch (UnsupportedEncodingException uee) {
        result1 = msg1.getBytes();
        result2 = msg2.getBytes();
    }
} else {
    result1 = msg1.getBytes();
    result2 = msg2.getBytes();
}

header[4] = (byte)1; // mark as response
ServletOutputStream out = res.getOutputStream();
out.write(header);

byte[] buf1 = new byte[64];
System.arraycopy(result1, 0, buf1, 0, result1.length);
out.write(buf1);

byte[] buf2 = new byte[64];
System.arraycopy(result2, 0, buf2, 0, result2.length);
out.write(buf2);

out.flush();
}

public void destroy() {
}
}

```



From JEUS 7 Fix #2 onwards, it is not recommended to use the `TCPServletRequest.getBody()` method. This method may cause a memory issue if the request body is too big. It is recommended to use the `ServletInputStream` class defined in the servlet standard instead. This object can be obtained by using the `TCPServletRequest.getInputStream()` method.

2.5.4. Configuring TCP Listeners for Custom Protocols

Configure the TCP listener based on the `SampleConfig.java` and `SampleTCPServlet.java` implementations and using the following steps.

1. Configure the TCP listener information in the web engine. For more information about TCP listeners, see [TCP Listeners](#). Ensure that the server listener port to which the TCP listener refers to '5555', and set '**Dispatcher Config Class**' to 'sample.config.SampleConfig'.
2. Start the JEUS server and deploy the web application that contains the previous TCP servlet. The following is an example of the deployment descriptor for the web application.

TCP Handler Deployment Descriptor: <web.xml>

```

<?xml version="1.0" encoding="UTF-8"?>
<web-app 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/web-app_5_0.xsd"
        metadata-complete="false"
        version="5.0">
    <display-name>test</display-name>
    <distributable/>
    <servlet>
        <servlet-name>SampleServlet</servlet-name>
        <servlet-class>sample.servlet.SampleTCPServlet</servlet-class>
        <load-on-startup>0</load-on-startup>
        <async-supported>true</async-supported>
    </servlet>
    <servlet-mapping>
        <servlet-name>SampleServlet</servlet-name>
        <url-pattern>/sample</url-pattern>
    </servlet-mapping>
    <login-config>
        <auth-method>BASIC</auth-method>
    </login-config>
</web-app>

```

2.5.5. Implementing TCP Clients

TCP clients connect to TCP listeners using a socket and uses the connection to send messages. The messages are byte streams that follow the custom protocol defined in [Defining Custom Communication Protocols](#).

The TCP listener receives the messages and calls the `service()` method of the `SampleTCPServlet` class based on dispatch information defined in the `SampleConfig` class. The `SampleTCPServlet` creates the response based on the client data and delivers it. The client receives the response and outputs it using `System.out`.

The following is the example of a TCP client implementation.

TCP Client Implementation: <Client.java>

```

package sample.client;

import java.io.*;
import java.net.*;

public class Client {
    private String address;
    private int port;

    private int magic = 7777;
    private byte type = 0;
    private int bodyLength = 128;
    private byte[] serviceName="sample".getBytes();

    public Client(String host, int port) {
        this.address = host;
        this.port = port;
    }
}

```

```

public void test()
    throws IOException, UnsupportedEncodingException {
    Socket socket = new Socket(address, port);
    DataOutputStream out = new DataOutputStream(
        new BufferedOutputStream(socket.getOutputStream()));
    DataInputStream in = new DataInputStream(
        new BufferedInputStream(socket.getInputStream()));

    out.writeInt(7777);
    out.write(type);
    out.writeInt(bodyLength);
    byte[] buf = new byte[32];
    System.arraycopy(serviceName, 0, buf, 0, serviceName.length);
    out.write(buf);
    byte[] msg1 = "test request".getBytes();
    byte[] msg2 = "test request2".getBytes();
    buf = new byte[64];
    System.arraycopy(msg1, 0, buf, 0, msg1.length);
    out.write(buf);
    buf = new byte[64];
    System.arraycopy(msg2, 0, buf, 0, msg2.length);
    out.write(buf);

    out.flush();

    // rx msg
    int magic = in.readInt();
    System.out.println("[Client] received magic = " + magic);

    byte type = (byte)in.read();
    System.out.println("[Client] received type = " + type);

    int len = in.readInt();
    System.out.println("[Client] received body length = " + len);

    byte[] svcname = new byte[32];
    in.readFully(svcname);
    System.out.println("[Client] received service name = " +
        (new String(svcname)).trim());

    byte[] body = new byte[128];
    in.readFully(body);
    String rcvmsg = null;
    rcvmsg = (new String(body, 0, 64)).trim();
    System.out.println("[Client] received msg1 = " + rcvmsg);
    rcvmsg = (new String(body, 64, 64, "euc-kr")).trim();
    System.out.println("[Client] received msg2 = " + rcvmsg);

    out.close();
    in.close();
    socket.close();
}

public static void main(String[] argv) throws Exception {
    Client client = new Client("localhost", 5555);
    client.test();
}
}

```

In the previous client code, note the various field configurations of the header that are required for the protocol.

The 'magic' number is set to '7777', the 'type' is set to '0' (request), the 'body length' is set to '128 bytes' (fixed length), and the 'service name' is set to 'sample'. (**The name of SampleServlet is specified in web.xml.**) After creating two messages, the header information is sent and the messages are sent to the TCP listener. Finally, the 'hostname' is set to 'localhost' and the port number is set to '5555'.

2.5.6. TCP Client Compilation and Execution

Assume that the TCP listener is set to 'localhost' with the port number '5555'. The TCP client is compiled and executed using the following steps.

1. Compile Client.java.

```
javac -classpath ${JEUS_HOME}/lib/system/jeus.jar -d . Client.java
```

2. Execute Client.class.

```
java -classpath ${JEUS_HOME}/lib/system/jeus.jar:. sample.client.Client
```

3. The JEUS administrator console must display the result of the TCP handler. (SampleServlet class).

```
[SampleServlet] received magic = 7777  
[SampleServlet] received type = 0  
[SampleServlet] received body length = 128  
[SampleServlet] received service name = sample  
[SampleServlet] received msg1 = test request  
[SampleServlet] received msg2 = test request2
```

4. The following messages are displayed on the client screen.

```
[Client] received magic = 7777  
[Client] received type = 1  
[Client] received body length = 128  
[Client] received service name = sample  
[Client] received msg1 = test response  
[Client] received msg2 = test response2
```

2.6. Using HTTP/2

HTTP/2 is the next version of HTTP/1.1. It is a new and improved HTTP standard.

The following describes the characteristics of HTTP/2.

- **Header Compression**

HTTP/2 uses Huffman coding to compresses and send headers. Also, each server and client creates an index in the header table so that repeated headers are not sent multiple times. The number of headers to be indexed can be set by using Settings Header Table Size.

- **Request/Response Multiplexing**

HTTP/2 refers to a series of request/reponse as a stream. A stream consists of multiple frames such as a headers frame and a data frame. The basic protocol unit in HTTP/2 is a frame. In HTTP/1.1, a single request can be sent after completing a single request/response. However in HTTP/2, multiple streams (requests) can be created and sent concurrently.

The Settings Max Concurrent Streams setting specifies how many streams are to be used concurrently. The size of a data frame can be set through the Settings Max Frame Size setting.

- **Server Push**

For information about Server Push, refer to [Server Push](#).



This guide only provides information about how to use HTTP/2 in JEUS. For more information about HTTP/2, refer to "[RFC Documentation](#)".

2.6.1. Configuring HTTP/2

HTTP/2 uses two identifiers, **h2c** and **h2**. The following table describes each identifier.

Term	Description
h2c	Indicates HTTP/2. "HTTP" is used.
h2	Indicates HTTP/2 that runs by using TLS (Transport Layer Security). "https" is used.



Most browsers only support **h2**, and do not support **h2c**.

To utilize HTTP/2 by using **h2**, additional work is required. h2 runs by using TLS (Transport Layer Security), and must use the ALPN (Application Layer Protocol Negotiation) function during a TLS handshake process. Since h2 runs by using TLS, it must use a port set with a secure listener.

The following tasks are required to use ALPN.



Since OpenJDK 8u252 version, ALPN is supported by JDK itself from OpenJDK 8u252 version onwards, so Jetty ALPN setting is not necessary as a JVM option. This may vary depending on the JDK.

1. In the following address, check the appropriate alpn-boot library for the JDK version to be used, and then download it in the "[MVN Repository](#)".

```
http://www.eclipse.org/jetty/documentation/current/alpn-chapter.html#alpn-versions
```



The alpn-boot library only supports OpenJDK and OracleJDK. Currently, HTTP/2 cannot be used in IBM JDK.

2. Add the <jvm-option>, which adds the downloaded alpn-boot library to bootclasspath, to domain.xml.

```
<jvm-config>
  <jvm-option>-Xbootclasspath/p:<path_to_alpn_boot_jar> ...</jvm-option>
</jvm-config>
```



1. The HTTP/2 specification specifies cipher suites that are not recommended (among the cipher suites required for TLS communication). (<http://httpwg.org/specs/rfc7540.html#BadCipherSuites>) According to the specification, if a cipher suite that is not recommended is used, connection may be disabled. None of the cipher suites provided in JDK 7 are recommended to be used, so it is required to use JDK 8 (which uses more powerful encryption suites).

In the HTTP/2 specification, TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA25 is specified to be supported. Therefore, it is recommended that this encryption suite is set in the Listener SSL setting.

2. For more information about settings related to the JEUS server, refer to "Listener Configuration" in *JEUS Server Guide*.

2.6.2. Server Push

Server Push is a function in which a response is sent by a server even if a client does not send a request for a specified resource. This decreases the amount of time required for sending a request, which results in faster response. To use the Server Push function, specify a resource to be server pushed during web application development.



Server Push is implemented based on the Servlet 4.0 API.

The following describes how Server Push is used: a PushBuilder object is obtained, then the resource path is specified, and then the push() method is called.

Server Push gets a PushBuilder object, and then specifies the resource path, and then calls the push() method. The following is an example of using Server Push.

Using Server Push: <ServerPushServlet.java>

```
package sample.serverpush

...

@WebServlet("/ServerPush")
public class ServerPushServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException,
    IOException {

        PushBuilder builder = req.newPushBuilder();
        String pushResourcePath = "resources/a.txt";

        // Enter the path of the resource to push and then call the push() method.
        builder.path(pushResourcePath);
        builder.push();

        resp.getWriter().write("main page\n");
    }
}
```



If Server Push is specified to be not used, then the above example will have no effect.

2.7. Tuning Listeners

Consider the following when configuring listeners for optimal performance.

- Increase the out buffer size by using many system resources or prolonging the waiting time.
- In general, if the min, max, and step values of the worker thread pool are large, performance improves when many clients access the web engine. To use less memory, lower these values.
- Disabling the '**Server Access Control**' setting may improve performance.
- As described earlier, in the WebtoB connector, set the number of worker threads for the web connector of the web engine equal to the value in http.m.

3. Web Contexts

This chapter describes how to package, deploy, and monitor JEUS web applications.

3.1. Overview

Web contexts are created when web applications are deployed to web engines. Since web applications and web contexts are practically synonymous, the terms 'web context' will be mainly used to signify 'web application'.

3.2. Basic Structure

This section describes the content and structure of web contexts.

3.2.1. Web Context Contents

Web applications consist of static and dynamic contents that are required to run web-based services requested by clients.

- Static content

Static content is any kind of data that will be returned without being processed by the web engine. The following are the available static content types.

- HTML pages
- Text files
- Image files
- Videos
- Others

- Dynamic content

Contents that are created as responses to user requests such as servlets and JSPs.

3.2.2. Web Context Structure (WAR File Structure)

Before being deployed to web engines, a web context must be packaged into a JAR file, a compressed file with the extension '.war'. The packaged file is called a WAR (Web ARchive) file.

The following is the structure of a WAR file.

```
WEB WAR
|--[T]index.html
```

```

|--static
|   |--[T].html
|--jsp
|   |--.jsp
|--images
|   |--[T].jpg, .gif
|--META-INF
|   |--[T]MANIFEST.MF
|--WEB-INF
|   |--[X]web.xml
|   |--[X]jeus-web-dd.xml
|   |--[X]ejb-jar.xml
|   |--[X]jeus-ejb-dd.xml
|--classes
|   |--[C]Servlet.class
|   |--[C]EJB.class
|--lib
|   |--[J]Library JAR
|--tlds
|   |--[T].tld

```

* 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/

This directory is optional. If this directory is used, it contains the "MANIFEST.MF" file, a descriptor file defined in the JAR format.

WEB-INF/

This directory is required. It contains servlets, filters, listener classes, and libraries. The following components are in this directory.

Component	Description
web.xml	Jakarta EE web application deployment descriptor file. It contains meta-information about the web application. Starting from Servlet 3.0, web.xml is not necessarily required. Instead, servlets, filters, and listeners can be added using annotations and the registration API.
jeus-web-dd.xml	JEUS web application deployment descriptor. For more information, refer to Configuring jeus-web-dd.xml .
ejb-jar.xml	From Java EE 6 onwards, EJBs can be included in a WAR file. EJBs can also be defined as an annotation like servlets. For more information, refer to the EJB standards or "JEUS EJB Guide".
jeus-ejb-dd.xml	

Component	Description
classes/	Contains servlet classes and utility classes in its subdirectories. It is organized into a standard Java package structure.
lib/	Contains the Java libraries that are required for web applications. The libraries are packaged into JAR files and stored in this path. The files are automatically added to the classpaths of all servlets.
tlds/	Contains custom tag library descriptors for JSP pages.

Others

Directories that contain contents such as JSP, HTML, and image files.

3.3. Deploying Web Contexts

This section describes how to deploy web contexts. Deployment is the process that creates web contexts and prepares them to run on JEUS web engines.

Logically, web contexts exist on a virtual host. For more information about virtual hosts, refer to [Virtual Hosts](#).

3.3.1. Configuring jeus-web-dd.xml

The web.xml and jeus-web-dd.xml files should be created under the 'WEB-INF/' directory only when necessary.

The following is a sample jeus-web-dd.xml file. It shows only a specific part of the file. To see the full file with all configuration items, refer to "13. Configuration of jeus-web-dd.xml" in "JEUS XML Reference".

Web Context Configuration File: <jeus-web-dd.xml>

```
<jeus-web-dd xmlns="http://www.tmaxsoft.com/xml/ns/jeus" version="9">
  <context-path>/examples</context-path>
  <enable-jsp>true</enable-jsp>
  <auto-reload>
    <enable-reload>true</enable-reload>
    <check-on-demand>true</check-on-demand>
  </auto-reload>
  <added-classpath>
    <class-path>/home/user1/libs/lib.jar</class-path>
  </added-classpath>
  <session-config>
    . . .
  </session-config>
  <thread-pool>
    <base>
      <name>base</name>
      <min>1</min>
```

