

Session Management 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 (TM, ®).

Open Source Software Notice

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

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

Document History

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

Contents

1. Session Tracking	1
1.1. Overview	1
1.2. Session Tracking Structure	1
1.3. Session Tracking Behaviors	3
1.3.1. Session Tracking Behaviors in Web Engines	3
1.3.2. Session Tracking in Clustered Environments	4
1.4. Session Tracking Modes	11
1.5. Sharing Sessions between Contexts	12
1.6. Configuring Session Tracking	12
1.6.1. Session Configuration	13
1.6.2. Session Server Configuration	16
1.7. Tuning Session Tracking	17
1.8. Monitoring Sessions	17
2. Session Servers	18
2.1. Overview	18
2.2. Basic Concepts	18
2.2.1. Central Session Servers	18
2.2.2. Distributed Session Servers	19
2.3. Server Structure	19
2.3.1. Central Session Servers	19
2.3.2. Distributed Session Servers	20
2.4. Server Process	22
2.4.1. Central Session Servers	22
2.4.2. Distributed Session Servers	22
2.5. Main Features	26
2.5.1. Duplicate Login Prevention	26
2.5.2. Failback	28
2.6. Session Cluster Modes	30
2.6.1. Default Session Cluster Mode	30
2.6.2. Domain-Wide Session Cluster Mode	30
2.6.3. Session Storage Scope Cluster Mode	31
2.7. Session Server Configuration	31

1. Session Tracking

This chapter describes the basic concepts of session tracking and the definitions of session, session ID, session cookie, and URL rewriting. This chapter will also describe the implementation and configuration of session tracking in a more complex clustered server environment (distributed environment).

1.1. Overview

In a narrow sense, session tracking is an action that finds a requested session.

In a clustered environment, depending on the scope of session tracking support, session tracking is classified into two types, session tracking through routing and through a session server.

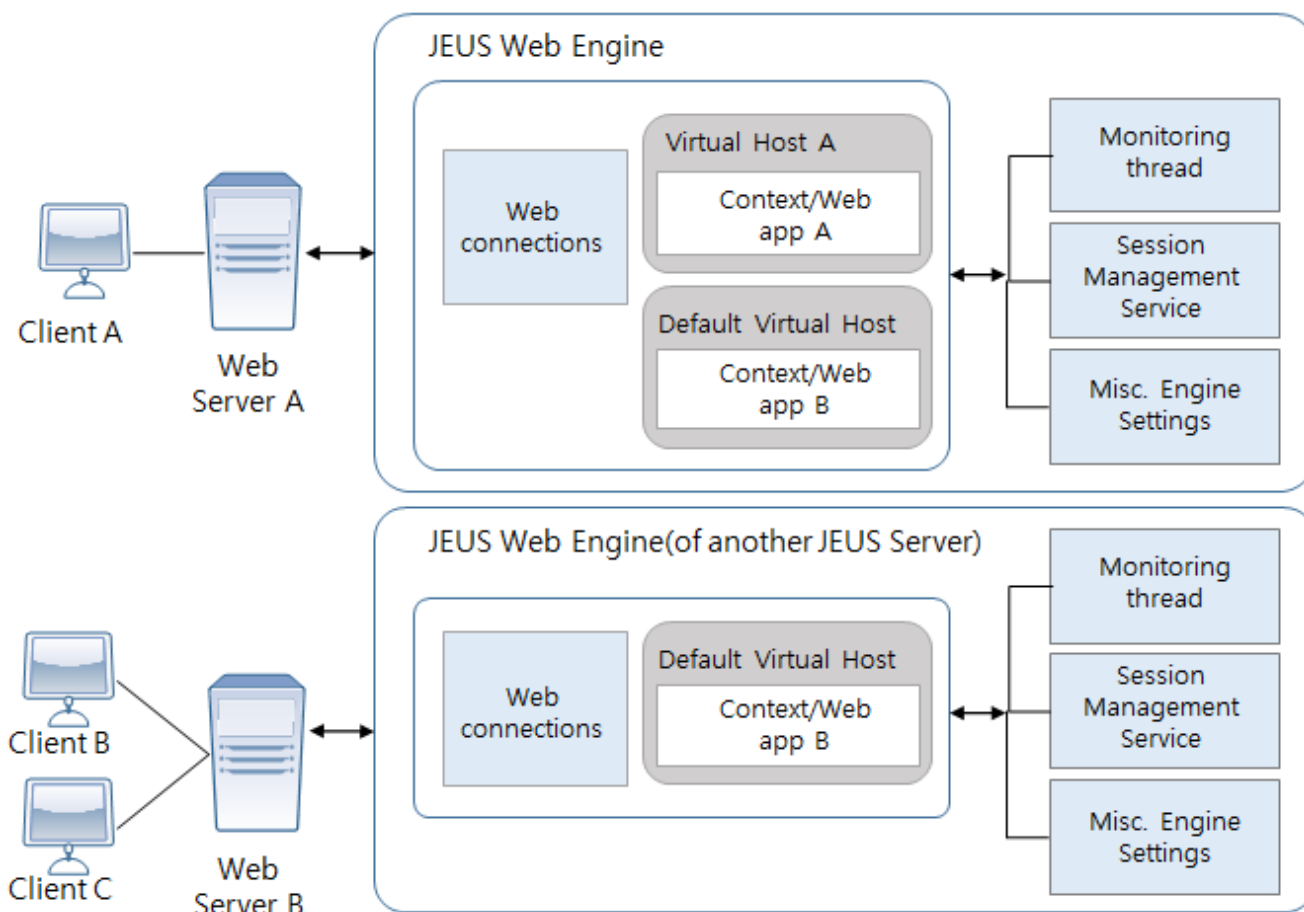
1.2. Session Tracking Structure

This section describes the basic structure of session tracking.



Since this section provides only a very simplified description of session tracking, users who are unfamiliar with the concept need to refer to other documents about servlet and session tracking.

The following diagram shows session tracking and the web engine components related to session management.



Session Parts in the JEUS Web Engine Structure

An HTTP session is a sequence of tasks related to HTTP requests from the same client (web browser). When multiple clients send requests over HTTP, since there is no unique "Client ID" in the header, web servers cannot differentiate the clients. Web servers cannot track user requests because HTTP is a stateless and connectionless protocol.



Web servers are also a part of session tracking.

The following shows the steps for processing an HTTP request.

1. A client connects to the web server.
2. The client sends a stateless HTTP request.
3. The client receives a response.
4. The HTTP connection is disconnected.

A client ID or continuous session is not included in the HTTP protocol. When the HTTP connection is disconnected or the response is sent, all information about the request is discarded. Thus, an HTTP request is not suitable for complex web applications where users are continuously sending related requests.

To overcome this problem, a special string which is referred to as session ID is added to each HTTP request. When a client makes a request for the first time, this unique ID is created and sent to the client. The session ID is attached to each request for any subsequent requests from the same client.

This allows the web engine to identify the source of each request and maintain conversational state during transaction processing thereby supporting sessions for stateless HTTP protocols.

The session ID is sent either as a cookie or through URL rewriting, which automatically adds the session ID as a parameter in the URL. The session ID can also be stored in a hidden field of an HTML form.

1.3. Session Tracking Behaviors

This section describes session tracking in JEUS web engines and in clustered environments.

1.3.1. Session Tracking Behaviors in Web Engines

JEUS web engine supports URL rewriting and cookies, which are enabled by default, for session tracking. The cookie that contains the session ID is called a session cookie.