```

    <max>5</max>
  </base>
  <service-group>
    <name>svcg1</name>
    <uri>/service1,/service2</uri>
    <min>10</min>
    <max>20</max>
  </service-group>
  <service-group>
    <name>longJsp</name>
    <uri>/jsp/servlet/asyncContext.jsp</uri>
    <min>5</min>
    <max>10</max>
  </service-group>
  <service-group>
    <name>Jsp</name>
    <uri>/jsp/servlet</uri>
    <min>1</min>
    <max>5</max>
  </service-group>
</thread-pool>
<upgrade>
  <upgrade-executor>
    <min>0</min>
    <max>30</max>
    <keep-alive-time>60000</keep-alive-time>
    <queue-size>4096</queue-size>
  </upgrade-executor>
</upgrade>
<async-config>
  <async-timeout-millis>60000</async-timeout-millis>
  <background-thread-pool>
    <min>0</min>
    <max>10</max>
  </background-thread-pool>
  <dispatch-thread-pool>
    <min>0</min>
    <max>10</max>
  </dispatch-thread-pool>
</async-config>
<properties>
  <property>
    <key>jeus.servlet.jsp.modern</key>
    <value>true</value>
  </property>
</properties>
</jeus-web-dd>

```

The following describes each configuration tag.

Tag	Description
<context-path>	Context root path. Must begin with a "/" and be unique in the virtual host. In the URL "http://www.foo.com/examples/index.html", the context path is "/examples".
<enable-jsp>	If set to false, JSP will not be used. By default, JSP is supported.

Tag	Description
<user-log>	User logs that are created by web applications through the servlet context's log method. For more information, refer to Configuring User Logs .
<auto-reload>	Option to automatically reload servlet classes and custom tag classes when they are modified. For more information, refer to Reloading Classes at Runtime .
<added-classpath>	List of libraries that are referenced when a servlet program is compiled or executed. Can use directories.
<allow-indexing>	Displays the list of URLs where the directory list is displayed upon client request. The directory list is displayed when the file requested by a client does not exist. Clicking on a file in the directory list will show the file contents.
<deny-download>	Filter that defines resources that cannot be accessed or downloaded directly. These filters are file name extensions. You can use paths that include file names. File names and directories are specified as relative paths to the directory context document base.
<aliasing>	Directory mapping for custom URL paths. This can be used to map a directory that is in a different location from the URL request path.
<file-caching>	Determines the static contents that will be cached in the runtime memory to improve response times. Child configurations include the maximum size of cache memory in megabytes, the maximum size of static content in cache, and the path to the contents that will be cached.
<jsp-engine>	Setting for JSP pages that are included in the web contexts. For more information about <jsp-engine>, refer to Configuring jeus-web-dd.xml .
<session-config>	Sessions that are used in the context. This has a higher priority than the web engine configurations. For more information about configuring sessions, refer to Sessions .
<webinf-first>	Option for class loading priority. If this is set to 'true', classes in WEB-INF will be loaded first for this application. If this is set to 'true', a specific package or class can be excluded when class loading related collision occurs as follows from JEUS 7 Fix#4 onwards.
	<pre> <webinf-first> <enabled>true</enabled> <excluded-package>package.name.</excluded-package> ... </webinf-first> </pre>

Tag	Description
<attach-stacktrace-on-error>	Option to display the error details in the browser if an error occurs on the server. The message can be helpful in a development environment, but it is not recommend for use in a production environment.
<keep-alive-error-response-codes>	<p>Error code that will send a Keep-Alive response instead of a Connection: Close. It is applied when the error is sent by the web application itself, but not when the engine decides that the connection needs to be terminated internally.</p> <p>For example, it is applied when a servlet implements a response.sendError(500) but not applied when the engine sends the 500 error because an exception occurred in the servlet. If multiple response codes need to be configured, separate them with a "," like "404, 503".</p>
<encoding>	<p>Encoding configuration for requests and responses. This is the same as <encoding> in domain.xml. If encoding configuration is set in both jeus-web-dd.xml and domain.xml, <request-encoding> and <response-encoding> in jeus-web-dd.xml has priority. For more information, refer to Encoding.</p> <ul style="list-style-type: none"> ◦ Request Encoding: URL Query String, application/x-www-form-urlencoded Body, getReader(). From JEUS 7 Fix #4, <request-encoding><url-mapping> is added to use encoding mapped to a specific Servlet path. However, it will be ignored if <forced> or <client-override> is set. ◦ Response Encoding: Encoding for response HTTP bodies.
<cookie-policy>	Policy that is applied when reading cookies from an HTTP request header or adding new cookies to an HTTP response header according to the application request. For more information, refer to Cookie Policy .
<async-config>	Async processing that was added in Servlet 3.0. Child configurations such as async timeout, background thread pool, and dispatch thread pool can be configured. For more information, refer to the servlet standards.
<upgrade>	Tread pool configuration internally used by the container to process Upgrade inbound messages.
<web-security>	Configuration group of security policies that are limited to web applications including the policy configuration of address validation when using sendRedirect.
<websocket>	Configuration about the web application's WebSocket Container. For more information, refer to Configuring WebSocket Container .
<properties>	Properties that apply to the web application. These properties only apply to the application that they are configured for.



The default value of <async-config><async-timeout-millis> is 30 seconds. If an error occurs because an operation performed by a background thread is taking too long, set this value in the jeus-web-dd.xml or adjust the timeout in the program code by using the jakarta.servlet.AsyncContext#setTimeout() method.

The following describes each tag for the thread pool configuration.

Tag	Description
<base>	Default thread pool to handle requests without matching URIs within the service group. Do not set this tag if requests are to be processed by higher-level thread pools, such as the Virtual Host or Web-Connection thread pool.
<service-group>	Thread pool used to handle requests for services with matching URIs.
<name>	Name to identify the base and service-groups.
<uri>	URIs to match requests that will be processed in the specified service-group thread pool. Multiple resources can be provided with comma separators, and each URI must start with '/'.

3.3.2. Web Context Redeployment (Graceful Redeployment)

This guide does not cover web context redeployment. For more information, refer to "Graceful Redeployment" in *JEUS Applications & Deployment Guide*.

3.3.3. Adding Shared Library References

Shared libraries can be added to the 'JEUS_HOME/lib/shared' directory instead of the 'WEB-INF/lib' directory so that they can be shared by multiple applications. When modifying the libraries.xml file and redeploying the module, the updated libraries will be applied without restarting JEUS.

To add a shared library, add the <library> tag and configure the JAR files to share in the 'JEUS_HOME/lib/shared/libraries.xml' file as in the following example. The specification version and the implementation class version can each be configured.

Adding Shared Library References: <libraries.xml>

```
<libraries xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
  <!-- DO NOT MODIFY OUR SYSTEM libraries!!! -->
  <library>
    <library-name>jsf</library-name>
    <specification-version>1.2</specification-version>
    <implementation-version>1.2.15</implementation-version>
    <files dir="jsf_ri_1.2"/>
  </library>
  <library>
    <library-name>jsf</library-name>
    <specification-version>2.2</specification-version>
    <implementation-version>2.2.13</implementation-version>
    <files dir="jsf_ri_2.2"/>
  </library>
  <library>
    <library-name>jsf-jeusport</library-name>
    <files dir=".">
      <include name="jsf-injection-provider.jar"/>
    </library>
  </library>
</libraries>
```

```

    <library-name>jsf-weld-integration</library-name>
    <files dir=".">
        <include name="jsf-weld-integration.jar"/>
</library>
<library>
    <library-name>jstl</library-name>
    <specification-version>1.2</specification-version>
    <implementation-version>1.2</implementation-version>
    <files dir="jstl_1.2"/>
</library>
<library>
    <library-name>jstl</library-name>
    <specification-version>1.2</specification-version>
    <implementation-version>1.2</implementation-version>
    <files dir="jstl_1.2"/>
</library>
<library>
    <library-name>spring-support</library-name>
    <specification-version>4.2.0.RELEASE</specification-version>
    <implementation-version>4.2.0.RELEASE</implementation-version>
    <files dir="spring-support/4.2.0.RELEASE"/>
</library>
<library>
    <library-name>spring-support</library-name>
    <specification-version>4.0.0.RELEASE</specification-version>
    <implementation-version>4.0.0.RELEASE</implementation-version>
    <files dir="spring-support/4.0.0.RELEASE"/>
</library>

<!-- Add user libraries from here -->
<library>
    <library-name>myLibrary</library-name>
    <specification-version>2.0</specification-version>
    <implementation-version>2.1</implementation-version>
    <files dir=".">
        <include name="commons-logging.jar"/>
        <include name="commons-util.jar"/>
    </files>
    <files dir="myLib-2.1"/>
</library>
</libraries>

```

To reference the configured shared libraries, set `<library-ref>` in `jeus-web-dd.xml` as in the following. If the version is not specified, the highest version of the library that is registered in `libraries.xml` will be referenced.

```

<library-ref>
    <library-name>myLibrary</library-name>
    <specification-version>
        <value>2.0</value>
    </specification-version>
</library-ref>

```

Configuring JSF and JSTL Libraries

In the `libraries.xml` file described in [Adding Shared Library References: <libraries.xml>](#), JSF and JSTL are added as shared libraries. Since the Jakarta EE standard includes JSF and JSTL, they are provided in JEUS by default.

The libraries can be provided by JEUS, but this will only allow one each of JSF and JSTL implementation class to be used. Since JSF implementation class has multiple versions besides 2.0, including 1.2, Apache myfaces, and SUN RI, JEUS provides the JSF and JSTL as shared libraries so that multiple implementations can be used.

The implementation classes are classified by their locations and types as in the following.

- JSF or JSTL implementation class in `WEB-INF/lib`.
- Implementation classes in shared library. (This has a lower priority)



Starting from JEUS 7 Fix#2, the Oracle JSF RI is not automatically added to the '`JEUS_HOME/lib/shared`' directory. To use it, explicitly configure the `<library-ref>` setting in `jeus-web-dd.xml`.

Use the following declaration to use the libraries included in the '`JEUS_HOME/lib/shared`' directory.

```
<library-ref>
  <library-name>jsf</library-name>
</library-ref>
```

Also, JEUS has been modified to not automatically add the web listener included in Oracle JSF RI 2.x. It must be configured in the `<listener>` setting of `web.xml`.

```
<listener>
  <listener-class>com.sun.faces.config.ConfigureListener</listener-class>
</listener>
```

To share the same JSF or JSTL libraries in multiple applications, manually configure the `<library>` setting in '`JEUS_HOME/lib/shared/libraries.xml`' and that `<library-ref>` setting in `jeus-web-dd.xml`.

For JSF, the resources provided by WAS must be accessible from managed beans by using annotations such as `@EJB` and `@Resource` and WAS must be able to inject the resources that are specified by the annotations.

JEUS provides '`jsf-injection-provider.jar`' as a shared library to support the injection of JEUS to JSF RI. The shared library exists in the following directory.

```
JEUS_HOME/lib/shared
```

The library is automatically included if `web.xml` contains the `jakarta.faces.webapp.FacesServlet` class.

Configuring Spring Support Library

The Spring support library is added in the libraries.xml file described in [Adding Shared Library References: <libraries.xml>](#).

The Spring Support library is required for applications developed with the Spring framework to use JEUS functions. The following only describes WebSocket-related functions.

When implementing the WebSocket function by using the Spring framework, the spring-websocket module is used. Since this module does not support JEUS WebSocket Handshake, the function does not work normally. To use the function normally, use the Spring support library.

To use the WebSocket function in the Spring support library, set <library-ref> in jeus-web-dd.xml and add a related bean to Spring settings as follows:

```
<beans ...>
  <websocket:handlers>
    <websocket:mapping .../>
    <websocket:handshake-handler ref="handshakeHandler"/>
  </websocket:handlers>

  <bean id="handshakeHandler"
class="org.springframework.web.socket.server.support.DefaultHandshakeHandler">
    <constructor-arg ref="upgradeStrategy"/>
  </bean>
  <bean id="upgradeStrategy" class="jeus.spring.websocket.JeusRequestUpgradeStrategy"/>
</beans>
```

Set as follows when using 4.2.0.RELEASE version library.

```
<beans ...>
  <websocket:handlers>
    <websocket:mapping .../>
    <websocket:handshake-handler ref="handshakeHandler"/>
  </websocket:handlers>

  <bean id="handshakeHandler" class="jeus.spring.websocket.JeusHandshakeHandler"/>
</beans>
```

3.3.4. Configuring Web Contexts for Compatibility

A web context can be configured to resolve compatibility issues with previous versions JEUS or another WAS.

Compatibility with Previous Versions of JEUS

Prior to Servlet 2.4, many standards were not described properly. They were supplemented over the course of servlet 2.4 and 2.5 releases and this naturally resulted in restrictions being created in JEUS. To resolve these issues, JEUS started supporting non-standard operations. Due to such reasons, JEUS

4 web applications may not be compatible with JEUS 6.

JEUS must comply with servlet and JSP standards, and also ensure compatibility with the existing applications, but these requirements can cause conflicts. Usually when the jakarta.servlet standard API or internal operation is modified to comply with the standard, problems may arise if there are applications that were created based on the previous faulty operation. In this situation, the applications must be modified since compliance requirements have a higher precedence. Otherwise, the applications cannot run in other containers, and they may conflict with frameworks that comply with the servlet standards when the applications are upgraded.

If such issues are not identified in advance until the official start of the services, this may require the modification of JEUS operations instead of user applications. To resolve this, JEUS provides properties that can be configured for compatibility with previous versions.

For more information about the compatibility of JSP engines and JEUS properties, refer to [Web Context Options for JSP Backward Compatibility](#) and "Properties for Compatibility" in *JEUS Reference Guide*.

Compatibility with Other WASs

Before Servlet 2.4, web applications were developed using servlets and JSPs directly, but a web framework is now being used. Starting from Java EE 5, JSF and JSTL libraries have been added to the standard web frameworks. Most web frameworks use Tomcat, a reference implementation class for servlet and JSP specifications, for development and testing. This may cause compatibility issues if the web frameworks support non-standard functions.

Compatibility can be configured selectively in jeus-web-dd.xml when there is a compatibility problem. For more information about configuring jeus-web-dd.xml, refer to "Web Engine Properties" in *JEUS Reference Guide*.

3.3.5. Configuring Additional Features

Separate configurations are required to use the following additional functions provided by JEUS.

- **Servlet Mapping for Static Resources**

The servlet standard specifies that the default servlet, which is provided by WAS, is used for a request that is neither a servlet nor JSP. The default servlet is referred to as ResourceServlet.

Map jeus.servlet.servlets.ResourceServlet to the <servlet-mapping> setting in web.xml as in the following.

Servlet Mapping for Static Resources: <web.xml>

```
<servlet-mapping>
  <servlet-name>jeus.servlet.servlets.ResourceServlet</servlet-name>
  <url-pattern>/static/*</url-pattern>
</servlet-mapping>
```

- **Registering mvc:default-servlet-handler in Spring 3.0 or Later**

From JEUS 9 onwards, the mvc:default-servlet-handler does not need to be registered in Spring 3.0 or later. Like Tomcat, the JEUS default servlet is provided in the name of "default".

- **Registering ProgressListener when Servlet 3.0 Multipart Function is Used**

Starting from Servlet 3.0, the multipart-config can be configured per servlet to assign the web container to process multipart/form-data of the POST request. A progressListener must be registered to trigger an event about the processing progress.

1. The **jeus.servlet.engine.multipart.ProgressListener** , which is included in the jeus-servlet.jar file, must be implemented. For information about the interface, refer to the Servlet API document.
2. Add the instance of the previous interface implementation class to the HttpServletRequest attribute using the same name as the interface, **jeus.servlet.engine.multipart.ProgressListener**. Note that this must be done before `getPart()`, `getParts()`, or `getParameter()` is called.

3.3.6. Configuring Web Security

This section describes configuring web security for each application in jeus-web-dd.xml.

Redirect Location Security Configuration

Applications can send the 302 Found response message by using the standard API `jakarta.servlet.http.HttpServletResponse.sendRedirect (String location)`. In this case, the location parameter value will be converted to URL and set to the location header without checking the value. This allows users with malicious intent to use attacks like CRLF injection. To prevent this, applications can implement the `jeus.servlet.security.RedirectStrategy` interface and configure it in `jeus-web-dd.xml`.

The following is an example of configuring the `jeus.servlet.security.RedirectStrategy` interface.

Redirect Location Security Configuration Interface

```
package jeus.servlet.security;

import jakarta.servlet.http.HttpServletRequest;

public interface RedirectStrategy {
    /**
     * Makes the redirect URL.
     *
     * @param location the target URL to redirect to, for example "/login"
     */
    String makeRedirectURL(HttpServletRequest request, String location)
        throws IllegalArgumentException;
}
```

This interface can be implemented as follows:

Redirect Location Security Configuration Implementation

```
package jeus.servlet.security;
import jakarta.servlet.http.HttpServletRequest;

public class RejectCrLfRedirectStrategy implements RedirectStrategy {
    private Pattern CR_OR_LF = Pattern.compile("\\r|\\n");

    @Override
    String String makeRedirectURL(HttpServletRequest request, String location)
        throws IllegalArgumentException {
        if (CR_OR_LF.matcher(location).find()) {
            throw new IllegalArgumentException("invalid characters (CR/LF) in redirect location");
        }
        return makeAbsolute(location);
    }

    private String makeAbsolute(String location) {
        // make code for make absolute path
    }
}
```

Jeus-web-dd.xml can be configured as follows:

Redirect Location Security Configuration: <jeus-web-dd.xml>

```
...
<web-security>
  <redirect-strategy-ref>
    jeus.servlet.security.RejectCrLfRedirectStrategy
  </redirect-strategy-ref>
</web-security>
..
```

The <redirect-strategy-ref> setting is set to an implementation class of the jeus.servlet.security.RedirectStrategy interface. This class must be included in the web application class path. The jeus.servlet.security.RejectCrLfRedirectStrategy class is a default RedirectStrategy provided. If a location that contains CR, LF, or CRLF character is sent to the sendRedirect API, an HTTP 500 error will occur.

In addition, the jeus.servlet.security.RemoveCrLfRedirectStrategy method that replaces CR, LF, or CRLF with an empty string is provided.

3.3.7. Mapping Security Roles

The security roles defined in web.xml must be mapped to system users and groups. This mapping is described in jeus-web-dd.xml.

The following is a sample web.xml file.

Mapping Security Roles: <web.xml>

```
<web-app>
  <security-role>
    <role-name>manager</role-name>
  </security-role>
  <security-role>
    <role-name>developer</role-name>
  </security-role>
  <servlet>
    . . .
  </servlet>
  <security-constraint>
    <web-resource-collection>
      <web-resource-name>
        MyResource
      </web-resource-name>
      <url-pattern>/jsp/*</url-pattern>
      <http-method>GET</http-method>
      <http-method>POST</http-method>
    </web-resource-collection>
    <auth-constraint>
      <role-name>manager</role-name>
    </auth-constraint>
  </security-constraint>
</web-app>
```

In the previous example, two security roles are declared in bold. The third bold element shows how the manager role is used in the <security-constraint> tag.

The previous mapping can be defined in jeus-web-dd.xml as follows:

Mapping Security Roles: <jeus-web-dd.xml>

```
<jeus-web-dd xmlns="http://www.tmaxsoft.com/xml/ns/jeus" version="9">
  . . .
  <role-mapping>
    <role-permission>
      <principal>Peter</principal>
      <role>manager</role>
    </role-permission>
    <role-permission>
      <principal>Linda</principal>
      <role>developer</role>
    </role-permission>
  </role-mapping>
  . . .
</jeus-web-dd>
```

When applications are deployed in a production environment, the manager and developer roles must be bound to a special user in the system. This mapping is defined in the <context><role-mapping> setting. The <role-mapping> tag contains the <role-permission> tag which can be used to define the <principal-to-role> mapping.

The following child tags are also bound. In the previous example, two roles, "manager" and

"developer", are mapped to "Peter" and "Linda", respectively.

Tag	Description
<role>(1, required)	<role-name> value defined in web.xml. The <role-name> is "manager" in the previous example.
<principal>(0 or more)	User name linked to the <role-name> and managed by JEUS. For more information, refer to "Configuring Web Module Security" in <i>JEUS Security Guide</i> .

3.3.8. Mapping Symbolic References

Since the roles are mapped to the actual users, EJB references, resource references, and managed object references must also be mapped to the actual system resources.

The following is an example of symbolic reference mapping.

Mapping Symbolic References: <web.xml>