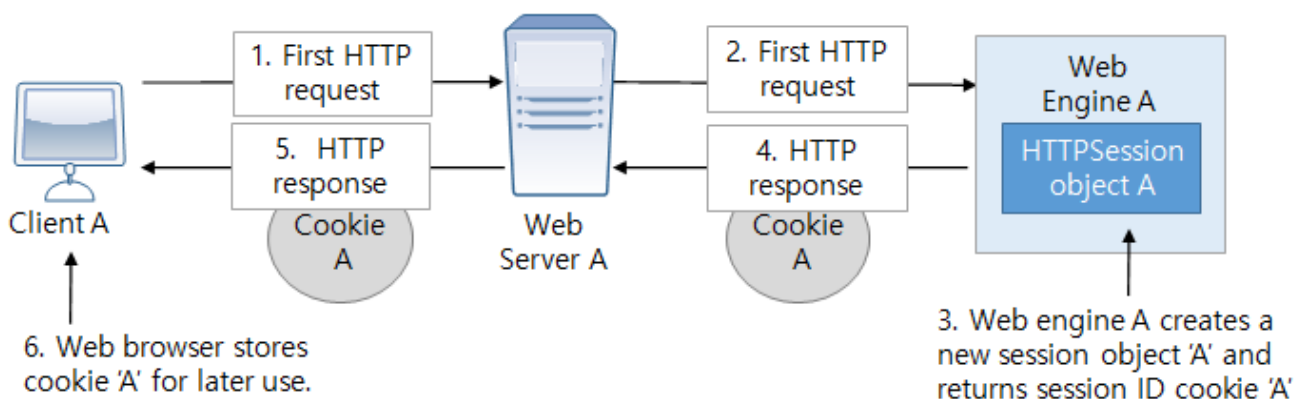
In web engines, a single session is an instance of the HTTP session class. The instance is connected to the session ID of a session cookie or the URL parameter from URL rewriting. By default, HTTP session objects exist in the web engine that created them. They contain user data such as preferences or list of products for purchase.



Both URL rewriting and cookies can be used for session tracking. The session ID is included in the URL link of an HTML page, but session cookies are included separately in the cookie header.

Session cookies are used when a client requests a resource that is managed by the JEUS web engine.

The following describes the process of a web engine creating a session cookie.



Creating a Session Cookie

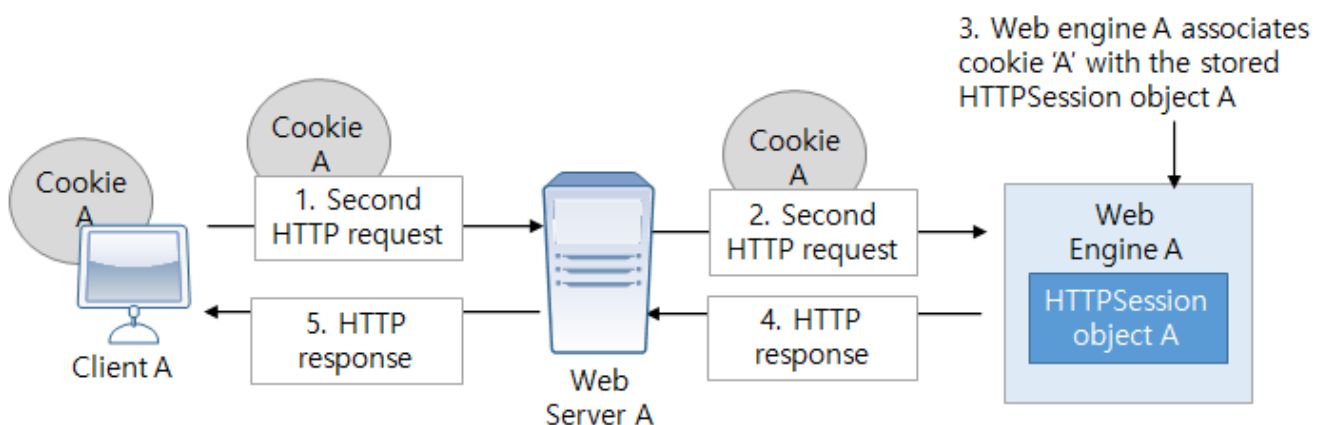
1. A client makes an initial request to the web server.
2. The web server sends the request to the web engine.
3. The web engine creates an HTTP session object for the client and a session cookie that contains

the session ID of the HTTP session. The session ID is used to retrieve the HTTP session object for any subsequent requests from the client.

4. The response data and session cookie are sent to the web server.
5. The session cookie and the response are sent to the client's browser, and the HTTP connection is disconnected.
6. The session cookie that contains the session ID is saved in the client's browser.

Now, the client can include the session cookie when sending requests to the same web server. The web engine identifies the client by the cookie's session ID and can retrieve the client's HTTP session object.

The following is a description of this process.



Sending a Second Request using the Session ID

1. The client sends another request to the same web server by attaching the session cookie previously received from the web browser.
2. The web server receives the request with the session cookie and just like in the first request, sends the request to the same web engine.
3. The web engine receives the request and session cookie. Then, it finds the HTTP session object corresponding to the session ID of the session cookie from its own memory. Then, the web engine uses the HTTP session data to process the request.
4. The response data and session cookie are sent to the web server.
5. The HTTP response is sent to the web browser and the HTTP connection is disconnected. The session cookie has to be included in the response only when the initial connection is created.

1.3.2. Session Tracking in Clustered Environments

[Session Tracking Behaviors in Web Engines](#) describes session tracking in a simple scenario where a client, web server, and web engine are connected. In a real-world environment, there are many situations where this type of simple structure is not sufficient for session tracking. Load balancing and clustering must be implemented in order to handle a large volume of requests. For detailed information about clustering, refer to "Configuring Web Server Load Balancing" in JEUS Web Engine Guide.

Extra caution is required when forming and configuring session mechanisms in web server clusters. There are three major issues to consider when managing sessions in a distributed cluster.

1. How will the request that includes the session cookie be sent to the web engine that first requested it?
2. How can HTTP sessions that were created in one engine be used in other engines so that every engine can process restricted requests?
3. How is session data backed up in case the web engine goes down due to a failure?

The first issue can be handled using **Sticky Session Routing**, and the rest can be handled using **Session Servers**. The following section discusses these issues and their solutions in detail.

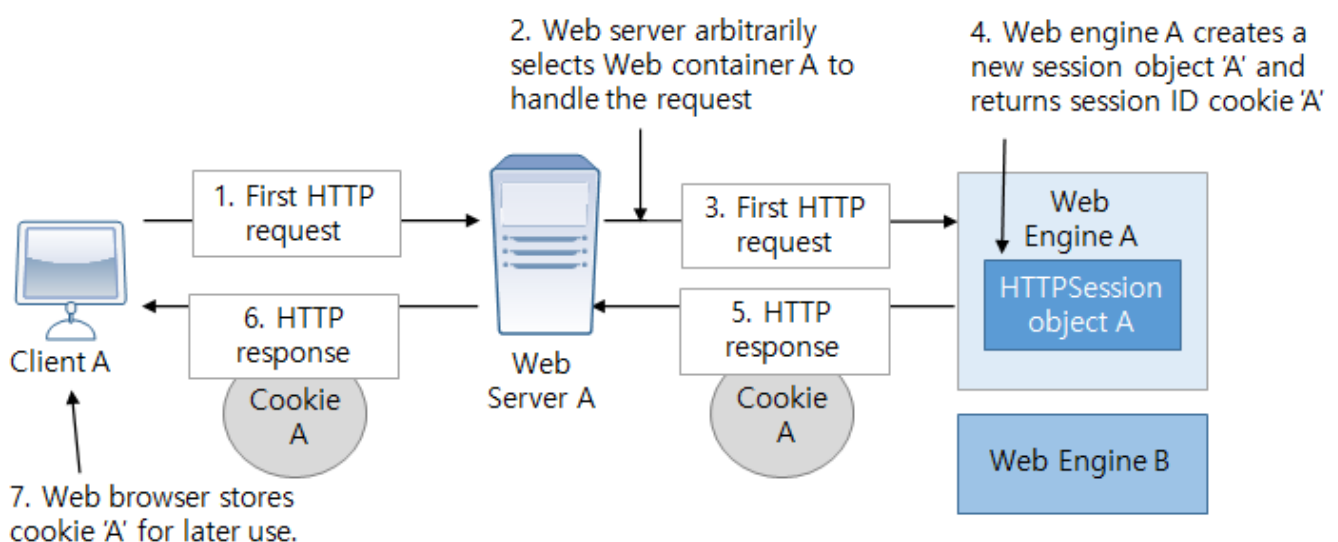


The first and the second issues deal with the same problem, but use different approaches to resolve it, sticky session routing and session server.

Sticky Session Routing

Sticky session routing attaches the ID of the engine where the session was created or where the session is saved to the end of the session ID in a clustered environment. Because the engine ID is included in the cookie, the web server can use the ID to identify and request for the cookie. This increases the efficiency of retrieving a session from the web engine.

The following is an example of processing a client request when two web engines, A and B, are connected to a single web server.



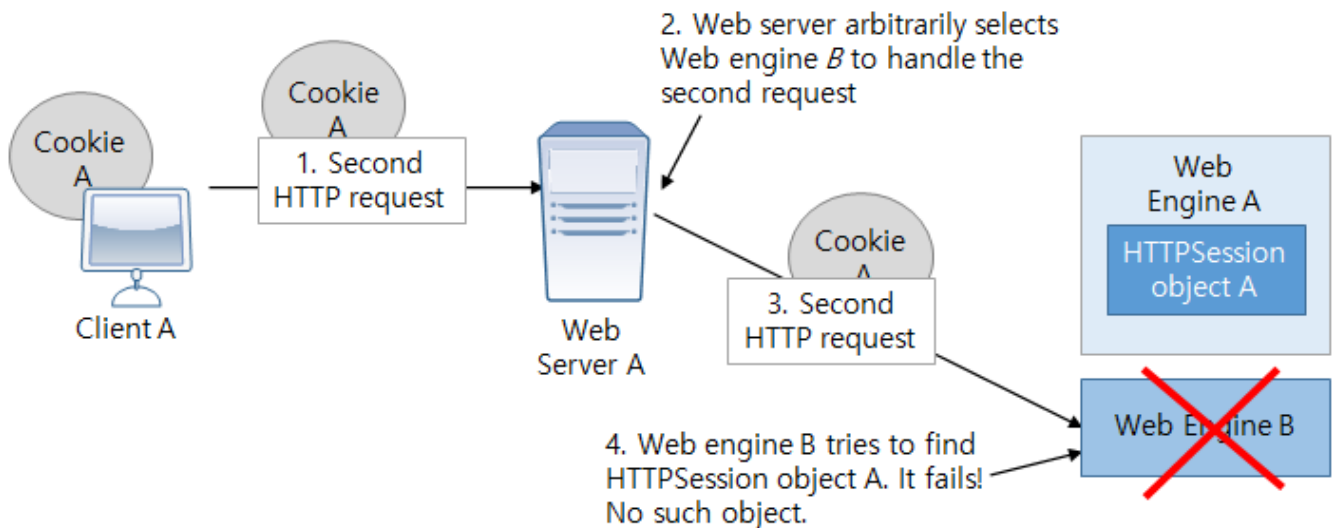
Using Session ID Cookie to Create a Session Using Two Web Engines

1. A client makes an initial request to the web server.
2. The web server arbitrarily selects a web engine to send the request to. Web engine A is selected in this example.
3. The request is sent to Web engine A.
4. Web engine A creates an HTTP session object and returns a session ID cookie with the response.

This ID is used when the HTTP session object handles future requests from the same client.

5. The web engine sends the response, and the session cookie is returned to the web server.
6. The session cookie and the response are sent to the client's browser, and the HTTP connection is disconnected.

The first request is processed successfully, but a critical error occurs when the same client makes a second request.

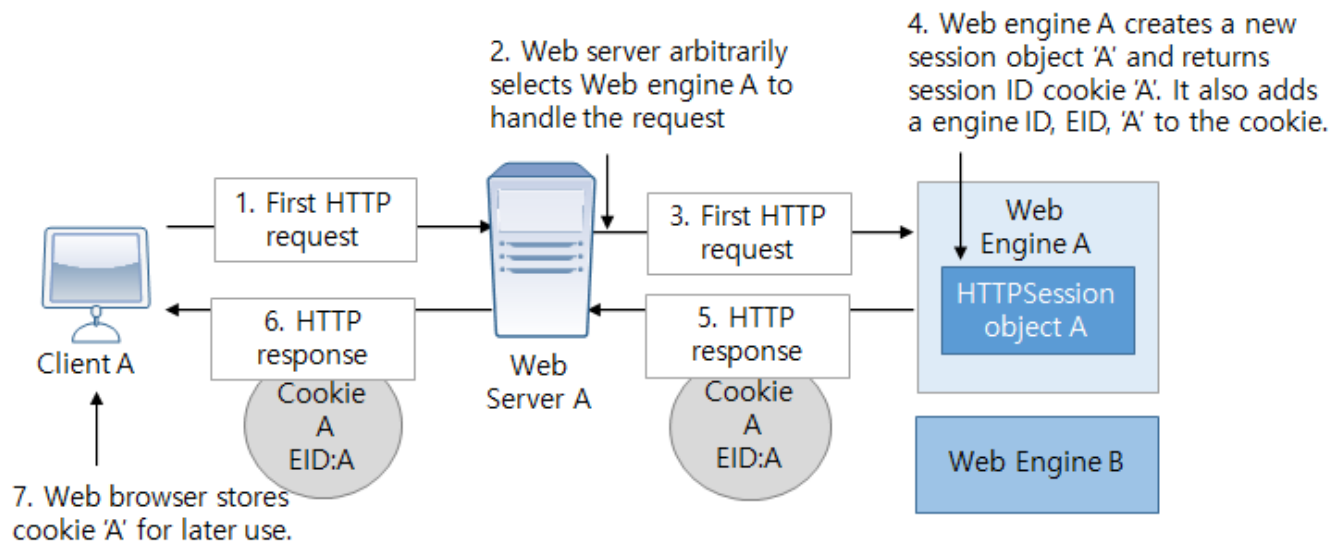


Deleting a Session When Sticky Session Routing is not Used

1. The client makes another request to the same web server. This time, the session cookie that was returned for the first request is sent with the request.
2. The web server accepts the request and arbitrarily selects one of the two web engines. In this example, Web engine B is selected.
3. The request and the session cookie are sent to Web engine B.
4. Web engine B receives the request and the session cookie. The web engine attempts to retrieve the HTTP session for the cookie, but there is no corresponding object. Web engine B cannot maintain the client session and either creates a new session or returns an error message, neither of which is recommended.

As shown in the previous figure, load can be distributed by increasing the number of engines that perform the same service. However, there is the problem of not being able to find the session information. To resolve this problem, each request must be properly routed to the session of the web engine that originally created the HTTP session object. Such routing can be accomplished by adding the web engine ID to the session cookie.

The following diagram shows this process.



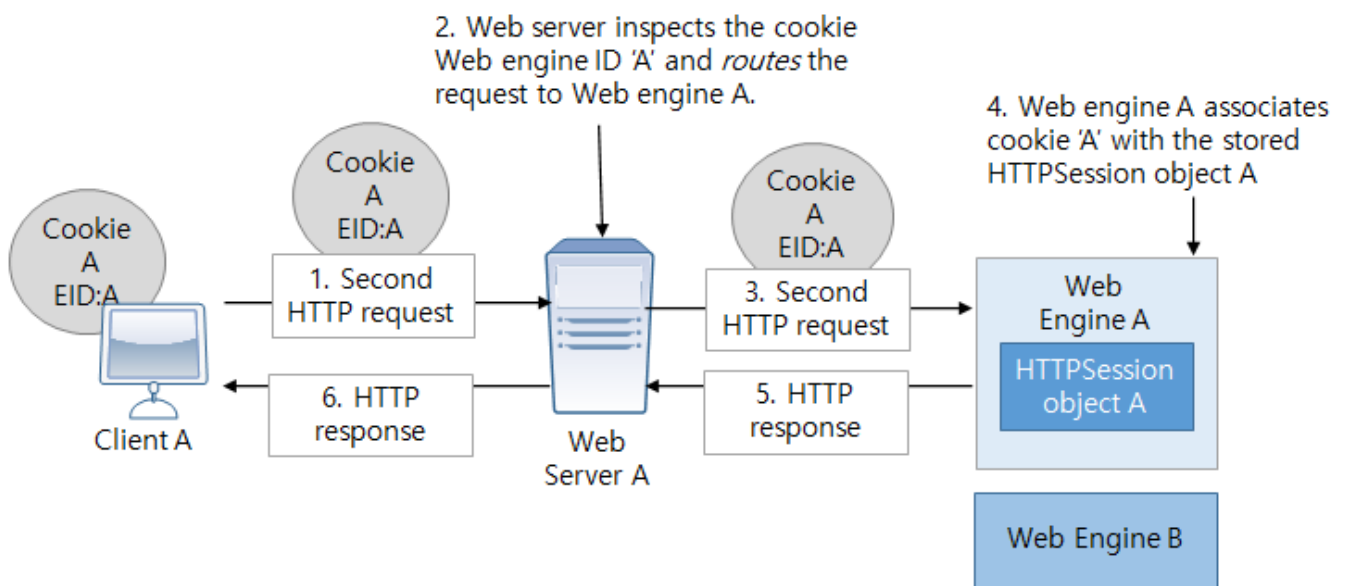
Adding Web Engine ID to the Session Cookie