```
<web-app>
  . . .
  <ejb-ref>
    <ejb-ref-name>ejb/account</ejb-ref-name>
    <ejb-ref-type>Entity</ejb-ref-type>
    <home>com.mycompany.AccountHome</home>
    <remote>com.mycompany.Account</remote>
  </ejb-ref>
  <resource-ref>
    <res-ref-name>jdbc/EmployeeAppDB</res-ref-name>
    <res-type>javax.sql.DataSource</res-type>
    <res-auth>Container</res-auth>
    <res-sharing-scope>Shareable</res-sharing-scope>
  </resource-ref>
  <resource-env-ref>
    <resource-env-ref-name>
      jms/StockQueue
    </resource-env-ref-name>
    <resource-env-ref-type>
      jakarta.jms.Queue
    </resource-env-ref-type>
  </resource-env-ref>
  . . .
</web-app>
```

To register the web application, all symbolic reference names used in the <ejb-ref>, <resource-ref>, <resource-env-ref> settings of web.xml must be mapped to the JNDI names of the corresponding resources. To map them, add the corresponding <ejb-ref>, <res-ref>, and <res-env-ref> tags in jeus-web-dd.xml.

These tags contain the following child tags.

- <jndi-info>

Tag	Description
<ref-name>	Reference name is the same as those defined in web.xml and the servlets.
<export-name>	JNDI export name that is the actual export name of the resource that <ref-name> refers to.

The following example of jeus-web-dd.xml maps the JNDI names to the aforementioned three references.

Mapping Symbolic References: <jeus-web-dd.xml>

```
<jeus-web-dd xmlns="http://www.tmaxsoft.com/xml/ns/jeus" version="9">
  . . .
  <ejb-ref>
    <jndi-info>
      <ref-name>ejb/account</ref-name>
      <export-name>AccountEJB</export-name>
    </jndi-info>
  </ejb-ref>
  <res-ref>
    <jndi-info>
      <ref-name>jdbc/EmployeeAppDB</ref-name>
      <export-name>EmployeeDB</export-name>
    </jndi-info>
  </res-ref>
  <res-env-ref>
    <jndi-info>
      <ref-name>jms/StockQueue</ref-name>
      <export-name>StockQueue</export-name>
    </jndi-info>
  </res-env-ref>
  . . .
</jeus-web-dd>
```

3.4. Monitoring Web Contexts

Monitoring refers to checking information about web contexts. A list of web contexts that are deployed to the web engine and their current states can be checked in the console tool.

Using the console tool

Executing the **application-info** command displays all information about deployed web contexts. For more information about the command, refer to "application-info" in *JEUS Reference Guide*.

```
application-info -type WAR
```

To display information about a specific web context, execute the command with the `-id <application-id>` option as follows:

```
application-info -id <application-id>
```

3.5. Controlling Web Contexts

Controlling a web context refers to changing the state of the web context to modify the service. The console tool can be used to reload, suspend, and resume web contexts.

3.5.1. Reloading Web Contexts

If a deployed web context is changed or the context needs to be reset, the web context can be reloaded instead of being deleted and redeployed. The console tool can be used to reload web contexts.

Using the console tool

Execute the **reload-web-context** command as in the following. For more information about the command, refer to "reload-web-context" in *JEUS Reference Guide*.

```
reload-web-context -ctx <context-name>  
successfully reloaded
```

3.5.2. Monitoring a Thread Pool for Asynchronous Requests in a Web Context

To use thread pools provided by JEUS when web contexts are processing asynchronous requests, the relevant information needs to be configured in the `<async-config>` element of `jeus-web-dd.xml`. After configuring the `<async-config>` element, asynchronous thread pool information can be monitored.

The following is an example of configuring `jeus-web-dd.xml`. The async thread pool information can be checked using commands in the console tool.

`<async-config>` Configuration : `<jeus-web-dd.xml>`

```
...  
<async-config>  
  <dispatch-thread-pool>  
    <min>5</min>  
    <max>20</max>  
  </dispatch-thread-pool>  
  <background-thread-pool>  
    <min>5</min>  
    <max>20</max>  
  </background-thread-pool>  
  <async-timeout-millis>120000</async-timeout-millis>  
</async-config>
```

...

The following describes each configuration tag.

Tag	Description
<dispatch-thread-pool>	Minimum and maximum thread pool sizes that are used when <code>AsyncContext#dispatch</code> is invoked.
<background-thread-pool>	Minimum and the maximum thread pool sizes that are used when <code>AsyncContext#start</code> is invoked.
<async-timeout-millis>	Standard time for when the web container processes timeout during an asynchronous task execution. This value is returned when an application calls <code>AsyncContext#getTimeout</code> without calling <code>AsyncContext#setTimeout</code> . (Default value: 30000, Unit: ms)

Using the console tool

To check the thread information of web contexts using the console tool, execute the `thread-info` command as in the following. For more information about the `thread-info` command, refer to "thread-info" in *JEUS Reference Guide*.

```
thread-info -server <server-name>
```

3.5.3. Suspending Web Contexts

For a specific reason, administrators can suspend a deployed web context that is running, and resume it later. For more information, refer to [Resuming Web Contexts](#).

Using the console tool

Execute the **suspend-web-component** command as in the following. For more information about the command, refer to "suspend-web-component" in *JEUS Reference Guide*.

```
suspend-web-component -server <server-name> -ctx <context> -svl <servlet>
```

or

```
suspend-web-component -cluster <cluster-name> -ctx <context> -svl <servlet>
```

3.5.4. Resuming Web Contexts

Suspended web contexts can be resumed using the console tool.

Using the console tool

Execute the **resume-web-component** command as in the following. For more information about the command, refer to "resume-web-component" in *JEUS Reference Guide*.

```
resume-web-component -server <server-name> -ctx <context> -svl <servlet>
```

or

```
resume-web-component -cluster <cluster-name> -ctx <context> -svl <servlet>
```

3.6. Tuning Web Contexts

Consider the following when tuning web context configurations (jeus-web-dd.xml) for optimal performance.

- Increase the user log size and refrain from using "stdout".
- Disable the JSP engine when not in use.
- Always disable the <enable-reload> and <check-on-demand> options. These options are intended for a development environment where classes are frequently modified.
- If possible, use caching. Set as many static content directories in the caching tags as possible. Use a large value for the <max-idle-time> and <max-cache-memory> settings of the cache. They can be set to infinity.

3.7. Asynchronous Processing Programming

This section describes what application developers must understand about asynchronous processing that has been added starting from servlet 3.0.

3.7.1. Precautions for Using Servlet Standards

Application developers must directly handle the threads due to the nature of asynchronous processing programming, and this may trigger the following errors.

- [Creating Too Many Threads](#)
- [Sharing Unsafe Objects with Other Threads](#)

- [Using jakarta.servlet.RequestDispatcher from Asynchronous Threads](#)

Some of these errors cannot be detected during testing, but they may occur when too much load is incurred in the production environment. **Note that the application developers, and not JEUS, are responsible for preventing these errors.**



This section is written based on the servlet standard. (It is applicable to other products as well, besides JEUS.)

Creating Too Many Threads

Creating too many threads reduces the performance of the JVM or causes an `OutOfMemoryError` to occur. Following is an example of a faulty asynchronous processing programing that creates too many threads.

Example of Faulty Asynchronous Processing Program (1) : `<WrongAsyncServlet1.java>`

```
import jakarta.servlet.AsyncContext;
import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import java.io.IOException;

@WebServlet(urlPatterns = "/WrongAsyncServlet1", asyncSupported = true)
public class WrongAsyncServlet1 extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException {
        AsyncContext asyncContext = req.startAsync();
        Thread thread = new Thread(new TestRunnable(asyncContext));
        thread.start();
    }

    private class TestRunnable implements Runnable {
        private final AsyncContext asyncContext;

        public TestRunnable(AsyncContext asyncContext) {
            this.asyncContext = asyncContext;
        }

        @Override
        public void run() {
        }
    }
}
```

There can be more than thousands of concurrent asynchronous requests, which creates thousands of concurrent threads. This can greatly reduce the performance of the JVM or cause an "OutOfMemoryError: unable to create new native thread" to occur. Therefore, it is recommended to use the `jakarta.servlet.AsyncContext#start(Runnable)` method or a thread pool that can be

directly managed.

```
AsyncContext asyncContext = req.startAsync();
asyncContext.start(new TestRunnable(asyncContext));
```

Sharing Unsafe Objects with Other Threads

Sharing unsafe objects that are not supposed to be shared with other threads can reduce the performance, result in a deadlock, or violate data integrity.

The following is an example of a faulty asynchronous program that allows different threads to share unsafe objects.

Example of Faulty Asynchronous Processing Program (2) : <WrongAsyncServlet2.java>

```
import jakarta.servlet.AsyncContext;
import jakarta.servlet.ServletException;
import jakarta.servlet.ServletInputStream;
import jakarta.servlet.ServletOutputStream;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import java.io.IOException;

@WebServlet(urlPatterns = "/WrongAsyncServlet2", asyncSupported = true)
public class WrongAsyncServlet2 extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException {
        AsyncContext asyncContext = req.startAsync();
        asyncContext.start(new TestRunnable(asyncContext));

        byte[] buffer = new byte[1024];
        ServletInputStream inputStream = req.getInputStream();
        inputStream.read(buffer);

        ServletOutputStream outputStream = resp.getOutputStream();
        outputStream.write(buffer);
    }

    private class TestRunnable implements Runnable {
        private final AsyncContext asyncContext;

        public TestRunnable(AsyncContext asyncContext) {
            this.asyncContext = asyncContext;
        }

        @Override
        public void run() {
            byte[] buffer = new byte[1024];
            try {
                ServletInputStream inputStream = asyncContext.getRequest().getInputStream();
                inputStream.read(buffer);
            } catch (IOException e) {
```



```

}

private class TestRunnable implements Runnable {
    private final AsyncContext asyncContext;

    public TestRunnable(AsyncContext asyncContext) {
        this.asyncContext = asyncContext;
    }

    @Override
    public void run() {
        // ..... do something
        RequestDispatcher requestDispatcher =
            asyncContext.getRequest().getRequestDispatcher("/views/report.jsp");
        try {
            requestDispatcher.forward(asyncContext.getRequest(),
                asyncContext.getResponse());
        } catch (ServletException e) {
            // log error
        } catch (IOException e) {
            // log error
        }
    }
}
}
}

```

The `jakarta.servlet.AsyncContext` class provides a way of dispatching from the asynchronous processing thread to another servlet or JSP.

```
asyncContext.dispatch("/views/report.jsp");
```

3.7.2. Other Considerations

There are other precautions in addition to those for the servlet standard.

- **When `java.util.concurrent.RejectedExecutionException` occurs in the thread when `AsyncContext.start(Runnable)` is invoked**

`AsyncContext.start` represents a new task. A thread is needed to process the task, and a thread pool is used because there are limited thread resources. If there is no available thread in the thread pool that can process the new task, the task will be accumulated in the queue.

When tasks that require a long time to process cause delays in processing, other tasks are accumulated in the queue. Since the queue has a limited size (Maximum: 4096), it may not be able to add a new task when it is full. This will cause **`java.util.concurrent.RejectedExecutionException`** to occur. Hence, application developers must consider this issue when programming asynchronous processing.

- **When the listener, which communicates with a Web server such as WebtoB and Apache, performs asynchronous processing**

A WebtoB or AJP13 listener uses a limited number of TCP connections. If multiple HTTP requests are delivered to WAS from the web server simultaneously, the service may hang. This method was designed for the original synchronous processing model. When the web application server receives an HTTP request from the web server, it transfers the ownership of the TCP connection to the servlet. It is assumed that the servlet will return the HTTP response immediately after processing and return the TCP connection.

But it cannot be assumed that the servlet will immediately return the HTTP response in asynchronous processing. If all the TCP connections with the web server are used for asynchronous processing, regular HTTP requests cannot be processed temporarily.

Consider the following solutions to address this problem.

- Use reverse proxy through URL mapping for asynchronous requests. The reverse proxy targets an HTTP listener.
- Specify sufficient number of TCP connections to the web server to avoid any asynchronous processing.



It is recommended to use WebtoB 4.1.6 or later.

4. Thread Pool

This chapter describes the structure and configuration of thread pools.

4.1. Overview

In a web engine, thread pools are required to handle incoming requests.

A thread pool can be configured at different levels of the web application architecture, such as Web Connection, Virtual Host, and Context. Each level has its own priority level, allowing requests to be processed in separate thread pools.

To utilize all three thread pools, you need to return to the selector. Here, you need to configure the AJP, HTTP, and TCP listeners for server listeners and the use-nio setting for the WebtoB connector as true. If the use-nio setting for the WebtoB and Tmax connectors are set to false, requests are handled only in the Web Connection level thread pool.

The Context and Virtual Host level thread pools are optional, while the Web Connection level thread pool is a mandatory setting. The Context thread pool can be configured in jeus-web-dd.xml, and Virtual Host and Web Connection level thread pools are configured under <virtual-host> and <web-connections> respectively in domain.xml.



In versions prior to JEUS 21, each Web Connection in the web engine had its own dedicated thread pool. In JEUS 21 Fix#0, AJP, HTTP, and TCP Web Connections with server listeners were processed independently in different thread pools, which were specifically allocated at the Web Container, Virtual Host, and Context levels. In addition, the WebtoB and Tmax connectors had their own dedicated thread pools for request processing.

4.2. Basic Structure

This section provides an overview of the generation, expiration, and utilization of each level of thread pools (Web Connection, Virtual Host, and Context).

The Web Connection level thread pool is created during the startup of the web engine or when a web connection is dynamically added. It serves as the default thread pool for processing incoming requests if neither the Context nor Virtual Host level thread pools exist.

The Virtual Host level thread pool is generated for each virtual host during the web engine's startup or when a virtual host is dynamically added, provided that there is a configuration for the thread pool in place. Note that the system host, which is responsible for system applications and the default host do not generate thread pools.

The Context level thread pool is created when an application is deployed with the thread pool configuration. At the Context level, unlike the other levels, multiple thread pools can be created. A

Context level thread pool can configure one base thread pool and one or more service group thread pools. A service group thread pool can have multiple comma-separated URIs for processing requests targeted at specific URIs, allowing efficient resources utilization based on the incoming request load for each specific URI. If a specific URI does not have a mapped service group thread pool, the requests for that URI will be directed to the base thread pool, provided that this is configured. If neither the base thread pool is available, the Virtual Host or the Web Connection level, which is the default thread pool, will handle those requests. When the application is undeployed, all the associated thread pools are expired.

4.3. Configuration

This section describes how to configure thread pools of each level.

The Web Connection level thread pool must be configured in the domain.xml file. If it is not configured, JEUS cannot successfully start up.

Web Connection level thread pool : <domain.xml>

```
<web-connections>
  <http-listener>
    <thread-pool>
      <min>10</min>
      <max>40</max>
      <max-idle-time>300000</max-idle-time>
      <max-queue>30</max-queue>
    </thread-pool>
  </http-listener>
</web-connections>
```

As shown in the above example, a Web Connection thread pool can be configured for each listener or connector defined under <web-connections> in the xml file. The Web Connection thread pool is the default thread pool if neither the Context nor Virtual Host level thread pool is specified.

In addition, a thread pool must be configured when a web connection is dynamically added from the jeusadmin console. For more information, see [Adding the AJP Listener](#).

The Virtual Host level thread pool is configured in domain.xml. It is optional, and if not configured, the Web Connection level thread pool handles the requests.

Virtual Host level thread pool: <domain.xml>

```
<web-engine>
  <virtual-host>
    <virtual-host-name>testHost</virtual-host-name>
    <host-name>192.1.1.1</host-name>
    <thread-pool>
      <min>10</min>
      <max>20</max>
      <max-idle-time>300000</max-idle-time>
      <max-queue>30</max-queue>
    </thread-pool>
  </virtual-host>
```

```
</web-engin>
```

As shown in the above example, one Virtual Host thread pool can be configured under `<virtual-host>`. If no Context level thread pool is available, requests will be directed to the Virtual Host thread pool, and if neither the Virtual Host thread pool is configured, all incoming requests will be directed to the Web Connection thread pool.

The thread pool can also be configured when dynamically adding a virtual host from the jeusadmin console. For more information, see [Configuring Virtual Hosts](#).

The Context level thread pool is configured in `jeus-web-dd.xml`. It is optional, and if not configured, the Virtual Host level thread pool handles the requests.

Context level thread pool: `<jeus-web-dd.xml>`

```
<jeus-web-dd>
  <thread-pool>
    <base>
      <name>baseThreadPool</name>
      <min>30</min>
      <max>50</max>
      <max-idle-time>300000</max-idle-time>
      <max-queue>30</max-queue>
    </base>
    <service-group>
      <name>service1</name>
      <uri>/index.html,/index.jsp</uri>
      <min>10</min>
      <max>20</max>
      <max-idle-time>300000</max-idle-time>
      <max-queue>30</max-queue>
    </service-group>
    <service-group>
      <name>service2</name>
      <uri>/mostRequestedResource</uri>
      <min>10</min>
      <max>20</max>
      <max-idle-time>300000</max-idle-time>
      <max-queue>30</max-queue>
    </service-group>
  </thread-pool>
</jeus-web-dd>
```

As shown in the above example for the Context level thread pool, one base thread pool and one or more service-group thread pools can be configured under `<jeus-web-dd>`. To delimit multiple context thread pools, a unique name must be specified for each thread pool. If the requested URI is mapped to the URI configured for one of the service-groups, that service-group thread pool handles the request. Otherwise, the request is directed to the base thread pool. If neither the base thread pool is configured, the request is re-directed to the Virtual Host level thread pool.

When deploying an application through jeusadmin, a single Context thread pool can be added. For more information, refer to "deploy-application" in *JEUS Reference Guide*.

- **Common Settings for Thread Pool**

- Worker thread pools have common setting items.
- The following describes each setting item.

Item	Description
Min	Minimum number of worker threads to be managed in the pool.
Max	Maximum number of worker threads to be managed in the pool.
Maximum Idle Time	Maximum amount of time during which a thread remains unused before being removed from the pool.
Max Queue	Maximum number of threads to be queued.
Thread State Notify	Notification for a blocked worker thread. Each thread pool contains the Thread State Notify setting which defines the action to be taken when a failure occurs. For more information about this setting, refer to Configuring Automatic Thread Pool State Management (Thread State Notify) .

4.4. Configuring Automatic Thread Pool State Management (Thread State Notify)

Thread State Notify defines the minimum number of worker threads under specific failed conditions to send an email notification or to recommend a server restart.

The following describes the process of configuring thread state notification in the console tool. When the configuration is complete, you can receive the desired notification for a thread state change in the thread pool.

Once automatic thread pool management is set up, worker threads are managed based on the value specified in '**Max Thread Active Time**'. Starting from the point in time when the request processing is initiated, all threads that exceed the specified Max Thread Active Time will be treated and managed as blocked threads.

If the email notifier is used, refer to the "SMTP Handler" section in "Logging Configuration" in JEUS Server Guide.