1. A client makes an initial request to the web server.
2. The web server arbitrarily selects a web engine. Web engine A is selected in this example.
3. The request is sent to Web engine A.
4. Web engine A creates an HTTP session and a session ID cookie, and inserts the web engine ID (EID) into the cookie.
5. The response and session cookie are returned to the web server.
6. The response and session cookie are returned to the browser.
7. The session cookie, which includes the web engine ID (EID), is saved in the browser.



JEUS 6 uses the actual engine ID. In JEUS 9, the ID is encoded.

The following diagram shows the sticky session routing process.



Sticky Session Routing

In the second request, the web server finds the session cookie and the engine ID (EID). The web server routes the request to the original web engine, which has the HTTP session object.



Web servers other than WebtoB require the `mod_jk` module to be installed so that the engine IDs can be recognized. This function is built-in in WebtoB. For detailed information about the installation of `mod_jk`, refer to "Configuring Web Server Load Balancing" in *JEUS Web Engine Guide*.

To use sticky session routing, every web server must be connected to every web engine. This is because when a load balancer is used, the session might get disconnected if the web server that received the request cannot connect to the correct web engine. If the load balancer independently supports session routing, it is not necessary that every web server be connected to every web engine. For example, if WebtoB is used with the load balancer, then session routing is available even if the servers are not completely connected to each other.

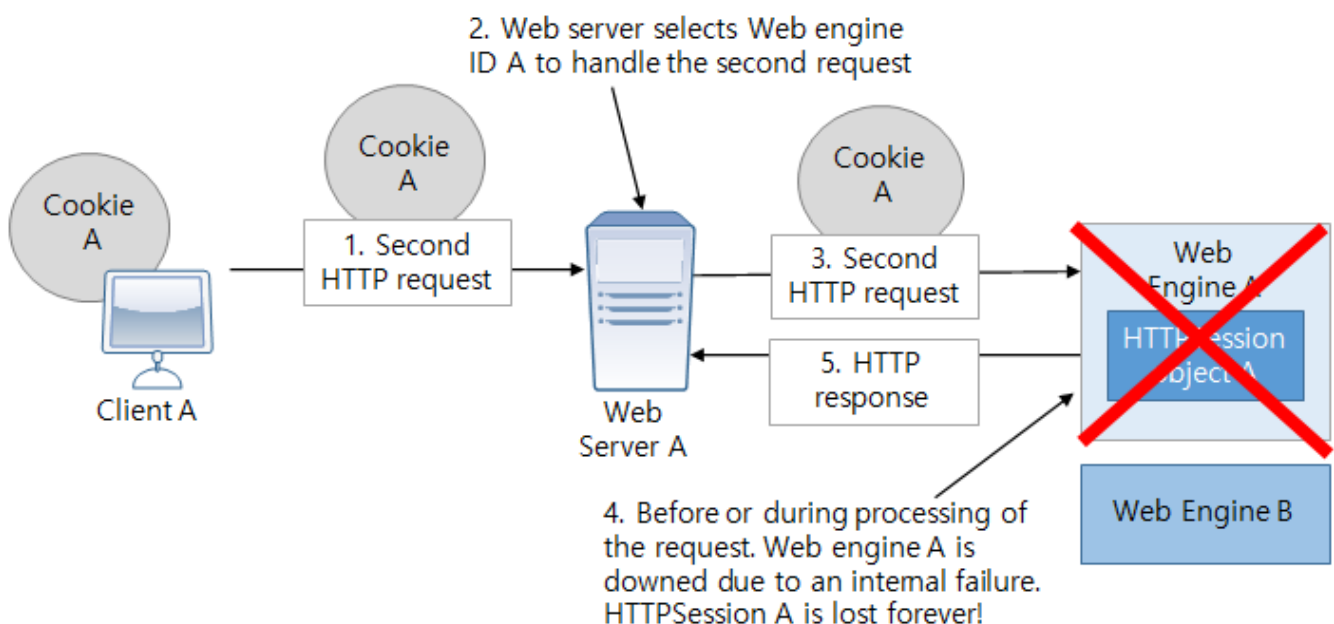
Sticky session routing controls routing only. If the routing operates effectively, it prevents the creation of duplicate sessions as well as duplicate saves, thereby improving performance. However, the routing does not force a request to be sent to a specific web engine. If a target engine has a failure or is under heavy load, a request is sent to another engine. That is, a complete cluster environment cannot be configured just with sticky session routing.

Session Servers

Session servers are more powerful than sticky session routing.

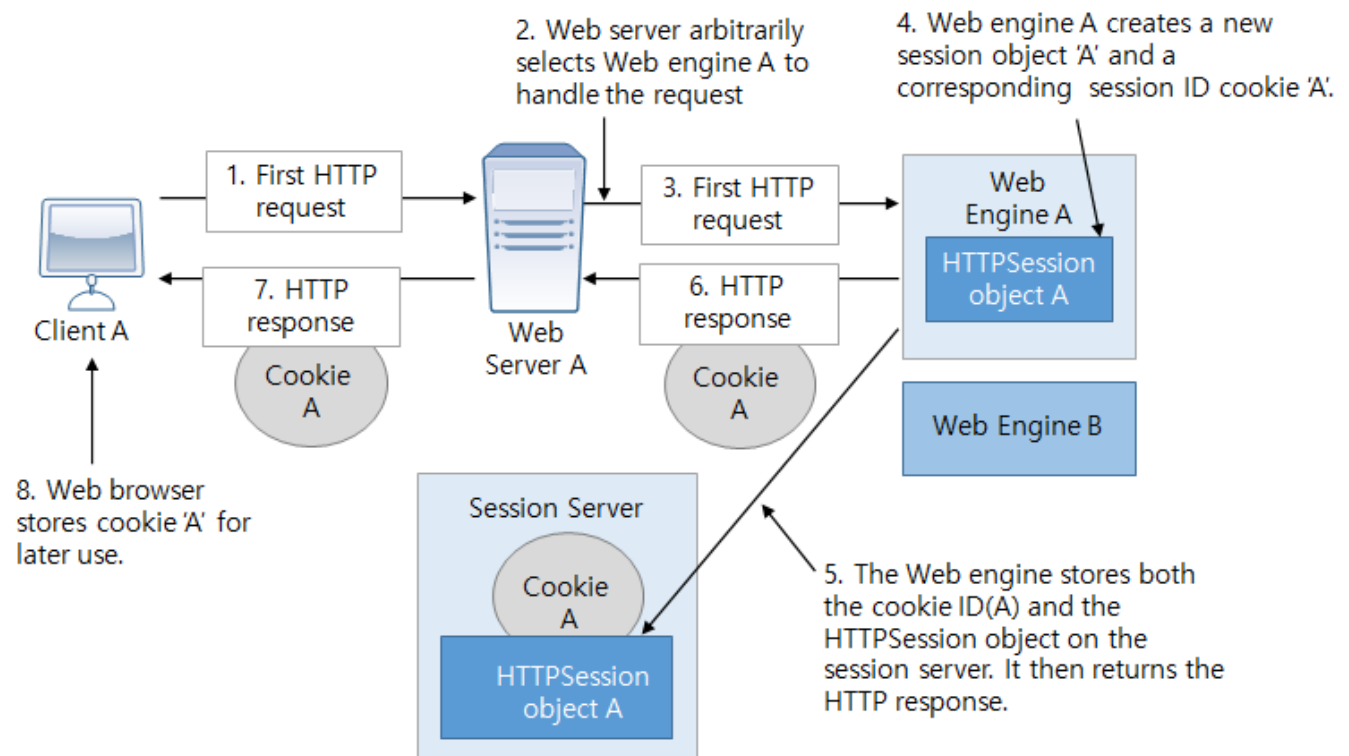
When using session servers, it is not necessary to connect every web engine to every web server in a cluster. Session data in a cluster is backed up by session servers, and so even if a web engine fails, the session data is saved and another web engine can handle the request.

The following diagram shows what happens when a web engine fails. In this scenario, all session data in the web engine is lost.



Web Engine Failure

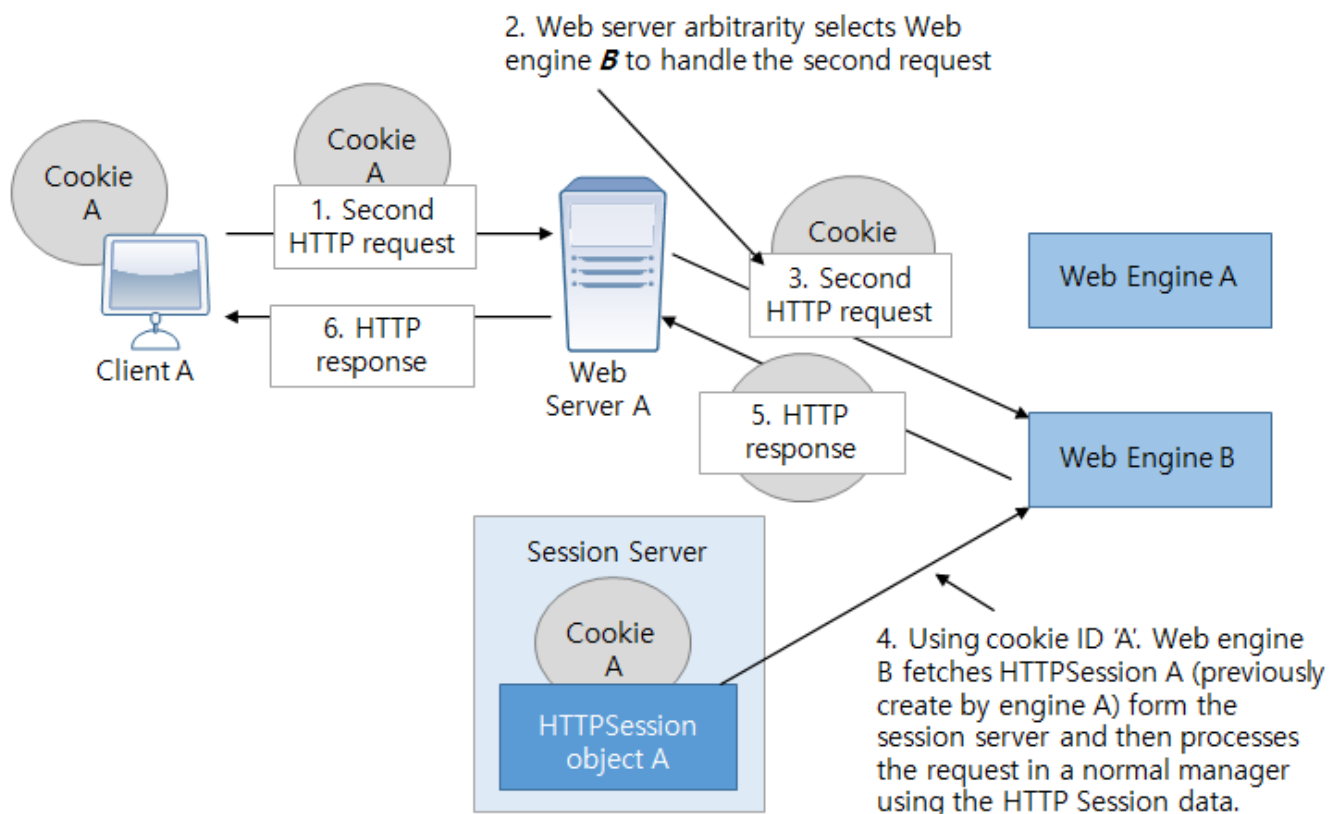
To maintain the sessions when a web engine fails, session servers are added to the cluster. When session servers are used, a client's first HTTP request is handled as in the following figure.



Session Servers - First Request

1. A client sends a request to the web server.
2. The web server arbitrarily selects Web engine A in the cluster to handle the request.
3. The web server sends the request to Web engine A.
4. The web engine creates an HTTP session object and a session cookie. This ID is used to retrieve the HTTP session object for future requests from the same client.
5. When the request is handled, the web engine saves the HTTP session object and the session ID in the session server.
6. The response data and session cookie are sent to the web server.
7. The session cookie and the response are sent to the client's browser, and the HTTP connection is disconnected.
8. The browser saves the session cookie.

Even if Web engine B is later selected by the web server to handle Client A's request or if Web engine A fails, the session data can be retrieved from the session server.



Session Servers - Second Request

As shown in the previous figure, the session server's role is common storage. Web engines A and B are separate storage mediums and are constructed in such a way that they cannot access each other's data. However, they can use the session server as a common storage where session data is saved and retrieved for session sharing.

Conceptually, we assume here that the engine A and engine B are separate storages while the session server provides a common storage. But the web engines A and B could also be common storages themselves like the session server. Here, the 'common' storage means that they can be accessed via network, or other interfaces.

To provide safer session data storage, backup servers are used for fault tolerance. The backup server is automatically selected from the cluster and is automatically updated when a server fails or a new server is added.

Session servers provide the benefit of being able to continuously maintain sessions even if a web engine fails. Like central session server in previous versions of JEUS, session servers contain separate storage and interface that can be accessed by any web engine. This allows sessions to persist even during failure.

However, a centralized session server cannot maintain high performance in a large scale clustered environment due to the overhead caused by centrally storing all web engines' sessions. Also, a single server solution is not appropriate for a distributed environment.

For this reason, JEUS uses distributed session server. Distributed session servers are designed to produce increased performance in clustered environments.

In a distributed session server, each web engine has its own session server. It uses session routing

by default although session routing is not required it helps increase performance. Like session servers, the session data backup can be configured which allows sessions to persist even if the web engine fails.

A distributed session server is automatically included in a server cluster. If a server cluster is configured, the contexts of all servers included in the cluster share and maintain sessions through a distributed session server without separate configurations. For detailed information about distributed session server configuration, refer to [Distributed Session Server Configuration](#).



JEUS 9 supports its own centralized session server as well as Redis and Hazelcast as session servers. For information about how to configure them, refer to [Distributed Session Server Configuration](#).

Mixed Mode

Mixed mode is a combination of session routing and session server. The following table discusses the benefits and limits of session routing and session server.

	Session Routing	Session Servers
Pros	Fast because it accesses the session objects in the web engine.	All session objects are saved in the session server. Session data is not lost when a web engine fails.
Cons	Session data is lost when a web engine fails.	Session data must be saved and retrieved from the session server, which is slower than session routing.

Mixed mode provides the advantages of both methods. When the methods are combined, session objects exist on every session server and in the web engine that created them. The web engine only retrieves or modifies session objects from the session server when it needs them. Mixed mode uses only about half the network bandwidth of using only session servers. Mixed mode also guarantees the safety of all session data. Mixed mode session management is recommended for clustered environments.