```
<web-engine>
  <web-connections>
    <http-listener>
      <thread-pool>
        ...
        <thread-state-notify>
          <max-thread-active-time>10000</max-thread-active-time>
          <interrupt-thread>false</interrupt-thread>
          <active-timeout-notification>false</active-timeout-notification>
          <notify-threshold-ratio>0.1</notify-threshold-ratio>
          <notify-subject>Notify-Subject</notify-subject>
```

```

        <restart-threshold-ratio>0.1</restart-threshold-ratio>
        <restart-subject>Restart-Subject</restart-subject>
    </thread-state-notify>
</thread-pool>
</http-listener>
</web-connections>
</web-engine>

```

The following describes each setting item.

Item	Description
Max Thread Active Time	<p>Maximum amount of time during which a worker thread remains active before being blocked.</p> <p>This duration begins upon the start of the requested service by the worker thread (servlet being executed). It expires when the time period is exceeded or the thread is freed up and returns to the thread pool (thread is no longer blocked). If set to 0 or a negative value, this setting is ignored.</p>
Notify Threshold Ratio	<p>Maximum ratio of blocked threads in the total number of threads. If this threshold is reached, it is determined that an error condition has occurred, and thereby an email notification is triggered and sent via the email notifier.</p> <p>Set the value to a decimal point that is smaller than 1. If set to 0 or a negative value, this setting is ignored.</p>
Restart Threshold Ratio	<p>Maximum ratio of blocked threads in the total number of threads. If this threshold is reached, it is determined that an error condition has occurred, and thereby an email notification is triggered and sent via the email notifier. After that, a message recommending a server restart is logged in the server log.</p> <p>Set the value to a decimal point that is smaller than 1. If set to 0 or a negative value, this setting is ignored.</p>
Notify Subject	Used for sending a warning email.
Restart Subject	Used for an engine logging or notification email.
Interrupt Thread	<p>Option to trigger an interruption when a worker thread continues to execute a request after expiration of the Max Thread Active Time, which is the active timeout.</p> <ul style="list-style-type: none"> ◦ True: Interruption is triggered. ◦ False: Interruption is not triggered. (Default)
Active Timeout Notification	<p>Option to send an email notification for active timeout.</p> <ul style="list-style-type: none"> ◦ True: Email is sent. ◦ False: Email is not sent. (Default)



To configure an SMTP handler for the logger to receive email notifications, specify

the name as "jeus.servlet.threadpool.notify" for separate management.

4.5. Rules

The following rules are applied when configuring a thread pool.

- Using the Context and Virtual Host level thread pools requires identification between context and host by reading the request, prior to the thread pool execution.

The corresponding web connections include the HTTP, AJP13, and TCP listeners, which can reference server listeners, and the WebtoB connector with the use-nio setting as true. If use-nio for the WebtoB and Tmax connectors is set to false, these connectors use only the Web Connection level thread pool.

- To be Noted when configuring WebtoB

Configuring a WebtoB connector requires attention to thread pool configurations depending on the use-nio setting. If use-nio is true, the number of connections is no longer related to threads, and therefore an appropriate value can be set according to the tuning. If use-nio is false, on the other hand, the minimum and maximum values of connection-count must match those of thread-pool. Otherwise, server startup fails.

- When configuring service group thread pools, ensure that duplicate URIs are not used in different service groups. A URI must start with '/'. According to the servlet specifications, a URI corresponds to the servlet path and path information after the context path.
- A URI path can include subdirectories. For example, if you specify /myServlet/ as the URI for a service group, incoming requests sent to /testContext/myServlet/servlet1 and /testContext/myServlet/servlet2 are both processed by the same service group with the /myServlet/ URI setting.
- If multiple service groups are configured, choose the most matched URI.

For example, The URI for Service Group1 is specified as /myServlet/ and Service Group2 as /myServlet/servlet2, incoming requests sent to /testContext/myServlet/servlet1 are handled in Service Group1, and those sent to /testContext/myServlet/servlet2 are processed in Service Group2.

- Determining which thread pools will handle which requests is simple:
 1. Determine whether a web connection that has received a request can use either the Context or Virtual Host thread pool. If neither of those thread pools is available, the request will be handled by the thread pool of the web connection itself.
 2. Check whether the context that matches the request contains thread pools. If service group thread pools have been configured for the context, check the URI. Select the service group with the matching URI, if any; otherwise, proceed down to another thread pool to find the matching URI.
 3. If the base thread pool has been configured in the context, use it.

4. If the virtual host that matches the request contains the thread pool, use it.
5. If no thread pools are available from 1 to 4, use the Web Connection level thread pool.

5. JSP Engine

This chapter describes the concepts, functions, and configurations of the JSP engine.

5.1. Overview

The JSP engine is embedded in web engines. It exists in each web context. This section does not explain JSP standards. Refer to the JSP standards for more information about creating a JSP engine.

When web clients request JSP pages, the JSP engine finds the pages and converts them to servlets. During the servlets conversion process, the engine creates Java files and SMAP files, and compiles the Java files to create class files that will be used for the services.

If JSP pre-compilation is not used, JSP files are compiled the first time the request is received, so when the first request happens, the response might take more time because the access to the OS file system occurs frequently.

In an environment where network-attached storage (NAS) is used, if there are many files that require compilation because of tags and JSP include relations, many requests will be sent to the NAS, which delays the response. Depending on the NAS driver operation, a JSP file with a size of 0 might be read without generating a `java.io.IOException`. Jasper reads files using the JVM file I/O API, so parsing of the JSP file will fail.

JSP in the Web Context

The JSP file exists under the root of the web context. It can be packaged by creating a separate directory, and can be packaged under the `META-INF/resources/` directory defined from Servlet 3.0 onward.

The JSP file can be found even if there is a `META-INF/resources/` directory in the `*.jar` library files under `WEB-INF/lib/`. However, the JSP files under the `WEB-INF` directory are not serviced. Due to security reasons, the web client cannot access the files under `WEB-INF/`. For more information about the internal structure of the web context, refer to [Web Context Structure \(WAR File Structure\)](#).



For more information about `META-INF/resources/`, refer to the resource-related API description in [Jakarta ServletContext API Document](#).



Internal testing shows that the `javac` library provided by the Oracle JDK 6 is not thread-safe, so the property `jeus.servlet.jsp.compile-java-source-concurrently` is provided (default value: `false`). When the request threads compile `.java` files, they set a lock at the JVM scope. If the JDK-provided `javac` library is thread-safe, you can set the property to `true` in `jeus-web-dd.xml` as follows. In this case, you need to test whether JSP compiling is successful in a multi-thread setup, not in a single-thread configuration.

```
<properties>
  <property>
    <key>jeus.servlet.jsp.compile-java-source-concurrently</key>
    <value>true</value>
  </property>
</properties>
```

5.2. Apache Tomcat Jasper

JEUS has always included Jasper, the Tomcat JSP parser, but changed the package name and included it in jeus.jar. JEUS 9 uses the Tomcat 8 based Jasper, and the package is named org.apache.jasper, but it is packaged as a separate library jasper.jar. The library is located in the following path.

```
$JEUS_HOME/lib/system/jasper.jar
```

When using Jasper, consider the following.

- The jasper.jar library is modified for JEUS, so it must not be overwritten with the Tomcat library. If it is, the server will not start.
- When an application uses Tomcat's jasper.jar in WEB-INF/lib, the <webinf-first> option in jeus-web-dd.xml must be set to 'true'. Otherwise, the jasper.jar classes from JEUS will be used, which can lead to unintended results.

Differences between Tomcat Jasper and JEUS Jasper

As explained above, JEUS modifies parts of the version of Jasper that are provided by Tomcat, so consider the following.

- JEUS provides functions including in-memory JSP compilation, which Tomcat Jasper does not provide.
- JEUS does not use the tag handler pool or the page context pool.

64 KB Method Size Limitation when Compiling Java Source

If there is too much content in a single .jsp file when creating JSP, the size of a method in the file may exceed 64KB. In this case, the .java file cannot be compiled by a Java compiler because it violates the Java Language Standards. The 64KB limit can be avoided if most of the content is in the string format, such as SQL statements or HTML tags.

Configure the web.xml file as follows:

```
<servlet>
  <servlet-name>jeus.servlet.servlets.JspServlet</servlet-name>
  <servlet-class>jeus.servlet.servlets.JspServlet</servlet-class>
```

```
<init-param>
  <param-name>genStringAsCharArray</param-name>
  <param-value>>true</param-value>
</init-param>
</servlet>
<servlet-mapping>
  <servlet-name>jeus.servlet.servlets.JspServlet</servlet-name>
  <url-pattern>*.jsp</url-pattern>
</servlet-mapping>
```

If a file with too many Java codes fails to be compiled, separate responses using the `<jsp:include>` element. The following is an example of using the `<jsp:include>` element.

```
<body>
  Template Page<br/>
  <jsp:include page="module_one.jsp" />
  <jsp:include page="module_two.jsp" />
</body>
```

5.3. JSP Engine Functions

The JSP engine provides graceful reloading, precompiling, as well as JSP compiling and running at the memory.

5.3.1. JSP Graceful Reloading

JSP files are usually used for views rather than for logic, so users often want to change the JSP files while still running the service, so web applications that use JSP are often deployed as a directory instead of a WAR.

- When deployed as a directory

If a JSP file is changed in the directory where the application is deployed as a directory, the JSP engine compares the last modified date of the existing compiled class files with the modified date of the original files, and then compiles the files.

When this happens, if the configuration `<check-included-jspfile>` is 'true', even if the requested JSP file has not changed, the JSP files or tag files (.tld) that are included in the JSP file will be checked to recompile the JSP file.

The JSP engine supports graceful reloading. Therefore, even if recompilation occurs from changing the JSP file, then continuous service can be supported with the existing compilation.

- When deployed as a WAR

If applications are deployed in WAR, all of the files will be repackaged to be redeployed, so the risk is high in the perspective of service operation. If a large amount of files are changed, consider graceful redeployment, which is provided Starting from JEUS. For more information

about graceful redeployment, refer to [Web Context Redeployment \(Graceful Redeployment\)](#).



To reload the JSP, a class loader must be created per JSP and the class loader must be replaced when reloading. The references for the native library is managed by the JVM in the class loader. When the class loader instance goes through the garbage collection, the reference gets cleaned up. Thus, if the JVM GC does not occur when the JSP class loader needs to be replaced, JSP reloading will fail. Since the JVM GC cannot be controlled from JEUS and this issue occurs due to the internal structure of the JVM, it cannot be resolved. Therefore, it is recommended to load a native library only once at server startup. When a native library is modified, JEUS cannot guarantee a successful reloading of the library within the JVM.

5.3.2. JSP Precompilation

JSP precompilation compiles JSP pages of a web context in advance. This can be done using the **appcompiler** script or the **precompile-jsp** (jpsc) command in jeusadmin. The appcompiler can precompile JSP files even in an offline state and precompile-jsp can precompile the modules that are deployed on JEUS. Precompilation can help improve performance when the JSP is called for the first time. For more information about appcompiler or the precompile-jsp command in the console tool, refer to "appcompiler" and "precompile-jsp" in JEUS Reference Guide.

5.3.3. JSP Compilation and Execution in Memory

When multiple web engines execute a service using the same source file from NAS, a delayed service or errors can occur due to OS file system access operations that are needed at JSP compilation time. In order to address this issue, the JSP precompilation and JSP graceful reloading functions are provided, but a more fundamental resolution of compiling and executing in memory is also provided.

The JSP engine saves the .java and .class files, the results of JSP compilation, in memory. Therefore, the engine does not access the file system except for when reading metadata and content from the .jsp file. This reduces the number of file accesses, and thereby reducing any service delays that can occur at JSP compilation time.

The .java and .class files are actually needed for other reasons. They are used in the file system by using a background thread that every JSP engine has, instead of using a request processing thread. Since the thread handles requests in the order they are received, the chances of bottlenecks happening in the file system due to file I/O requests are low, but the .smap files are not used. The previous JSP compilation method should be used if the .smap files need to be used.

The JSP engine uses the in-memory compilation method by default. To use the previous method, configure it in jeus-web-dd.xml. For detailed information, refer to [Configuring jeus-web-dd.xml](#).

5.4. Configuring JSP Engines

The JSP engine can be configured in domain.xml or in each application's jeus-web-dd.xml file.

5.4.1. Web Engine-Level Configuration

The following is an example of configuring the JSP engine of a web engine by using domain.xml.

```
<web-engine>
  <jsp-engine>
    <check-included-jspfile>true</check-included-jspfile>
    <keep-generated>true</keep-generated>
    <use-in-memory-compilation>true</use-in-memory-compilation>
    <graceful-jsp-reloading>false</graceful-jsp-reloading>
    <graceful-jsp-reloading-period>30000</graceful-jsp-reloading-period>
    <jsp-work-dir>...</jsp-work-dir>
    <java-compiler>java6</java-compiler>
    <compile-output-dir>...</compile-output-dir>
    <compile-option>...</compile-option>
  </jsp-engine>
</web-engine>
```

The following describes each setting item.

Item	Description
Jsp Work Dir	<p>Root directory where Java source files created by JSP are saved. The files are not directly created in the root directory. Directories are created using the domain name, server name, and web application name. The files are created in these directories. In other words, class files are not shared between different web engines.</p> <p>If 'Compile Output Dir' is not set, class files are created in the same path. By default, the class files are created in the following location.</p> <ul style="list-style-type: none">◦ EAR application <div style="border: 1px solid #ccc; padding: 5px; margin: 5px 0;">INTERNALGENERATED_HOME/ear1/web1/___jsp_work</div> <ul style="list-style-type: none">◦ Standalone web module <div style="border: 1px solid #ccc; padding: 5px; margin: 5px 0;">INTERNALGENERATED_HOME/web1/___jsp_work</div>
Java Compiler	<p>Command that executes the Java compiler that compiles JSP Java source to servlet classes.</p> <p>Since the default value is the most efficient, it is not recommended to change this setting.</p>

Item	Description
Compile Output Dir	Location of the class files that are created by compiling the Java files, which were created by the JSP parser. The class files are used to execute the services. If not set, they are created in the same directory as the Java files.
Compile Option	Servlet compiler
Check Included Jspfile	The JSP engine checks whether a requested JSP page has been modified. As well, it checks whether all JSP files and tag files that are included with the <code><%@ include file="xxx.jsp" %></code> directive are modified and then performs compilation. If set to 'false', then only checks whether the requested JSP page has been modified and then performs compilation.
Keep Generated	Option to keep the Java and SMAP files created by the JSP parser. This is useful for debugging. If set to 'false', files will be deleted after being created. For performance reasons, it is not recommended to change this setting.
Graceful Jsp Reloading	Creates a JSP page instance every specified period if a JSP file is changed.
Graceful Jsp Reloading Period	Period in which Graceful Jsp Reloading is run. (Unit: ms)
Use In Memory Compilation	Function used to create, compile, and execute .java and .class files when JSP files that are running need to be recompiled. Refer to JSP Compilation and Execution in Memory .

5.4.2. Configuring jeus-web-dd.xml

The JSP engine can be configured in jeus-web-dd.xml.

Web Context Configuration File: <jeus-web-dd.xml>

```
<jeus-web-dd xmlns="http://www.tmaxsoft.com/xml/ns/jeus" version="9">
  <enable-jsp>true</enable-jsp>
  <jsp-engine>
    <jsp-work-dir>/home/user1/myjspwork/</jsp-work-dir>
    <java-compiler>java6</java-compiler>
    <compile-option>-g:none -verbose</compile-option>
    <compile-encoding>UTF-8</compile-encoding>
    <check-included-jspfile>true</check-included-jspfile>
    <keep-generated>true</keep-generated>
    <use-in-memory-compilation>true</use-in-memory-compilation>
  </jsp-engine>
</jeus-web-dd>
```

The following describes each configuration tag.

Tag	Description
<enable-jsp>	Option to use JSP. If set to false, JSP is not used and JSP file is executed through the default JEUS servlet.
<jsp-engine>	Settings for JSP pages that belong to the web context. If the JSP engine is configured in domain.xml, this setting has a higher priority. For more information about configuring the JSP engine in domain.xml, refer to Web Engine-Level Configuration .

5.4.3. Web Context Options for JSP Backward Compatibility

In JEUS 4 and 5, non-standard functions are provided for applications that were developed before servlet 2.3, and the syntax does not conform to the JSP syntax. Starting from JEUS 6, syntax checking is more strict and the JSP parser has been replaced with a Jasper-based JSP parser to support JSP 2.1, but the existing JSP parser is also supported for backward compatibility with existing JSP.

Errors may occur when deploying web modules that ran successfully in JEUS 4 and 5. To use the current specifications such as JSP 2.1, check the error messages that are generated when compiling the JSP, and modify the module accordingly.

To use the existing modules without using the current specifications, add the following configurations to jeus-web-dd.xml. This will allow using the JSP parser, which is compatible with JEUS 4 and 5, in the web context.

Web Context Options for JSP Backward Compatibility: <jeus-web-dd.xml>

```
<properties>
  <property>
    <key>jeus.servlet.jsp.modern</key>
    <value>>false</value>
  </property>
</properties>
```

The options in jeus-web-dd.xml are applied to the web context, but it is also possible to apply them to the web engine or the virtual host. To apply the options at the engine level, set the following VM option. Refer to "Adding Servers" in *JEUS Server Guide*.