1.4. Session Tracking Modes

A session tracking mode can be selected in Servlet 4.0.

- Cookie Mode

This mode uses cookies to track sessions.

Cookies are closely connected to the browser and this is the most common way to track sessions. However, since cookies are dependent on browsers, sessions may not be maintained in situations where cookies are not used. Session tracking is possible as long as cookies are used according to their specifications.

- URL Mode

URL Rewriting mode is used when cookies cannot be used or when the browser does not support cookies.

This is the same approach used for URL rewriting in JEUS 6. This saves session data in the URL without using cookies. This allows sessions to be maintained without using a web browser that supports cookies.

To use this mode, a special tag is required in the jeus-web-dd.xml deployment descriptor. When the `<tracking-mode><url>` tag is used, the session ID is saved using URL Rewriting which allows session tracking across multiple domains.

- SSL Mode

SSL Mode is restricted to SSL connections, and is recommended for situations where a restricted transfer of sessions is desired. This mode can be used to store secure data in a session. If SSL mode is used, sessions can only be transferred through an SSL connection.

1.5. Sharing Sessions between Contexts

Sessions are usually managed by the same context, but they can be shared among different contexts.

Using session servers does not mean that all sessions are shared. To share session data between contexts, configure the **scope** for the distributed session configuration. The scope must be configured for session sharing between different contexts. For more information about the configuration, refer to [Distributed Session Server Configuration](#).

1.6. Configuring Session Tracking

The following configurations are required to use session routing or session server.

- Sticky Session Routing

Session routing is supported by default and thus requires no additional configuration.

Encoding information can be additionally configured.

Value	Description
BASE64	Encodes information by using the BASE64 rule. Used to avoid exposing engine or domain information through the session ID.

Value	Description
BASE64_WITHOUT_PADDING	<p>Encodes information by using the BASE64 rule without PADDING ("=").</p> <p>Used to avoid exposing engine or domain information through the session ID.</p>
RAW	<p>Does not encode information.</p> <p>This is used for debugging because it is difficult to determine which engine sent the request by using encoded information, or under a circumstance where no security issues exist even though an engine name is exposed.</p>

- Session Server Configuration

To use distributed session servers, the servers must be included in a cluster.



1. For detailed information about sticky session routing and session servers, refer to [Distributed Session Server Configuration](#).
2. Session tracking can also be configured using jeusadmin, a console tool. Refer to "set-sessionstorage-scope-session-config" in *JEUS Reference Guide*.

1.6.1. Session Configuration

Many items can be configured for session management. All aspects of a session such as whether to share session objects, session cookie configuration, and timeout can be configured.

The following is how to configure sessions in domain.xml.

1. In domain.xml, go to **[domain] > [server] > [web-engine] > [session-config]**.

These settings define the common session configuration used in the web engine. Settings can be overridden by the context, which has a higher configuration priority than the web engine.

```
<session-config>
  <timeout>10</timeout>
  <max-session-count>-1</max-session-count>
  <reload-persistent>>false</reload-persistent>
  <tracking-mode>
    <cookie>true</cookie>
    <url>false</url>
    <ssl>false</ssl>
  </tracking-mode>
  <session-cookie>
    <cookie-name>JSESSIONID</cookie-name>
    <url-cookie-name>jsessionid</url-cookie-name>
    <version>0</version>
    <path>/</path>
    <max-age>-1</max-age>
  </session-cookie>
</session-config>
```

```
<secure>false</secure>
<http-only>true</http-only>
<same-site>Disable</same-site>
<partitioned>false</partitioned>
</session-cookie>
</session-config>
```

The following describes the configuration tags.

- **Tracking Mode**

Set the method of transferring sessions for session tracking. The following describes three modes of session tracking, which can be used all at once. If not specified, only the cookie-based session tracking will be used by default.

- **Session Cookie**

The standard way to track sessions is to use session cookies in the client response. This configures the cookies in the HTTP header of the response from the web engine. Although cookies are usually created in the web engine, this configuration can be used to create specific cookie data.

The following describes each configuration item.

Value	Description
Timeout	<p>Expiration period of sessions created by the server.</p> <p>The session timeout is usually configured in web.xml, the web application configuration. If the session timeout is already configured in web.xml, this setting is ignored. The session timeout setting in web.xml has the highest priority.</p> <p>The units are in minutes. The default value is 30.</p>
Max Session Count	<p>Maximum number of sessions to maintain in memory. If the number of sessions exceeds the maximum number of sessions, an error will occur.</p> <p>Since the number of sessions for an application must not have any influence on session creation for another application, this maximum number needs to be set for each application.</p> <p>The default value is -1, which means that sessions can be created infinitely.</p>
Reload Persistent	<p>Option to persist session object properties when servlet context is modified and reloaded. Normally in such case all properties are deleted. If not set, the default value is used.</p> <ul style="list-style-type: none">◦ true: Properties of session objects persist after reloading.◦ false: Properties of session objects are deleted after reloading (default value).

Value	Description
Cookie	<p>Cookie-based session tracking.</p> <ul style="list-style-type: none"> ◦ true: Uses cookie-based session tracking. (Default value. Recommended) ◦ false: Does not use cookie-based session tracking. This may affect session maintenance.
Url	<p>URL rewriting-based session tracking.</p> <ul style="list-style-type: none"> ◦ true: Uses URL rewriting for session tracking. ◦ false: Does not use URL rewriting for session tracking. (Default value)
Ssl	<p>SSL-based session tracking.</p> <ul style="list-style-type: none"> ◦ true: Transfers sessions only via SSL during session tracking. This may not be fully functional. ◦ false: Transfers sessions not only via SSL, but also any other types of connections during session tracking. (Default value)
Cookie Name	Name used instead of the standard "JSESSIONID". This is a string, and the default value is JSESSIONID.
Version	<p>Version of the cookie ID.</p> <p>Options are:</p> <ul style="list-style-type: none"> ◦ 0: Netscape cookie (default value). ◦ 1: RFC cookie.
Domain	<p>Domain name sent with the session cookie. Cookies can be accessed only from this domain. The domain name must start with ".". For more detailed information, refer to RFC-2109.</p> <p>This is a string. If not set, the cookie's domain information is omitted.</p>
Path	<p>URL(string) path in the domain where the session cookie is sent. Cookie is sent when there is a request from this URL in the specified domain.</p> <p>If the path is configured as "/examples" and the domain is configured as ".foo.com", the client will only send the cookie to the server when the request location is "www.foo.com/examples". For more detailed information, refer to RFC-2109.</p> <ul style="list-style-type: none"> ◦ If not set: An appropriate path in the web engine is selected. ◦ If set: This value is always included in the cookie. <p>If this setting is set to a value other than the root path "/", take caution when setting this value by considering session sharing characteristics of the applications.</p>

Value	Description
Max Age	<p>Session cookie expiration. When a cookie expires, it is deleted by the client and is no longer sent. The default value is -1, and the unit is in seconds.</p> <p>If set to the default value, the cookie expires when the browser closes according to the browser's life cycle.</p>
Secure	<p>Option to use a secure HTTP connection.</p> <ul style="list-style-type: none"> ◦ true: Cookies are only sent over HTTPS. ◦ false: Cookies can be sent over non-HTTPS connections (default value).
Http Only	<p>HttpOnly property of the session ID cookie.</p> <p>The HttpOnly property is a new function in Servlet 3.0 that prevents cookies from being used by scripts.</p> <ul style="list-style-type: none"> ◦ true: Session cookies can only be used in HTTP requests (default value). ◦ false: Session cookies can be used by scripts.
SameSaite	<p>Option to use a security mode that prevents the session ID cookies from being used for an unintended request (Cross-site request forgery, CSRF).</p> <ul style="list-style-type: none"> ◦ Disable: Does not add the same-site cookie, and complies with the web browser's policy. ◦ None: Allows the cookie to be sent in cross-site requests. ◦ Strict: Disallows the cookie from being sent in cross-site requests (preventing the CSRF attack). ◦ Lax: Only allows cross-site requests that are relatively safe against the CSRF attack. For more information, refer to relevant specifications.
Comment	<p>Description that help users understand the purpose of the cookie.</p> <p>This is not supported in the Netscape Version 0 cookies.</p>



If the scope is configured for inter-context session sharing, the session timeout setting uses the scope's session timeout value.



Each configuration item cannot be changed dynamically, thus the server needs to be restarted to apply the changes.

1.6.2. Session Server Configuration

A cluster is required to use session servers in the domain structure. Configure the servers that are participating in the cluster with domain.xml to use session servers without configuring the contexts.

For detailed information about clustering and configuration of distributed session servers, refer to [Session Cluster Modes](#) and [Distributed Session Server Configuration](#) respectively.

1.7. Tuning Session Tracking

The following guidelines provide the best performance in a clustered environment.

- For better performance and stable operation, always use sticky session routing and session servers together.
- Properly configure and tune web servers.
- Use distributed session servers for sites that are overloaded with user requests.

1.8. Monitoring Sessions

The number of sessions can be determined through basic monitoring of the created sessions.

Sessions can be monitored by using the following method.

- **Using the console tool**

For information about monitoring sessions using the console tool, refer to "show-web-statistics" and "list-session" in *JEUS Reference Guide*.

2. Session Servers

This chapter describes the structure, operation, and configuration of distributed session server that are used for session tracking in clustered environments.

2.1. Overview

Session servers are used to manage or back up client session data. They are especially useful when managing session data from multiple web servers and servlet engines in a clustered environment. Centralized session servers and distributed session servers are used to manage session data.



For versions earlier than JEUS 7, central session servers were used. In JEUS 7 and later versions before JEUS 9, only distributed session servers were used, due to issues related with the centralized system management. From JEUS 9 onwards, centralized session servers can now be used again with the support for Redis and Hazelcast as external session storages.

2.2. Basic Concepts

This section describes distributed session servers in comparison with central session servers, to provide a clear picture of the overall concept.

2.2.1. Central Session Servers

Simply put, a session server is a storage of session objects in JEUS.

A session object represents an HTTP session object for a servlet API. An object like HTTP session maps an HTTP request, which is a stateless protocol, to a temporary data. This temporary data is used by WAS for distinguishing clients in operation from other clients, and this allows retention of information about previous tasks. This concept of Session Tracking is the most important part of a Website.

The problem is, however, that an HTTP session object is stored only in the servlet engine where it was created. This means that in a cluster setup consisting of multiple servlet engines, only that one where the session was created must receive all of the incoming requests from the client. But it is not that requests are always sent to a specific servlet engine, since there are certain cases where, for example, the web server does not support session routing, or the servlet engine shuts down unexpectedly due to an internal error, causing the session object to be lost.

A central session server serves as the hub for gathering and managing all session data in a unified location. In simpler terms, whether session objects belong to a particular client or were generated within a specific servlet engine, they are all directed to utilize the central session server. This arrangement offers two distinct advantages:

- Firstly, there's no longer a need for web server-side session routing.
- Secondly, by employing a primary central session server alongside a backup server, session data can be effectively backed up, contributing to the enhanced stability of the entire system.

In this manner, sessions can be securely maintained even in cases where servlet engines are shut down.

2.2.2. Distributed Session Servers

Distributed session servers provide session clustering with enhanced scalability and safety. They enhance performance in a large scale clustered environment.

Distributed session servers have the same basic functionality as the central session server. In a clustered environment with many servlet engines, they allow sessions to persist even if a sequence of requests from the same client is not handled by the same servlet engine.

Distributed session servers mean that the managing agents of the sessions are distributed which provides high degree of scalability.

The following is a summary of distributed session server's characteristics.

- It is possible to persist sessions in a clustered environment composed of many servlet engines.
- Even if the servlet engine fails while processing a request, other servlet engines can continue to process the request without losing the session.
- Scaling is easy because of the distributed protocol.

2.3. Server Structure

This section compares central and distributed session server structures.

2.3.1. Central Session Servers

A central session server is operated in association with the JEUS Web container, providing session routing or backup capabilities for the client in the Web container. The server consists of Session Manager, Session Cache Memory, and Session Storage. For the session storage, JEUS supports its own centralized session server as well as Redis and Hazelcast.

The following describes those subcomponents of a central session server.

- Session Manager
 - Executes overall controls over engine sessions. Sessions are gathered from the local memory or the session server.
 - The local web engine obtains session objects to use from the Session Manager by using the getSession method. First, it collects cached sessions, and then it retrieves those from the

session storage.

- When all sessions are handled, it updates the session storage.

- **Cached session(Session Cache Memory)**

Session objects that are active or have been used once are stored here for quicker access.

- **Session Storage**

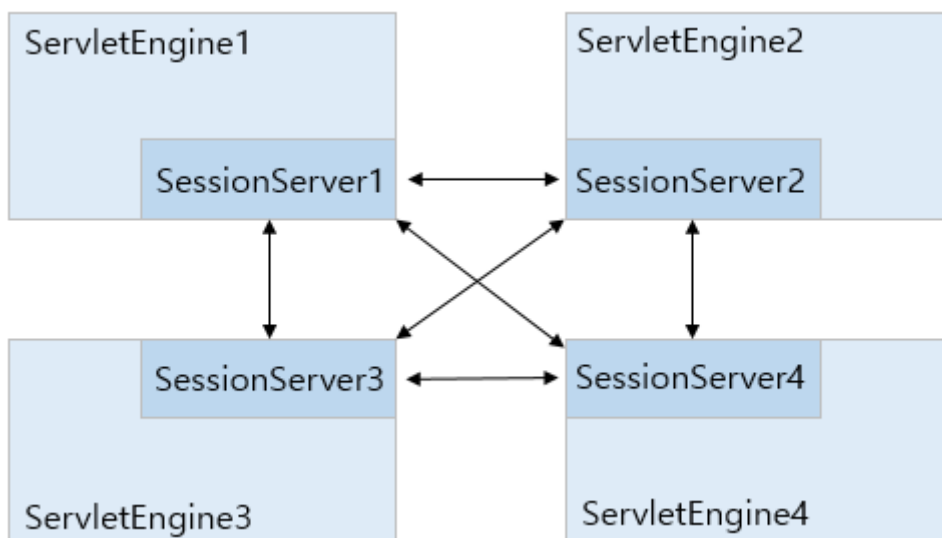
Serves as a session server to store sessions. JEUS's centralized session server, Redis and Hazelcast are supported for the session storage.

2.3.2. Distributed Session Servers

Distributed session servers use a distributed structure in which the servers that manage session objects are distributed to each servlet engine (web engine) or EJB engine.

When using distributed servers, a separate distributed session server exist in each engine of the cluster, and they provide continuous services by communicating with other distributed session servers.