```
-Djeus.servlet.jsp.modern=false
```

VM options can be replaced at the virtual host or web context level, and the settings in jeus-web-dd.xml are applied last. If the option is not set in jeus-web-dd.xml, the default configuration of the parent will be applied.



Use this option only for backward compatibility. For new application development, it is recommended to use the current standards. **Note that this compatibility option may no longer be provided from the next version or a**

new fix release onwards. For information about other options, refer to "Web Engine Properties" in *JEUS Reference Guide*.

5.4.4. Configuring jarscan.properties

When deploying an application, .tld files are scanned from all .jar libraries located in classpath. If there is a library that does not include a .tld file among .jar libraries used by the application, add the library to the jarscan.properties file in JEUS_HOME/domains/<domain-name>/config to avoid scanning and reduce the deploy time.

Since the JEUS system library is included in classpath when deploying an application, a system library that does not include a .tld file is set in the jarscan.properties file by default.

Setting for Avoiding Unnecessary tld Scanning: <jarscan.properties>

```
jeus.servlet.jsp.JarScanFilterImpl.skipSet= \  
  lib/system/EJML-core-0.29.jar, lib/system/EJML-dense64-0.29.jar, \  
  ...  
  lib/shared/wsit-2.3/webservices-rt-2.3.1.jar, lib/shared/wsit-2.3/webservices-tools-2.3.1.jar, \  
  WEB-INF/lib/noTldLib.jar
```

6. Virtual host

This section describes the purpose and rules of virtual hosts and how ServletContext objects are related to virtual hosts.

6.1. Overview

Virtual hosts allow different web applications to be mapped with the same URL based on the Internet domain name. In other words, two or more domain names (for example, "www1.foo.com" and "www2.foo.com") can exist on a single web engine and use different web contexts.



In the context of web engines, web contexts and web applications are the same.

6.2. Web Engine and Virtual Hosts

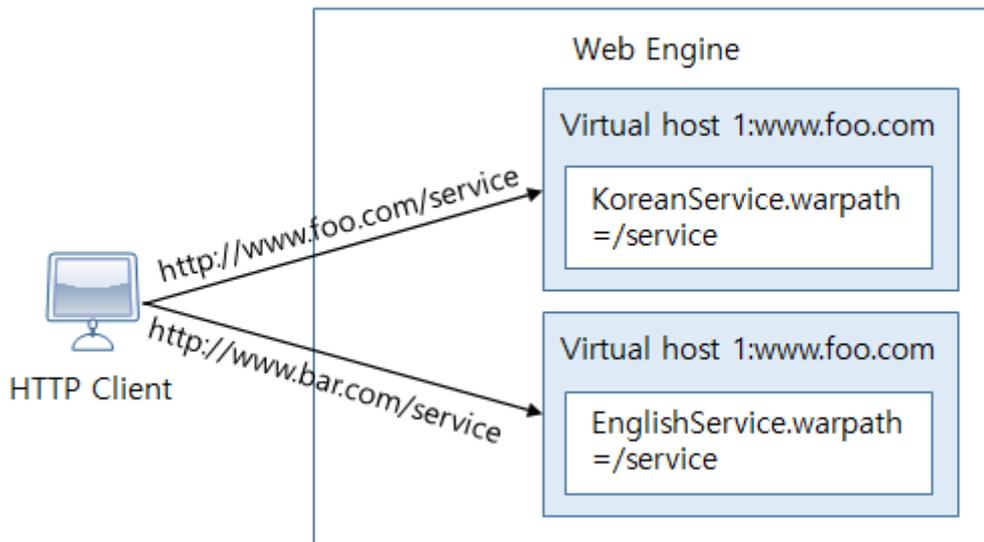
This section describes the purpose and rules of virtual hosts and how ServletContext objects are related to virtual hosts.

Purpose

Virtual hosts allows mapping of different web applications to the same URL based on the domain name mapped to the virtual host. This allows service providers to provide multiple web sites on a single web engine. This is similar to the function that provides a virtual host through the Host header of HTTP 1.1.

A virtual host is a group of web contexts that can be configured on a web engine. Refer to [Web Engine Components](#) for more information about where the virtual host is positioned as a web engine component.

The following diagram shows how virtual hosts are used depending on their purposes.



Virtual Host Usage Pattern

In the previous example, it appears that the same context path (`/service`) is being accessed through two different addresses. There is one server, but from the perspective of the clients, there appear to be two servers, "www.foo.com" and "www.bar.com".

The service provider can provide different services with the same `/service` address pattern. In the previous example, "www.foo.com" provides services in Korean and "www.bar.com" provides services in English.

Rules

The following rules apply when creating virtual hosts.

- A virtual host must have its own name.

The name is used to refer to the virtual host in the configuration file and must be unique within the web engine.

- One or more domain names and IP addresses can be mapped to a single virtual host.

JEUS calls this a host name. Note that the same host name cannot be mapped to different virtual hosts.

- Web contexts with the same name cannot be deployed to different virtual hosts.



The Servlet standard states that a same web context cannot be shared by different virtual hosts.

- Different web contexts with the same path can be deployed on different virtual hosts, but those with the same path cannot exist on the same virtual host.

The web context name is the application or module name that is defined in the Jakarta EE standard. The path is the context root or the context path that is defined in the web application.

The web context names are managed for the purpose of deployment, and the web engine manages the context paths.

JEUS web engines have a default virtual host. When deploying a web context, if a virtual host is not set, the web context will be deployed to the default virtual host. The name of the default virtual host is 'DEFAULT_HOME'. This is a reserved word that cannot be used as another virtual host name.

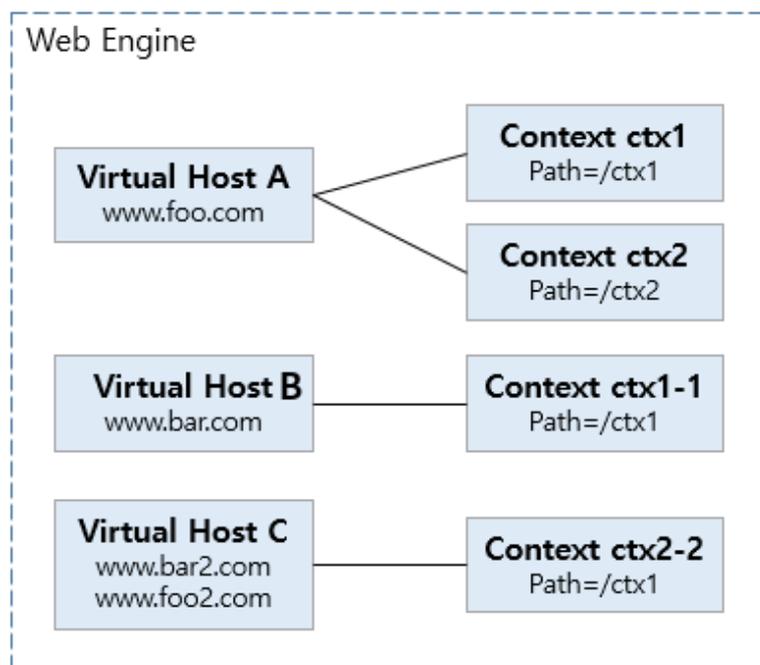
6.2.1. ServletContext Object and Virtual Hosts

In the Servlet API, the `jakarta.servlet.ServletContext.getContext(String contextPath)` method returns the `ServletContext` object that is located in the "contextPath". This method returns the `ServletContext` that exists on the virtual host to which the `ServletContext` belongs. If the object does not exist on the virtual host, the default virtual host will be searched for.

6.3. Web Context Requests to Virtual Host

This section describes how to map URLs to web contexts on a virtual host.

The following example shows the relationships between a web engine, virtual hosts, and web contexts.



Relationships between a Web Engine, Virtual Hosts, and Web Contexts

6.3.1. URL Matching Examples

The following describes the virtual hosts and web contexts that are mapped to URLs based on the previous figure, [Relationships between a Web Engine, Virtual Hosts, and Web Contexts](#).

•

`http://www.foo.com/ctx1/test.jsp`

Matching virtual host	A
-----------------------	---

Matching web context name	ctx1
---------------------------	------

- `http://www.foo.com/ctx2/test.jsp`

Matching virtual host	A
-----------------------	---

Matching web context name	ctx2
---------------------------	------

- `http://www.bar.com/ctx1/`

Matching virtual host	B
-----------------------	---

Matching web context name	ctx1-1
---------------------------	--------

- `http://www.bar2.com/ctx1/test.jsp`

Matching virtual host	C
-----------------------	---

Matching web context name	None (404 Not Found)
---------------------------	----------------------

- `http://www.foo2.com/ctx2/`

Matching virtual host	C
-----------------------	---

Matching web context name	ctx2-2
---------------------------	--------



Web context name and context path have different concepts. In general, they have the same value, but they are different when the services are distinguished by virtual host as in the previous example.

6.3.2. URL Matching Order

URLs are matched in the following order.

1. Match the host header domain name and port string with all registered virtual hosts. If there is a matching virtual host, the web context is searched within the virtual host.



Port information in the format of "IP:Port" can be mapped to a host name set in a virtual host. If port information exists, entire host headers (including ports) are matched.

2. If a web context cannot be found, search within the default virtual host.
3. If the web context does not exist in the default virtual host, a 404 Not Found error is thrown.

6.4. Configuring Virtual Hosts

Virtual hosts can be added, modified, and deleted in the console tool.



In the configuration examples of this section, "A", "B", and "C" are used as the host names for convenience. Use meaningful names in the production environment.

6.4.1. Add

Virtual hosts can be added in the console tool.

Using the console tool

To add a virtual host using the console tool, run the **add-virtual-host** command as follows:

```
add-virtual-host [-cluster <cluster-name> | -server <server-name>]
                 [-f, --forceLock]
                 <virtual-host-name>
                 -list <host-name-list>
                 [-tmin <The minimum number of threads>]
                 [-tmax <The maximum number of threads>]
                 [-tidle <max-idle-time>]
                 [-qs <max-queue-size>]
```

For more information about the command, refer to "add-virtual-host" in *JEUS Reference Guide*.

6.4.2. Modify

Virtual hosts can be modified in the console tool.

Using the console tool

To modify virtual hosts using the console tool, run the **modify-virtual-host** command. For more information about the command, refer to "modify-virtual-host" in *JEUS Reference Guide*.

```
modify-virtual-host [-cluster <cluster-name> | -server <server-name>]
                    [-f, --forceLock]
                    <virtual-host-name>
                    [-alhn1 <access-log-enable-host-name-lookup> |
                    -aluse <use-access-log (true/false)> | -alf
                    <access-log-format> | -aluph
                    <access-log-use-parent-handler (true/false)> |
                    -alex <access-log-excluded-extensions>]
                    [-hnm <host-name> | -hnadd <host-name>]
                    [-ast <attach-stacktrace-on-error>]
                    [-fhn <file-handler-name>]
                    [-fhp <file-handler-permission>]
                    [-tmin <The minimum number of threads>]
                    [-tmax <maximum-thread-num>]
                    [-tidle <max-idle-time>]
```

6.4.3. Delete

Virtual hosts can be deleted using the console tool.

Using the console tool

To delete virtual hosts using the console tool, run the **remove-virtual-host** command. For more information about the command, refer to "remove-virtual-host" in *JEUS Reference Guide*.

```
remove-virtual-host [-cluster <cluster-name> | -server <server-name>]
                    [-f, --forceLock]
                    <virtual-host-name>
```

7. WebSocket Container

This chapter describes the concepts, functions, and configuration of the WebSocket container.

7.1. Overview

The WebSocket container is included in the web engine and one exists for every web context. The WebSocket container provided by JEUS complies with JSR 356 and Java API for WebSocket. Thus, the server's WebSocket service must be written according to the API defined by the relevant standard. But, this document does not include detailed information about WebSocket RFC6455 and Java API for WebSocket standards.



1. Since the HTML5 WebSocket API is a client library, it is irrelevant to WebtoB and JEUS.
2. The Java API for WebSocket is included in Jakarta EE 8. Therefore, reference Jakarta EE 8 libraries to develop applications based on Jakarta EE 8.

The basic functions of the WebSocket container are to deploy WebSocket Server Endpoint objects included in the web context and connect a WebSocket Handshake request from the client to a Server Endpoint object that it maps to. Once a client and a WebSocket container are connected, WebSocket sessions are created and the life cycle of the session objects are also managed.

The role of a service developer is to implement a WebSocket server endpoint class (`jakarta.websocket.Endpoint`) and a configuration class (`jakarta.websocket.server.ServerApplicationConfig`), and package them into the web application.

7.2. Constraints

The following are the constraints for using WebSocket containers in JEUS 9.

- Basically available for HTTP listeners. To use websocket containers in association with WebtoB, refer to the manual of WebtoB.
- Detailed settings are possible under the `<websocket>` tag in `jeus-web-dd.xml`, but websockets can be used without those settings.
- The client API provided by the Java API for WebSocket standard is provided through `lib/system/jeus-websocket-client.jar`. To use another vendor's client module, if the full name of the another vendor's provider class in the `META-INF/service/jakarta.websocket.ContainerProvider` file of the above jar file is specified, then the module can be used via the service loader.

7.3. WebSocket Container Functions

This chapter describes the additional functions of the WebSocket container.

7.3.1. WebSocket UserProperties Failover

A WebSocket container provides space in memory for storing data per WebSocket session. The map object (hereafter UserProperties) obtained by calling the `jakarta.websocket.Session.getUserProperties()` API contains the data.

Assume that WebSocket applications (web context including the WebSocket Server Endpoint) are deployed to two different servers. If data are saved in the UserProperties by using the WebSocket and the server is abnormally terminated, all data will be lost. However, if the data are backed up on another server, the UserProperties can be recovered by connecting the WebSocket to the backup server even if the server is abnormally terminated. From the user perspective, the WebSocket service can be used seamlessly. This is called **WebSocket UserProperties Failover** or **Distributed WebSocket UserProperties**.



The existing user can get the data in UserProperties stored in advance, but the failed over WebSocket endpoint is an endpoint that exists in a different server (not the existing server), and WebSocket Session is newly created. Therefore, this function must be used in very limited cases.

Considerations

The following conditions must be met to use this function.

- WebSocket UserProperties Failover operates based on the HTTP session service. The WebSocket session cluster must be supported. For more information about the session cluster, refer to "Session Cluster Modes" in *JEUS Session Management Guide*.
- The following must be added to the `jeus-web-dd.xml` configuration file of the WebSocket application.

WebSocket UserProperties Failover Setting : `<jeus-web-dd.xml>`

```
<websocket>
  <distributed-userProperties>
    <enabled>true</enabled>
  </distributed-userProperties>
</websocket>
```

In general, the WebSocket session and HTTP session are not integrated for use.

- Data in the UserProperties object must be serializable.
- WebSocket connection is initiated by an HTTP request. The cookie header of the WebSocket

Handshake request must include the JSESSIONID. The HTTP session that the JSESSIONID points to must be created in the web context that the server endpoint belongs to.

For instance, when the JSP that calls the HTML5 WebSocket API is called, the HTTP session is created to send JSESSIONID to the cookie header. When HTML5 WebSocket JavaScript is executed in the web browser, the WebSocket Handshake request is sent to JEUS. The cookie header must be included in the request header. **WebSocket UserProperties Failover cannot be used if the WebSocket client does not support this.**

Firefox 30.0 attaches the cookie header to the WebSocket Handshake request when HTML5 WebSocket API is used.

```
GET /service/chat HTTP/1.1
Connection: keep-alive, Upgrade
Cookie:
JSESSIONID=FheDy8e0b0TP07KNqdeJ7Eps8j51CaQqcRWHvpYo9mdVw1BCSwlwrFiyrc1solkr.amV1czcvc2VydmVyMg==
Sec-WebSocket-Key: Csv/FCQo1g1eZfqMtPd8+g==
Sec-WebSocket-Version: 13
Upgrade: websocket
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:30.0) Gecko/20100101 Firefox/30.0
FirePHP/0.7.4
```



When this function is used with the WebtoB reverse proxy, the session routing must be configured in the reverse proxy configuration. For detailed information, refer to the WebtoB guide.

Considerations

Note the following when using this function.

- The HTTP session's timeout and the WebSocket session's timeout are not related.

Especially in the case of a WebSocket session, since it is dependent on the HTTP session, the HTTP session properties cannot be modified from the WebSocket container level. When serializable data are added(put) to the UserProperties object, depending on the circumstances the HTTP session can time out first which prevents failover from occurring. HTTP session's timeout must be adjusted appropriately taking such situations into consideration.



If the HTTP session times out when adding (put) serializable data to the UserProperties, a WARNING level message will be logged.

- The UserProperties of WebSocket sessions are not shared between different web contexts.

7.3.2. Other Functions

Information that cannot be obtained through the `jakarta.websocket.Session` API

The following information can be obtained through `Map<String, Object>` which is a return value of `Session.getUserProperties()`.

Item	Description
<code>jeus.websocket.remoteAddr(String)</code>	If <code>Forwarded</code> or <code>X-Forwarded-For</code> exists in an HTTP request header, the header value will be used. If not, a <code>HttpServletRequest.getRemoteAddr()</code> value will be used.
<code>jeus.websocket.remoteHost(String)</code>	If <code>Forwarded</code> or <code>X-Forwarded-For</code> exists in an HTTP request header, the header value will be used. If <code>Forwarded</code> or <code>X-Forwarded-For</code> exists in an HTTP request header, the header value will be used. If not, a <code>HttpServletRequest.getRemoteHost()</code> value will be used.
<code>jeus.websocket.remotePort(String)</code>	If <code>Forwarded</code> or <code>X-Forwarded-For</code> exists in an HTTP request header, a port value included in the header will be used. If a header does not have a port value, this property will be not supported. If there is no header, a <code>HttpServletRequest.getRemotePort()</code> value will be used. If the value is less than or equal to 0, it will be invalid.

7.4. Configuring WebSocket Container

Configure the WebSocket container settings in `jeus-web-dd.xml` of each web application.

Web Context Configuration File: `<jeus-web-dd.xml>`

```
<jeus-web-dd xmlns="http://www.tmaxsoft.com/xml/ns/jeus" version="9">
  <websocket>
    <max-incoming-binary-message-buffer-size>8192</max-incoming-binary-message-buffer-size>
    <max-incoming-text-message-buffer-size>8192</max-incoming-text-message-buffer-size>
    <max-session-idle-timeout-in-millis>1800000</max-session-idle-timeout-in-millis>
    <monitoring-period-in-millis>300000</monitoring-period-in-millis>
    <blocking-send-timeout-in-millis>10000</blocking-send-timeout-in-millis>
    <async-send-timeout-in-millis>30000</async-send-timeout-in-millis>
    <websocket-executor>
      <min>10</min>
      <max>30</max>
      <keep-alive-time>60000</keep-alive-time>
      <queue-size>4096</queue-size>
    </websocket-executor>
    <distributed-userProperties>
      <enabled>>false</enabled>
      <use-write-through-policy>>false</use-write-through-policy>
    </distributed-userProperties>
    <init-param>
```

```

    <name>name</name>
    <value>value</value>
  </init-param>
</websocket>
<upgrade>
  <upgrade-executor>
    <min>0</min>
    <max>30</max>
    <keep-alive-time>60000</keep-alive-time>
    <queue-size>4096</queue-size>
  </upgrade-executor>
</upgrade>
</jeus-web-dd>

```

The following describes each configuration tag.

- **<websocket>**

Tag	Description
<max-incoming-binary-message-buffer-size>	<p>Maximum size of the buffer used for text messages transmitted from the client.</p> <p>If the message size is larger than this value, the 1009 error occurs and the WebSocket session is closed.</p>
<max-incoming-text-message-buffer-size>	<p>Maximum size of the buffer used for text messages transmitted from the client.</p> <p>If the message size is larger than this value, the 1009 error occurs and the WebSocket session is closed.</p>
<max-session-idle-timeout-in-millis>	<p>Timeout for closing idle WebSocket sessions.</p> <p>If the value is larger than zero and smaller than 1000, it is set to 1000. (Default value: 30 minutes (1800000ms))</p>
<monitoring-period-in-millis>	<p>Cycle for checking the WebSocket session timeout.</p> <p>If the value is smaller than 1000, it is set to 1000. (Default value: 5 minutes (300000ms))</p>
<blocking-send-timeout-in-millis>	<p>Timeout for waiting for a synchronous send.</p> <p>Applied when using <code>jakarta.websocket.RemoteEndpoint.Basic</code>.</p> <p>(Default value: 10 seconds)</p>
<async-send-timeout-in-millis>	<p>Timeout for waiting for messages awaiting to be sent asynchronously.</p> <p>Returned by <code>jakarta.websocket.WebSocketContainer.getDefaultAsyncSendTimeout()</code>.</p>
<websocket-executor>	<p>Deprecated. Use <upgrade-executor> under <upgrade> instead.</p>

Tag	Description
<distributed-userProperties>	<p>WebSocket Session Failover related settings depending on what is defined in the <code>jakarta.websocket.Session.getUserProperties()</code> method.</p> <p>Set the following sub-tags.</p> <ul style="list-style-type: none"> ◦ <enabled>: Option to use WebSocket session failover. This option is usually not used because it needs to connect to an HTTP session. ◦ <use-write-through-policy>: Option to wait for backup server synchronization when processing put/remove into/from UserProperties of a WebSocket session. This option is usually not used, and the synchronization is processed in background.
<init-param>	<p>Additional configurations used for a WebSocket container.</p> <p>Set a parameter name to <name> and a value to <value>.</p>

- **<upgrade><upgrade-executor>**

A thread pool configuration used internally by the container to process Upgrade inbound messages. It is mainly used for processing the HTTP Upgrade handler during NIO Servlet processing, WebSocket endpoint processing, and handling the SendHandler during WebSocket asynchronous send operations.

Tag	Description
<keep-alive-time>	Timeout for removing idle threads from a thread pool. If set to 0, threads are not removed. (Default value: 1 minute (60000ms))
<queue-size>	Size of a queue that stores tasks processed by a thread pool. (Default value: 4096)

7.5. Using Spring WebSocket

This section describes how to use Spring WebSocket in JEUS.

Spring WebSocket offers an additional feature within the Spring Framework, leveraging the WebSocket container provided by the web engine. This WebSocket container adheres to JSR 356, the Java API for WebSocket, and offers compatible interfaces. However, importing the WebSocket container may vary depending on the vendor.

Since Spring WebSocket supports importing the WebSocket container from only specific vendors, JEUS provides an additional library for utilizing Spring WebSocket. This library requires specific configurations.



The library provided with JEUS is compatible with the Spring Framework 4.2.0 or later.

Add the following configuration for the Spring WebSocket-supporting library.

Web Context Configuration File: <jeus-web-dd.xml>

```
<jeus-web-dd xmlns="http://www.tmaxsoft.com/xml/ns/jeus" version="9">
  <library-ref>
    <library-name>spring-support</library-name>
    <specification-version>
      <value>4.2.0.RELEASE</value>
    </specification-version>
    <failon-error>true</failon-error>
  </library-ref>
</jeus-web-dd>
```

To use the Spring WebSocket-supporting library, specify the websocket handshake-handler as `jeus.spring.websocket.JeusHandshakeHandler`.

Spring Context Configuration File: <spring-context.xml>

```
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:websocket="http://www.springframework.org/schema/websocket"
  xmlns:schemaLocation="
    http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-4.2.xsd
    http://www.springframework.org/schema/websocket
    http://www.springframework.org/schema/websocket/spring-websocket-4.2.xsd">

  <websocket:handlers>
    <websocket:mapping path="/chat" handler="chatHandler"/>
    <websocket:handshake-handler ref="handshakeHandler"/>
  </websocket:handlers>

  <bean id="chatHandler" class="com.tmax.jeus.ChatHandler"/>
  <bean id="handshakeHandler" class="jeus.spring.websocket.JeusHandshakeHandler"/>
</beans>
```

8. JEUS WebCache

This chapter describes how to use JEUS WebCache to improve the web application performance.

8.1. Overview

Performance is an important concern for any application, but it becomes a critical factor when the application is a web service that is accessed by thousands of clients simultaneously. Performance degradation such as delayed responses due to high traffic can be addressed both at the hardware and software levels.

A resolution from the hardware side is to improve response times by installing more servers and distributing the requests on multiple servers with load balancing. However, this increases costs and the complexity of managing and operating servers due to clustering and other factors.

A resolution from the software side is to cache frequently-used data without server scaling. For subsequent requests, applications can use the cached data instead of regenerating the data, which improves response times and increases performance.

This section describes how to use JEUS WebCache in the JEUS system to improve the performance of web applications. The following are the provided caching methods.

- JSP Caching

Partially caches JSP pages by using the tag library. This is useful when there are requests for partially modified JSP pages.

- HTTP Response Caching

Caching the entire HTTP response. This is useful when there are requests for static contents.

Entries that are cached in JEUS WebCache use soft references, which can prevent OutOfMemory errors from occurring.

To make the best use of caching, select pages that are frequently used or incur long response times due to complex database queries as caching targets.

8.2. JSP Caching

JSP caching improves the performance of web applications by storing parts of JSP pages in JEUS WebCache by using the JSP tag library.

8.2.1. General Information

JEUS WebCache uses `<jeus:cache>` as a user-defined (custom) tag.

When putting a JSP page content in **<jeus:cache>**, the body content of the tag is created when the first request occurs. The content will be sent in the response to be cached. The cached content will be returned when the next request occurs.

The **<jeus:cache>** tag can be used as in the following.

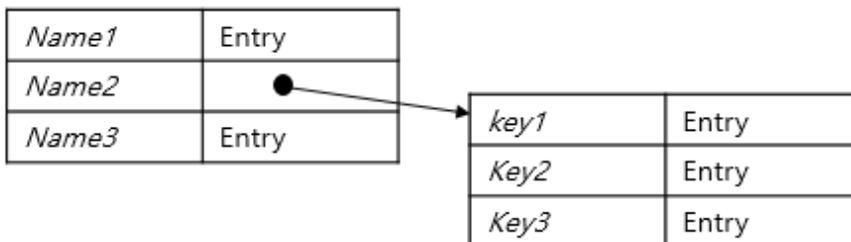
```
<%@ taglib uri="http://www.tmaxsoft.com/jeuscache" prefix="jeus" %>

<jeus:cache name="..." key="..." scope="..." timeout="..."
  size="..." async="..." df="...">
  . . . Body content to be cached. . .
</jeus:cache>
```

The **flush** attribute uses a self-closing tag (**/>**) and is described in [<cache> Tag](#).

The algorithm used in JSP Caching is LRU. If the number of JEUS WebCache entries exceeds the maximum allowed number, the cached entries will be deleted according to the LRU algorithm. The TLD file (Tag Library Descriptor) is included in the 'jeus.jar' file that is deployed. To send the URI information from the 'jeuscache.tld' file to the JSP engine, **'taglib uri'** in the **<jeus:cache>** tag must be specified as 'http://www.tmaxsoft.com/jeuscache'.

The following diagram shows the data structure that is used to cache entries by using the **'name'** attribute and the **'name'** + **'key'** attributes.



Cache Data Structure

In the previous figure, Name1 and Name3 are cached by only using the **'name'** attribute as a tag. If the attributes **'name'** and **'key'** are both used as tags, entries are cached in the same way as with Name2.

8.2.2. <cache> Tag

The jeuscache.tld file defines the user-defined tag **<jeus:cache>**. This file is included in jeus-servlet.jar and is located in the following path.

```
jeus/servlet/cache/resource/jeuscache.tld
```

The following example shows how to configure the **<cache>** tag in the jeuscache.tld file.

Configuring <cache> Tag: <jeuscache.tld>

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<!DOCTYPE taglib PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag
Library 1.2//EN" "http://java.sun.com/dtd/web-jsptaglibrary_1_2.dtd">

<taglib>
  <tlib-version>1.0</tlib-version>
  <jsp-version>1.2</jsp-version>
  <short-name>jeuscache</short-name>
  <uri>http://www.tmaxsoft.com/jeuscache</uri>
  <display-name>JEUSCache Tag Library</display-name>
  <tag>
    <name>cache</name>
    <tag-class>jeus.servlet.cache.web.tag.CacheTag</tag-class>
    <body-content>JSP</body-content>
    <description>JEUS WebCache</description>
    <attribute>
      <name>flush</name>
      <required>>false</required>
      <rtexprvalue>>true</rtexprvalue>
    </attribute>
    <attribute>
      <name>timeout</name>
      <required>>false</required>
      <rtexprvalue>>true</rtexprvalue>
    </attribute>
    <attribute>
      <name>scope</name>
      <required>>false</required>
      <rtexprvalue>>true</rtexprvalue>
    </attribute>
    <attribute>
      <name>name</name>
      <required>>false</required>
      <rtexprvalue>>true</rtexprvalue>
    </attribute>
    <attribute>
      <name>size</name>
      <required>>false</required>
      <rtexprvalue>>true</rtexprvalue>
    </attribute>
    <attribute>
      <name>key</name>
      <required>>false</required>
      <rtexprvalue>>true</rtexprvalue>
    </attribute>
    <attribute>
      <name>async</name>
      <required>>false</required>
      <rtexprvalue>>true</rtexprvalue>
    </attribute>
    <attribute>
      <name>df</name>
      <required>>false</required>
      <rtexprvalue>>true</rtexprvalue>
    </attribute>
  </tag>
</taglib>
```

The following describe each tag attribute.

Attribute	Description
flush	<p>Used to delete cache entries. To remove an entry, specify the 'name' attribute or the 'name' + 'key' attributes.</p> <p>When both 'name' and 'key' are used to cache the entries, if the 'flush' attribute only uses the 'name' attribute, all entries that have the 'key' attribute as an identifier will be deleted. Unlike other attributes, the 'flush' attribute uses a self-closing tag (<code></></code>) without any content.</p>
timeout	<p>The cache period in the Simple Date Format. Use a string that is a combination of a number and a character representing time.</p> <p>The valid characters that represent the time are 's' (seconds), 'm' (minutes), 'h' (hours), 'd' (days), and 'w' (weeks). For example, 10s indicates 10 seconds, 10d indicates 10 days, and 4w indicates 4 weeks. The default time character is seconds, so if a number is entered without a character, it is considered as seconds.</p> <p>If set to 0, the body content will not be cached. If set to -1, the cached content will not expire until it is forcibly flushed.</p> <p>(Default value: 1 hour)</p>
scope	<p>The scope of the cached entries.</p> <ul style="list-style-type: none"> ◦ application(Default value) ◦ session
name	<p>This allows multiple pages to share the cached data. The name must be unique within the scope.</p> <p>If not set, it will be created based on the URI. If the body content of <code>jeus:cache</code> does not need to be shared, do not use this attribute.</p>
size	<p>The maximum number of objects that can be cached.</p> <p>If the number of cached objects exceeds this value, objects in the cache will be deleted using the LRU algorithm. The maximum value should be set according to each web application. (Default value: Integer.MAX_VALUE)</p>

Attribute	Description
key	<p>Another value that identifies cached entries. This attribute must be used with the 'name' attribute, so the 'name' + 'key' value is used as a unique identifier for an entry.</p> <p>The 'key' attribute can be set to a list of scopes as in the following.</p> <pre><jeus:cache name=". . ." key="[parameter page session request application].keyname" . . . ></pre> <p>In the previous example, the 'keyname' value must be set when sending a request. For 'parameter', 'page', or 'request', the 'keyname' must be set to the key value from the request page that matches the cached page. For 'application' and 'session', it can be set from other pages.</p> <pre>http://sample.com:8088/Sample/index.jsp?keyname=test</pre>
async	<p>Option to allow other threads to accessing the entry when a thread is updating the entry.</p> <ul style="list-style-type: none"> ◦ true: Entry is not blocked, and the requesting threads retrieve the value of the entry before the update. (Default value) ◦ false: Entry is blocked, and threads wait for the entry to be updated and retrieves the updated entry.
df	<p>Deletes entries when overflow occurs. The number of entries that will be deleted can be set in the 'df' attribute as a ratio of 'size'.</p> <p>Available values are real numbers where '0.0 ≤ factor ≤ 1.0'. (Default value: 0.25)</p> <p>For example, if the factor is 1, all cached entries are deleted. If the factor is 0, no more requests will be cached if there is an overflow.</p>

8.2.3. Example of Using <jeus:cache>

This section shows how to use the <jeus:cache> tag attributes.

The following example compares the cached date with the current date when there is a request for the cache.jsp page.

Using <jeus:cache>: <cache.jsp>

```
<%@ taglib uri="http://www.tmaxsoft.com/jeuscache" prefix="jeus" %>
<HTML>
<BODY>
  Current time: <%= new Date() %><br>
  <jeus:cache timeout="60s">
    Cached time: <%= new Date() %>
  </jeus:cache>
```

```
</BODY>
</HTML>
```

When the first request is made using the `<jeus:cache>` tag, the current date is displayed on the screen and the content will be cached in the JEUS WebCache. On the next request, the cached date will be displayed. After 60 seconds, an updated date will be displayed.

The following example shows how to use the `<jeus:cache>` tag when `<jsp:include>` is used to include other pages.

Using `<jsp:include>`: `<main.jsp>`

```
<HTML>
<BODY>
  <jsp:include page="cache.jsp"/>
</BODY>
</HTML>
```

The content of the `<jeus:cache>` tag in `cache.jsp` that is included in `main.jsp` displays the same result as the first example that only uses `cache.jsp`.

8.2.4. Using Flush

The flush tag forcibly deletes cached entries. Flush the entry using the 'name' attribute or the 'name' + 'key' attributes.

This example assumes that stock information is cached using the 'name' + 'key' attributes.

```
<jeus:cache name="stock" key="parameter.company" scope="application">
  . . . stock content . . .
</jeus:cache>
```

Use the following to delete the stock information that corresponds to the "parameter.company" key in the cache.

```
<jeus:cache name="stock" key="parameter.company" scope="application" flush="true"/>
```

If **flush** is executed successfully as in the previous example, the updated stock content will be displayed when the stock information in the `<jeus:cache>` tag is requested. If only the 'name' attribute is used for the **flush** attribute, all entries that were saved with the "parameter.company" key will be deleted.

```
<jeus:cache name="stock" scope="application" flush="true"/>
```

If the entries were cached by using only the 'name' attribute without the 'key' attribute, only the

'name' attribute can be used to flush the entries.

8.2.5. Using Refresh

If refresh is used, the body content will be created whenever `<jeus:cache>` tag is called. This is not a `<jeus:cache>` tag attribute. It can be configured by setting `'_jeuscache_refresh'` to 'true'.

Use the following to refresh all entries in the application and session scopes.

```
<% application.setAttribute("_jeuscache_refresh", "true"); %>
<% session.setAttribute("_jeuscache_refresh", "true"); %>
```

This function checks the `'_jeuscache_refresh'` value in each scope when the `<jeus:cache>` tag is accessed. If the value is true, it updates the body content. To disable refresh, set `'_jeuscache_refresh'` to 'false'. If the 'timeout' and 'flush' attributes are used, only updated parts of the entries are displayed. If refresh is used, all body contents of the scope are updated each time.

8.3. HTTP Response Caching

JEUS WebCache supports HTTP Response Caching function that caches the entire HTTP response by using servlet filters.

This method is not appropriate for pages are updated dynamically, but is useful for static contents such as image files and PDFs. This method can be used for web pages whose contents do not change or do not change often.

```
http://www.sample.com/filter/respcacheTest.jsp?key=value
```

As in the previous example, the entire URI that includes the key and value is used as the entry key. If the key or value are updated dynamically, the response for each URI will be cached.



The HTTP response is only cached when the HTTP response state is 200 OK (HttpServletResponse.SC_OK). The URI is used as the entry key in JEUS WebCache.

This section describes how to apply HTTP response caching to web applications.

8.3.1. Configuring Filters

To use HTTP response caching, register a filter in web.xml.

The following example caches responses for all HTTP requests for which the `<url-pattern>` is `'/filter/'` for 10 minutes.



Note that the `jeus.servlet.cache.web.filter.CacheFilter` class must be used as the `<filter-class>`.

Configuring Filters: <web.xml>

```
<web-app>
  . . .
  <filter>
    <filter-name>CacheFilter</filter-name>
    <filter-class>jeus.servlet.cache.web.filter.CacheFilter</filter-class>
    <init-param>
      <param-name>timeout</param-name>
      <param-value>600</param-value>
    </init-param>
    <init-param>
      <param-name>lastModified</param-name>
      <param-value>on</param-value>
    </init-param>
    <init-param>
      <param-name>expires</param-name>
      <param-value>off</param-value>
    </init-param>
  </filter>

  <filter-mapping>
    <filter-name>CacheFilter</filter-name>
    <url-pattern>/filter/*</url-pattern>
  </filter-mapping>
  . . .
</web-app>
```

The following describe the initialization parameters that are sent to the filter class.

Parameter	Description
timeout	Timeout for caching HTTP responses. The default value is 3600 and the units are in seconds. This is set in the same way as the 'timeout' attribute for JSP caching, except that HTTP response caching does not provide the flush function. Hence, if 'timeout' is set to '-1', the web application will not expire unless it is undeployed.

Parameter	Description
lastModified	<p>Option to send the Last-Modified header in the HTTP response. This is used to reduce the load on the web engine.</p> <p>The browser can ask the web engine if the cached contents have been updated since the last HTTP request. The web engine then compares the If-Modified-Since header information in the HTTP request with the Last-Modified time of the current entry. If there are no changes, the web engine sends an HTTP 302 code (HttpServletResponse.SC_NOT_MODIFIED).</p> <p>Options are:</p> <ul style="list-style-type: none"> ◦ on: Determine the last modified time during filtering, and send an HTTP 304 code. ◦ off: Do not send a 304 code as the HTTP response. ◦ initial: Send a 304 code after setting the last modified time to the current time. This is the default value.
expires	<p>Option to send the Expires header information in the response. If the browser uses caching, the cached contents are valid until they expire. They are used for subsequent HTTP requests.</p> <p>However, if an entry in JEUS WebCache is updated, the new entry will conflict with the entry stored in the browser cache. In this case, set the Expires header information in the web engine to expire the content that is stored in the browser cache, and use the updated entry in the JEUS WebCache.</p> <p>Options are:</p> <ul style="list-style-type: none"> ◦ on: If a value is set in the filter chain, send the Expires header. ◦ off: Do not send the Expires header. ◦ timeout: Add the previous timeout parameter value to the last-modified value of the HTTP response, set it in the Expires header, and send the response to the client.



In addition to these parameters, the **scope**, **size**, **async**, and **df** parameters can also be configured. They are the same as those described in [JSP Caching](#).

9. Reloading Classes at Runtime

This chapter describes servlet auto reloading that helps reduce the web application development time.

9.1. Overview

In general, the Java EE development life cycle consists of the following steps.

1. Edit
2. Build
3. Deploy
4. Test

Servlet classes are often modified when developing Java EE applications, especially web applications. Much effort has been made to shorten the development process, and WebLogic 10.3 provides functions that reduce the redeployment process through FastSwap.

Java EE 5 introduced a function that can redefine classes without terminating the class loader or instances while applications are running, but declared fields and methods cannot be modified.

Up to JEUS 6, to reload a modified class, the application must be redeployed or a new class loader must be created using automatic reloading to replace the existing class loader. Automatic reloading can be executed at a fixed interval or on demand.

However, if the application is too big, it takes a long time to redeploy the application and automatic reloading uses as much overhead as redeployment because it recreates the class loader.

From JEUS 7 onwards, **JEUS HowSwap** functionality is provided, which can reload modified Java classes dynamically without reloading them (Automatic reloading) by using the JDK instrumentation package. It can currently only be used for web application classes.



The automatic reloading (JEUS HotSwap activated/inactivated) function can generate unexpected system overhead in a production environment, so it is recommended to only use it in a development environment.

9.2. Basic Configurations and Operation

This section describes how to configure and operate automatic reloading.

9.2.1. Configuring Servers

JEUS must be configured to use HotSwap before it is started. Auto reloading that does not have the JEUS HotSwap function starts without the system option when the server is started.

The system option is `jeus.server.useHotSwapAgent` and its default value is 'false'. If it is not set, auto reloading cannot be used.

The following are examples of configuring JEUS start scripts.

- **startDomainAdminServer script configuration**

Configuring Automatic Reloading: `<startDomainAdminServer>`

```
...
"${JAVA_HOME}/bin/java" $VM_OPTION $SESSION_MEM
-Xbootclasspath/p:"${JEUS_HOME}/lib/system/extension.jar"
...
-Djeus.server.useHotSwapAgent=true
...
jeus.launcher.Launcher ${BOOT_PARAMETER}
...
```

- **startManagedServer script configuration**

Configuring Automatic Reloading: `<startManagedServer>`

```
...
"${JAVA_HOME}/bin/java" $VM_OPTION $SESSION_MEM
-Xbootclasspath/p:"${JEUS_HOME}/lib/system/extension.jar"
...
-Djeus.server.useHotSwapAgent=true
...
jeus.launcher.ManagedServerLauncher ${BOOT_PARAMETER}
...
```



Enable automatic reloading during development, but disable it in the production environments.

9.2.2. Configuring Applications

To use automatic reloading, `<auto-reload>` must be configured in `jeus-web-dd.xml`. To use JEUS HotSwap, `<use-jvm-hotswap>` inside `<auto-reload>` must be set to true. These functions can only be used in class files that are packaged as directories (exploded directories) of web applications. These functions are used to dynamically reload modified classes in already deployed directories. In other words, they can only be used for modified classes in the 'WEB-INF/classes' directory. Automatic reloading in JEUS runs from the time when the class modification occurs after the application is initially deployed.

Configure `<auto-reload>` in `jeus-web-dd.xml` as in the following. Set the monitoring interval for auto reloading in `domain.xml`. For more information about setting the monitoring interval in `domain.xml`,

refer to [Monitoring](#).

Configuring Automatic Reloading: <jeus-web-dd.xml>

```
<jeus-web-dd xmlns="http://www.tmaxsoft.com/xml/ns/jeus" version="9">
  . . .
  <auto-reload>
    <enable-reload>true</enable-reload>
    <use-jvm-hotswap>true</use-jvm-hotswap>
    <check-on-demand>true</check-on-demand>
  </auto-reload>
  . . .
</jeus-web-dd>
```

Automatic reloading uses the following settings.

Tag	Description
<enable-reload>	Option to use automatic reloading. Must be set to true.
<use-jvm-hotswap>	<ul style="list-style-type: none">◦ true: Instead of replacing the class loader, redefines the modified classes or applies the changes using class retransformation. If there are any classes that cannot be redefined or retransformed during the process, JEUS HotSwap will stop and auto reloading is executed to reload the changes. For information about class modifications that cannot be applied using JEUS HotSwap, refer to JEUS HotSwap Constraints.◦ false: JEUS HotSwap is not used. Replaces the class loader of the application to reload the modified classes.
<check-on-demand>	<ul style="list-style-type: none">◦ true: When a request is received, this immediately finds the modified files by checking the last modified time of each class based on the application class loader. If modified classes are found, automatic reloading will be performed according to the <use-jvm-hotswap> setting.◦ false: The web engine finds modified files by checking the last modified time of each class based on the application class loader at the interval set in 'Check Class Reload'. The default value is 300 seconds. For more information about 'Check Class Reload' refer to Monitoring. <p>If modified classes are found, automatic reloading will be performed according to the <use-jvm-hotswap> setting.</p>

9.2.3. Applications and Modifications Supported by JEUS HotSwap

JEUS HotSwap supports POJOs and web application classes in exploded directories.

The following modifications are supported by JEUS HotSwap.

- Adding and deleting static class constructors
- Adding and deleting regular class constructors

- Changing static method bodies
- Changing regular method bodies

The following list of class modifications are supported by JEUS HotSwap.

- **Java class instances (non-abstract)**

Java Change Type	Supported	Notes
Adding and deleting methods	No	
(1) Adding and deleting fields	No	
(2) Changing method bodies	Yes	
(3) Adding and deleting constructors	Yes	
(4) Changing field modifiers	No	

- **Static classes**

Java Change Type	Supported	Notes
Adding and deleting methods	No	
Changing method bodies	Yes	

- **Abstract Java classes**

Java Change Type	Supported	Notes
Adding and deleting abstract methods	No	
Change types (1)-(4) of Java class instances	Yes	

- **Final Java classes**

Java Change Type	Supported	Notes
Change types (1)-(4) of Java class instances	Yes	

- **Final Java methods**

Java Change Type	Supported	Notes
Change types (1)-(4) of Java class instances	Yes	

- **final Java fields**

Java Change Type	Supported	Notes
Change types (1)-(4) of Java class instances	Yes	

- **Enum**

Java Change Type	Supported	Notes
Adding and deleting constants	No	
Adding and deleting methods	No	

- **Anonymous internal classes**

Java Change Type	Supported	Notes
Adding and deleting fields	Not Available	Java does not support this.
Adding and deleting methods	No	

- **Static internal classes**

Java Change Type	Supported	Notes
Change types (1)-(4) of Java class instances	Yes	

- **Regular internal classes**

Java Change Type	Supported	Notes
Change types (1)-(4) of Java class instances	Yes	

- **Java interfaces**

Java Change Type	Supported	Notes
Adding methods	No	

- **Java Reflection**

Java Change Type	Supported	Notes
Accessing existing fields and methods	Yes	
Accessing new methods	No	New methods are not visible when using reflection.
Accessing new fields	No	New fields are not visible when using reflection.

- **Annotations on Classes**

Java Change Type	Supported	Notes
Adding and deleting method and property annotations	No	

- **Annotations**

Java Change Type	Supported	Notes
Adding and deleting method and property annotations	No	

- **Exception classes**

Java Change Type	Supported	Notes
Change types (1)-(4) of Java class instances	Yes	

- **EJB interfaces**

Java Change Type	Supported	Notes
Adding and deleting methods	No	Changing EJB interfaces related to reflection is not supported.

- **EJB 3.0 Session and MDB implementation classes**

Java Change Type	Supported	Notes
Adding and deleting methods and fields	No	Only supported classes among the classes referenced by EJB can be modified.

- **EJB 2.X Entity Bean**

Java Change Type	Supported	Notes
Adding and deleting methods and fields	No	Only supported classes among the classes referenced by EJB can be modified.

- **EJB Interceptors**

Java Change Type	Supported	Notes
Adding and deleting methods and fields	No	Only supported classes among the classes referenced by EJB can be modified.

9.2.4. JEUS HotSwap Constraints

Not all types of changes are supported by JEUS HotSwap. If an attempt is made to make an unsupported change, the JDK will return an `UnsupportedOperationException`. JEUS receives the error message and logs it as "Re-transforming all modified classes in the servlet context [AAA] failed." When this happens, the application will be not updated dynamically, but instead, automatic reloading is maintained.

```
Retransforming all modified classes in the servlet context [AAA] failed.
```

The following changes are not supported in JEUS HotSwap.

- Since Java reflection results cannot include newly modified fields and methods, using reflection on modified classes can cause unexpected results.
- Changing the existing class hierarchy is not supported. For example, changing a list that implements class interfaces or modifying the class' superclass are not supported.
- Adding or deleting Java annotations is not supported, due to the limitations of reflection.
- Adding or deleting EJB interface methods is not supported because it requires the reflection of the changes.
- Adding or deleting enumeration constants is not supported.
- Adding or deleting finalize methods is not supported.

10. Valves and Filters

This chapter describes how to configure Valves and Filters in JEUS.

10.1. Overview

A Valve, a concept adopted from Apache Tomcat, is a component that can be inserted into the request processing pipeline to check or modify requests. Each Valve can be configured at different levels, such as the Web Engine, Virtual Host, and Context. A Filter is another component used for processing requests, similar to Valves. It is represented by `jakarta.servlet.Filter`. Unlike Valves, Filters can be configured at the servlet level. For more information, refer to `jakarta.servlet.Filter`.

10.2. Concept of Valves

This chapter describes the basic concept and use of a Valve.

If Filters operate only at the servlet level, Valves serve in a broader scope, such as the server and virtual host levels. The Valve code execution starts at the web engine level, then proceeds to the virtual host level, and finally, the context level. If the code is called within the same level, the specified order will be applied. If Filters and Valves are both configured, Valves take precedence over Filters.

Configuring jeus-web-dd.xml

Create jeus-web-dd.xml under the WEB-INF/ directory. For further description for this, see [Web Contexts](#).

The following is a sample jeus-web-dd.xml file. It shows only a specific part of the file. To see the full file with all configuration items, refer to "13. Configuration of jeus-web-dd.xml" in "JEUS XML Reference".

Web Context Configuration File: <jeus-web-dd.xml>

```
<jeus-web-dd xmlns="http://www.tmaxsoft.com/xml/ns/jeus" version="9">
<context-path>/examples</context-path>
  <pipeline>
    <valve>
      <class-name>jeus.servlet.valve.RemoteAddressValve</class-name>
      <property>
        <key>deny</key>
        <value>127\.\0\.\0\.\1</value>
      </property>
      <property>
        <key>denyStatus</key>
        <value>403</value>
      </property>
    </valve>
  </pipeline>
</jeus-web-dd>
```

The following example configures a Valve at the web engine level in domain.xml.

Server configuration file: <domain.xml>

```
<domain xmlns="http://www.tmaxsoft.com/xml/ns/jeus" version="9">
  <web-engine>
    <pipeline>
      <valve>
        <class-name>jeus.servlet.valve.RemoteAddressValve</class-name>
        <property>
          <key>deny</key>
          <value>127\.\0\.\0\.\1</value>
        </property>
        <property>
          <key>denyStatus</key>
          <value>403</value>
        </property>
      </valve>
    </pipeline>
  </web-engine>
</domain>
```

The following describes each configuration tag.

Tag	Description
<valve>	Valve to use.
<class-name>	Class name of the valve. Same as the filter class name.
<property>	Property for the valve. This property consists of key-value pairs, which are stored in the parameter map within the valve. Multiple properties can be configured.
<key>	Key to use in the parameter map within the valve object. Must be unique within the map. If duplicate keys are found, the later configured key value is applied.
<value>	Value to store in the parameter map for the valve. The data type is string, and the type casting is performed in the init() method, which will be discussed later in this chapter.



The class to be specified in <class-name> must inherit the jeus.servlet.valve.ValveBase class.

10.3. Configuring JEUS-implemented Valves

JEUS provides valve implementation for user convenience.

10.3.1. Restricting the Use of HTTP Methods

This valve restricts or allows the use of certain HTTP methods.

Adding a valve to restrict methods: <jeus-web-dd.xml>

```
<jeus-web-dd xmlns="http://www.tmaxsoft.com/xml/ns/jeus" version="9">
<context-path>/examples</context-path>
  <pipeline>
    <valve>
      <class-name>jeus.servlet.valve.MethodConstraintValve</class-name>
      <property>
        <key>allow</key>
        <value>GET,POST</value>
      </property>
    </valve>
  </pipeline>
</jeus-web-dd>
```

The class provided by JEUS for restricting the use of HTTP methods is `jeus.servlet.valve.MethodConstraintValve`. You can set the key to either allow or deny, and the value to the methods to allow or restrict. Separate each method by comma (,).

10.3.2. Allowing or Blocking Remote Access

This valve determines whether to allow or block a remote host/address.

To block, set the key to deny. To allow, set the key to allow. The key is case-sensitive. In the value tag, specify the remote address or host to block or allow. Here, the value for the remote host must be the value returned by the `getRemoteHost()` method which can be called with the `ServletRequest` object. For more information, see [getRemoteHost\(\) Method of ServletReques](#).

The string to be specified in the value tag must comply with the Java regular expressions. The `java.util.regex` provides the classes and interfaces for regular expressions. The `denyStatus` key specifies the HTTP status code to return when blocking the request. If not specified, the default value is 403.

The following example only allows local accesses.

Adding a valve to allow remote access by address: <jeus-web-dd.xml>

```
<jeus-web-dd xmlns="http://www.tmaxsoft.com/xml/ns/jeus" version="9">
<context-path>/examples</context-path>
  <pipeline>
    <valve>
      <class-name>jeus.servlet.valve.RemoteAddressValve</class-name>
      <property>
        <key>allow</key>
        <value>127\.\0\.\0\.\1</value>
      </property>
      <property>
        <key>denyStatus</key>
        <value>403</value>
      </property>
    </valve>
  </pipeline>
```

```
</jeus-web-dd>
```

The following example configures a valve to block remote accesses by a remote host 'www.tamx.co.kr'.

Adding a valve to block remote access by host: <jeus-web-dd.xml>

```
<jeus-web-dd xmlns="http://www.tmaxsoft.com/xml/ns/jeus" version="9">
<context-path>/examples</context-path>
  <pipeline>
    <valve>
      <class-name>jeus.servlet.valve.RemoteHostValve</class-name>
      <property>
        <key>deny</key>
        <value>www\.tmax\.co\.kr</value>
      </property>
    </valve>
  </pipeline>
</jeus-web-dd>
```



The key and value strings are case-sensitive. Also, do not specify both deny and allow at the same time.

10.3.3. URL Rewriting

This valve enables URL rewriting and implements the same rules as the RewriteValve in Tomcat. URL rewriting serves two primary purposes: preventing domain exposure and redirecting requests to other locations. Define the rules for rewriting in a dedicated configuration file named 'rewrite.config'. For more information about rewriting rules, refer to [Apache Tomcat 8.5 Rewriting Document](#).

You can configure the URL rewriting valve either at the web engine or context level. To configure the valve at the web engine level, place the configuration file under the DOMAIN/config directory. To configure the valve at the context (application) level, place the file under APP_HOME/WEB-INF. The class provided by JEUS for URL rewriting is jeus.servlet.valve.rewrite.RewriteValve.

Adding a valve to enable URL rewriting: <jeus-web-dd.xml>

```
<jeus-web-dd xmlns="http://www.tmaxsoft.com/xml/ns/jeus" version="9">
<context-path>/examples</context-path>
  <pipeline>
    <valve>
      <class-name>jeus.servlet.valve.rewrite.RewriteValve</class-name>
      <property>
        <key>encoding</key>
        <value>utf-8</value>
      </property>
    </valve>
  </pipeline>
</jeus-web-dd>
```

Enter 'encoding' in the key tag with the desired encoding for URL rewriting in the value tag. If the encoding is not explicitly specified, the request query encoding of JEUS is used by default. Ensure that the encoding remains consistent across JEUS, the rewrite.config file, and the RewriteValve class, especially when using Korean characters.

In JEUS, the query encoding is determined in the following order of priority:

1. Forced URL encoding in domain.xml
2. Forced request encoding
3. Client's override encoding
4. URL mapping encoding
5. Default encoding
6. ISO-8859-1, if none of the above is set.



You must avoid using `jeus.servlet.request.6CompatibleSetCharacterEncoding` along with any of the above settings. It is intended for backward compatibility, and may lead to out-of-specification behaviors and disrupt the priority order of encoding.

10.4. User Custom Valves

Developers using JEUS can also implement inherited valves.

To add a valve, inherit `jeus.servlet.valve.ValveBase`. Inheriting this class requires two libraries: `jakarta.servlet-api.jar` and `jeus-servlet-engine.jar` located in `JEUS_HOME/lib/system`. After building the class, package it into a jar file along with these libraries and add the file to `JEUS_HOME/lib/thirdparty`. When adding the jar file, ensure that it does not contain duplicate `jakarta.servlet-api.jar` and `jeus-servlet-engine.jar`.

The following shows a sample valve that adds test to the HTTP response header.

Adding test to the response header: test

```
package com.test.valve;

import jeus.servlet.valve.ValveBase;
import jeus.servlet.valve.ValveException;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.util.Map;

public class TestValve extends ValveBase {
    private String test;

    @Override public void init() throws ValveException {
        Map<String,String> paramMap = getValveParameterMap();
```

```

        test = paramMap.get("test");
    }
    @Override public void invoke(HttpServletRequest req, HttpServletResponse resp) throws
IOException, ServletException {
        //Before running application
        getNext().invoke(req, resp);
        //After running application
        resp.setHeader(test, test);
    }
}

```

If the valve is configured at the context level, the `init()` method is invoked when the application is deployed. If configured in `domain.xml`, it is invoked when the virtual host or web engine starts up. The `getValveParameterMap()` method imports the properties map which is configured in `jeus-web-dd.xml` or `domain.xml`. The `invoke()` method defines the behaviors of the valve for incoming requests. It basically has the same structure as in a servlet filter. The `getNext().invoke()` method functions the same as the `chain.doFilter()` method in filter implementation. Before `getNext().invoke()` is called, filter is operated with the request, and then response filtering is performed with the response.

Refer to [JEUS_HOME/docs/api/jeus-servlet](https://jeus.apache.org/docs/api/jeus-servlet) for information about `jeus.servlet.valve.ValveBase`.



Even if the `init()` method performs no specific operation, it must be implemented anyway. The `invoke()` method must be implemented as well, because it defines the behaviors of the valve. Also, ensure that the `getNext().invoke()` method is called when implementing the `invoke()` method. Request processing is always performed at the last valve defined in JEUS. Therefore, if a request fails to proceed to the subsequent valve, it cannot be processed.

10.5. Configuring JEUS-implemented Filters

JEUS provides filter implementation for user convenience.

10.5.1. SameSite Restriction

SameSite restriction can be configured individually at the context level, but cannot operate differently by identifying individual user-agents. Since there may be some clients that cannot process SameSite, JEUS provides a filter designed to remove SameSite based on the identification of different user-agents.

Adding a filter for SameSite restriction: `<web.xml>`

```

<?xml version="1.0" encoding="UTF-8"?>
<web-app 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/web-app_5_0.xsd"
    metadata-complete="false"
    version="5.0">

```

```

<filter>
  <filter-name>NoSameSiteFilter</filter-name>
  <filter-class>jeus.servlet.filters.NoSameSiteFilter</filter-class>
  <init-param>
    <param-name>agent</param-name>
    <param-value>^Chrome/\w$</param-value>
  </init-param>
</filter>

<filter-mapping>
  <filter-name>NoSameSiteFilter</filter-name>
  <servlet-name>NoSameSiteServletTest</servlet-name>
</filter-mapping>

</web-app>

```

The class provided by JEUS for SameSite restriction is `jeus.servlet.filters.NoSameSiteFilter`. Specify 'agent' as the parameter name in the `init-param` tag, with the regular expression for the user-agent for which the SameSite attribute will be removed in the value tag.

10.5.2. Forwarded Header

This filter retrieves information from the client instead of the proxy through the `jakarta.servlet.HttpServletRequest` API, by identifying the RFC 7239-defined Forwarded header and the de facto headers (e.g. X-Forwarded).

The filter class is `jeus.servlet.filters.ForwardedFilter`, which automatically finds and applies the Forwarded header. However, only four Forwarded headers (for, host, proto, by) are supported based on the specification. For more information, refer to RFC 7239.

To apply de facto headers as well, specify the headers to use in the `InitParameter` (`init-param`) for the filter. `InitParameter` supports 6 keys: `Forwarded-For`, `Forwarded-Host`, `Forwarded-Proto`, `Forwarded-By`, `Forwarded-Port`, and `Forwarded-Server`; In the `InitParameter` value, specify the headers to apply by separating each with a comma(,). The following shows an example.

```

<filter>
  <filter-name>ForwardedFilter</filter-name>
  <filter-class>jeus.servlet.filters.ForwardedFilter</filter-class>
  <init-param>
    <param-name>Forwarded-For</param-name>
    <param-value>X-Forwarded-For,Proxy-Client-IP,WL-Proxy-Client-IP,HTTP_CLIENT_IP,HTTP_X_FORWARDED_FOR</param-value>
  </init-param>
</filter>

```

Among the headers, Forwarded has the highest priority. After this, those specified in the `param-value` tag are applied in the written order. It should be noted that if you specify headers that are not related with the key (host headers), invalid values may be returned by the API.

To check the candidate headers filtered through the filter, call the `HttpServletRequest.getAttribute()` API with the following. The value will be returned in the format of `Map<String, List<String>>`. Here, the String is the header name and `List<String>` contains the header values.

```
jeus.servlet.filters.ForwardedFilter.For
jeus.servlet.filters.ForwardedFilter.Host
jeus.servlet.filters.ForwardedFilter.Proto
jeus.servlet.filters.ForwardedFilter.By
jeus.servlet.filters.ForwardedFilter.Port
jeus.servlet.filters.ForwardedFilter.Server
```

The associated `HttpServletRequest` APIs are `getRemoteAddr()`, `getServerName()`, `getServerPort()`, `getScheme()`, and `isSecure()`.

Adding a filter for Forwarded headers: <web.xml>

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app 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/web-app_5_0.xsd"
  metadata-complete="false"
  version="5.0">

  <filter>
    <filter-name>ForwardedFilter</filter-name>
    <filter-class>jeus.servlet.filters.ForwardedFilter</filter-class>
    <init-param>
      <param-name>Forwarded-For</param-name>
      <param-value>X-Forwarded-For,Proxy-Client-IP,WL-Proxy-Client-
IP,HTTP_CLIENT_IP,HTTP_X_FORWARDED_FOR</param-value>
    </init-param>
    <init-param>
      <param-name>Forwarded-Host</param-name>
      <param-value>X-Forwarded-Host</param-value>
    </init-param>
    <init-param>
      <param-name>Forwarded-Proto</param-name>
      <param-value>X-Forwarded-Proto, WL-Proxy-SSL</param-value>
    </init-param>
    <init-param>
      <param-name>Forwarded-By</param-name>
      <param-value>X-Forwarded-By</param-value>
    </init-param>
    <init-param>
      <param-name>Forwarded-Port</param-name>
      <param-value>X-Forwarded-Port</param-value>
    </init-param>
    <init-param>
      <param-name>Forwarded-Server</param-name>
      <param-value>X-Forwarded-Server</param-value>
    </init-param>
  </filter>
  <filter-mapping>
    <filter-name>ForwardedFilter</filter-name>
    <servlet-name>ForwardedFilterServletTest</servlet-name>
```

```
</filter-mapping>  
</web-app>
```

The class provided by JEUS for Forwarded headers filter is `jeus.servlet.filters.ForwardedFilter`.

11. Practice

Follow the following steps in this chapter to easily become familiar with JEUS. The following example assumes that the JEUS server is named "server1". Change this to the actual name of the JEUS server that is installed.

1. Verify that JEUS is installed properly and that the system paths and variables are set properly. Ensure that the directory "JEUS_HOME/bin/" is included in the system path.



1. In this guide, "JEUS_HOME" refers to the root of the JEUS installation directory.

(For example, "/home/user/jeus9")

2. For more information about installing JEUS, refer to "JEUS Installation and Getting Started Guide".

2. At least one connection must exist on a web engine of a managed server. Refer to [Web Connection Management](#) for more information about configuring web connections.

An HTTP listener with a port set to '8088' can be added as described in [HTTP Listeners](#).

3. After completing the configurations, enter the following command to execute the JEUS launcher. The arguments '-u' and '-p' are used to specify the administrator name and password that were set during the installation.

The following example assumes that the administrator name and password have been set to 'jeus'.

```
startMasterServer -domain domain1 -server server1 -u jeus -p jeus
```

4. Open a different console window and execute '`jeusadmin -u jeus -p jeus`'. (If you omitted '-u' and '-p' values, enter '`login -u jeus -p jeus`' to log in).

```
$ jeusadmin
JEUS 9 Administration Tool
To view help, use the 'help' command.
offline>login -u jeus -p jeus
Attempting to connect to 127.0.0.1:9736.
The connection has been established to JEUS Master Server [adminServer] in the domain [domain1].
[MASTER]domain1.adminServer>
```

5. Execute the following command to list the commands that can monitor and control the web engine.

```
[MASTER]domain1.adminServer>help -g Web
```

[Web]

<code>add-ajp-listener</code>	Add AJP listener.
<code>add-backup-webtob</code>	Add backup WebtoB server.
<code>add-http-listener</code>	Add HTTP listener.
<code>add-response-header</code>	Add an HTTP response custom header.
<code>add-tcp-listener</code>	Add TCP listener.
<code>add-tmax-connector</code>	Add Tmax Connector.
<code>add-valve</code>	add a new valve configuration.
<code>add-virtual-host</code>	Add virtual host.
<code>add-web-cookie-policy</code>	Add the cookie policy configuration.
<code>add-web-encoding</code>	Add web engine charsetset encoding.
<code>add-web-properties</code>	Add web engine properties.
<code>add-webtob-connector</code>	Add the WebtoB Connector.
<code>clear-thread-local</code>	Change the on/off setting for clearing ThreadLocal
<code>clear-web-statistics</code>	Resets the web engine statistics.
<code>list-session</code>	Show session list sorted by idle time.
<code>modify-jsp-engine</code>	Modify JSP engine configurations.
<code>modify-response-header</code>	Modify the HTTP response custom header.
<code>modify-session-configuration</code>	Modifies the session configuration.
<code>modify-tmax-connector</code>	Modify the thread pool number of the tmax-connector.
<code>modify-virtual-host</code>	Modify the access log format of a virtual host dynamically. The access log must be enabled.
<code>modify-web-cookie-policy</code>	Modify the cookie policy configuration.
<code>modify-web-encoding</code>	Modify the web engine charsetset encoding configurations.
<code>modify-web-engine-configuration</code>	Modify some parts of the web engine configuration dynamically.
<code>modify-web-listener</code>	Modify the thread pool number of the web listener (<code>http-listener</code> , <code>tcp-listener</code> , <code>orajp13-listener</code>).
<code>modify-web-properties</code>	Modify web engine properties.
<code>modify-webtob-connector</code>	Modify the thread pool number of the webtob-connector.
<code>notify-auto-scale</code>	Notify completed.
<code>reload-web-context</code>	Forcibly reloads the servlet context.
<code>remove-backup-webtob</code>	Remove backup WebtoB server.
<code>remove-response-header</code>	Remove the HTTP response custom header.
<code>remove-session</code>	Remove time exceeded session.
<code>remove-tmax-connector</code>	Remove the tmax-connector.
<code>remove-valve</code>	Remove the valve configuration.
<code>remove-valve-property</code>	Remove the property of the valve.
<code>remove-virtual-host</code>	Remove virtual host.
<code>remove-web-cookie-policy</code>	Remove the HTTP cookie policy configuration.
<code>remove-web-encoding</code>	Remove the web engine charsetset encoding configurations.
<code>remove-web-listener</code>	Remove a web listener (<code>http-listener</code> , <code>tcp-listener</code> , or <code>ajp13-listener</code>).
<code>remove-web-properties</code>	Remove web engine properties.
<code>remove-webtob-connector</code>	Remove webtob-connector.
<code>resume-web-component</code>	Temporarily resumes the given component (servlet element of context or given web-connection name).
<code>set-valve-property</code>	Set property of the valve. If the corresponding key does not exist, a key=value pair is added to the valve.

<code>show-request-processing-flow</code>	Shows the request processing flow of mapped URL patterns and the specified host name.
<code>show-session-configuration</code>	Shows the session configuration.
<code>show-session-server-backup-table</code>	Shows the session server backup table
<code>show-web-engine-configuration</code>	Show web engine configurations, including the monitoring period and access-logs.
<code>show-web-statistics</code>	Shows the web engine statistics.
<code>show-webtoB-connector</code>	Show the WebtoB Connector Information.
<code>suspend-web-component</code>	Temporarily suspends the servlet.
<code>precompile-jsp</code>	Precompile JSP files for a deployed web module. Connect to JEUS Master Server or a server.

To show detailed information for a command, use 'help [COMMAND_NAME]'.
 ex) help connect

6. Execute a web engine control or monitoring command. For more information about web engine commands, refer to "Web Engine Commands" in *JEUS Reference Guide*.

If a problem occurs in the previous steps, check and adjust the JEUS environment configurations. To find the source of the problem, refer to the log records in the JEUS manager console log.



For more information about configuring a JEUS environment, refer to "JEUS Installation and Getting Started Guide" and "JEUS Server Guide".