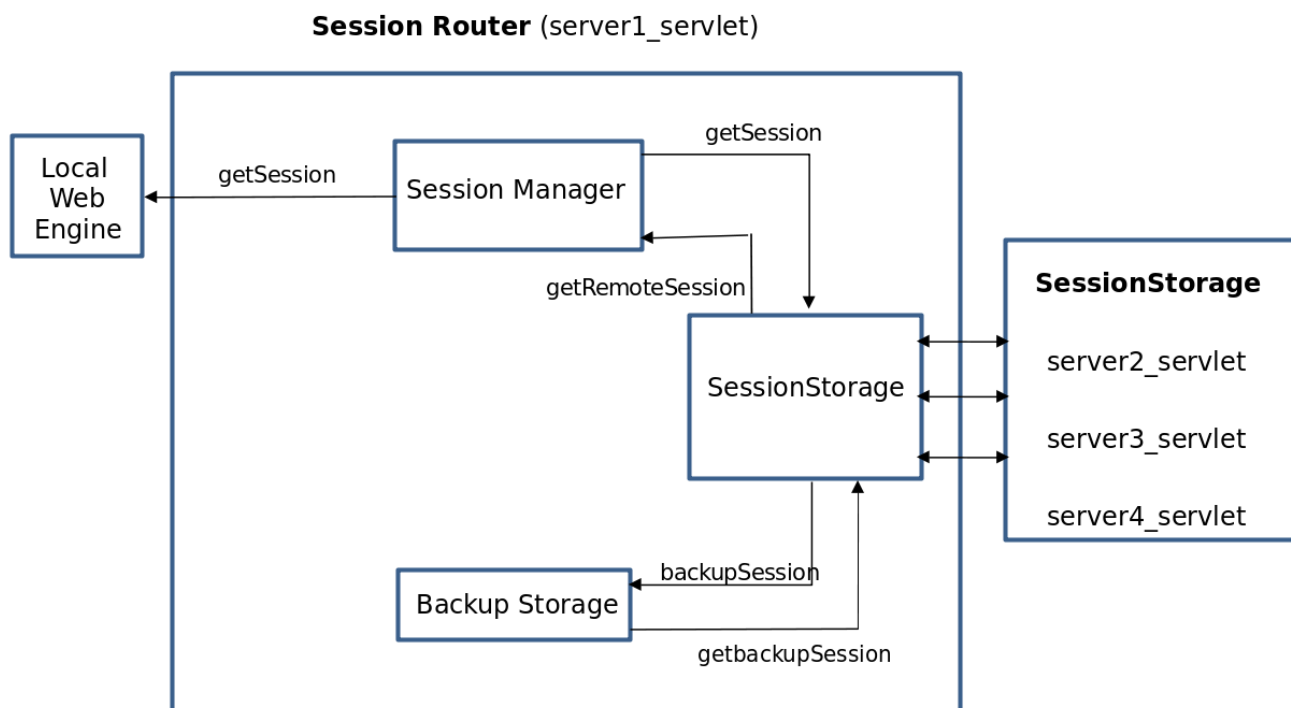
The following diagram shows the session clustering structure with four web engines.



Distributed Session Servers - Structure

The arrows in the previous structure map represent possible socket connections between the distributed session servers. Instead of having all sockets always be connected to each other, they connect and maintain the connection only when they need to communicate with other engines in order to maintain a session.

The following diagram shows the internal structure of a distributed session server, which is a subcomponent of the web engine.



Distributed Session Servers - Internal Structure

• Session Manager

- Executes overall controls over engine sessions. Sessions are gathered from the local memory, files, or other remotely distributed session servers.
- The local web engine obtains session objects to use from the Session Manager by using the `getSession` method. First, it collects sessions from the sticky servers, and then retrieves those from the previous backup servers.
- If a remote backup exists, modified sessions are backed up in memory using the `backupSession` method to a distributed session server that is designated as the backup server in another web engine, through the Cluster Manager.

The '**Backup Level**' setting is used to determine if a session has been modified. A session is considered modified when a session operation, such as `setAttribute`, `removeAttribute`, and `setMaxInactiveInterval`, is called while the 'Backup Level' is set to "all", the '**Backup Level**' is set to "get" and a getter or setter method is called, or the 'Backup Level' is set to "set" and a setter method is called.

For details about this configuration, refer to [Session Server Configuration](#).

• Backup Storage

- Backup Store manages the backup session objects that are periodically sent from another engine's distributed session server. This backup session object provides a replacement session if a failure occurs in the engine that contains the original session.
- It receives the backup sessions from the remote web engine which is designated as a backup, and then saves and manages the backup session using `backupSession` and `getBackupSession` methods.
- When the conditions are met for locally modified session objects, they are backed up to the

distributed session server. Once the request is complete, the modified session objects are immediately backed up so that failover can be performed in case of a system failure.

- **Cluster Manager**

- Dynamic Remote Web Engine is a module that acts as a mediator when a specific operation is performed on a session in the remote web engine.

This engine operates by considering the dynamic nature of the environment. It selects an appropriate backup every time a server is added or removed. This is one of the major changes in JEUS that is not related to environment configuration. The connection to the remote engine is verified through SCF instead of constant pinging to avoid sending repeated error messages for an ongoing error situation.

`getSession()` and `removeSession()` from a remote web engine can be examples.



Previous versions of JEUS only recognizes the backup engine that is fixed in the configuration file. Now, it is designed to allow dynamic changes to the remote engine.

2.4. Server Process

This section compares the behaviors of central and distributed session servers.

2.4.1. Central Session Servers

The Web container is linked to the central session server. When the Web container detects a newly created or modified session object, it stores or updates the object in the central session server. It also retrieves session objects from storage in response to client requests.

2.4.2. Distributed Session Servers

By default, distributed session servers use session routing. Session routing attaches the name of a specific session server to the session ID so that it knows which server contains the session data. Performance is enhanced when session routing is used.

Web servers that support session routing route requests to the web engine that contains the required session object. Session routing works in the same way for distributed session servers. For additional information, refer to [Session Tracking in Clustered Environments](#).

A session key is used for distributed session servers that use session routing.

The following example shows a session key being used in a web engine.

```
<SessionID>.<primary-engine-name>
```

Example) XXX.domain1/server1

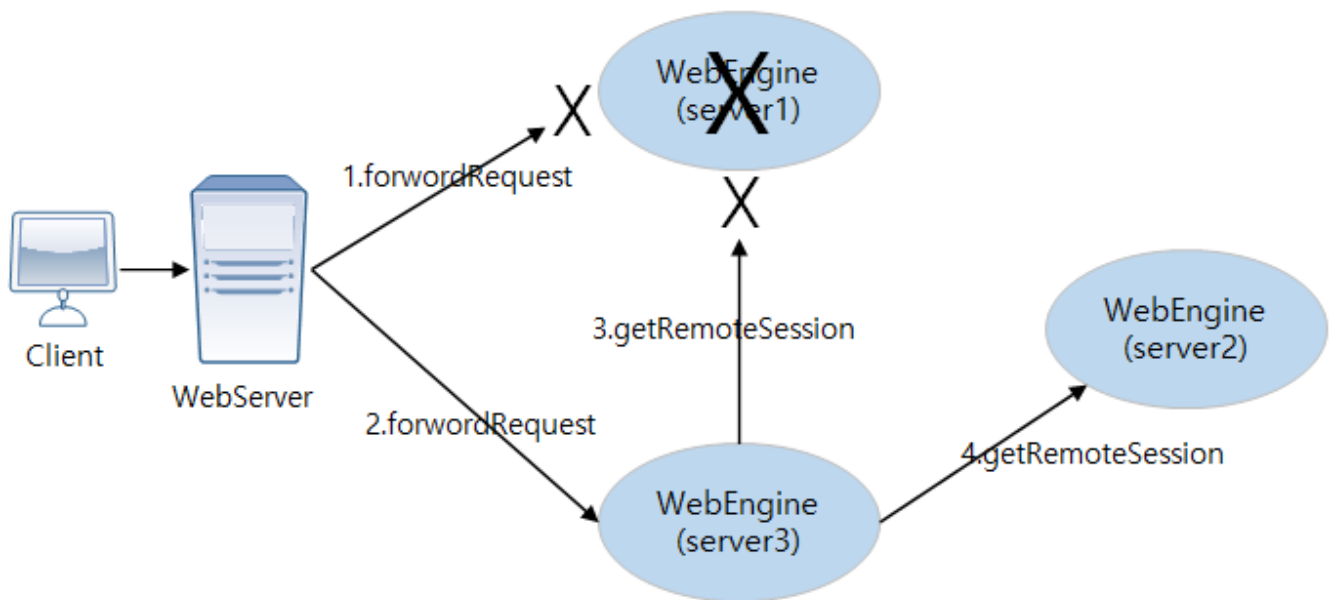
Value	Description
XXX	Session ID. The actual session ID is a random string that is longer in length.
domain1/server1	Routing information. A servlet engine named domain1 is used.

Each engine is assigned a session routing ID for session routing that identifies the web engine when it connects to the web server. The session routing ID is automatically created according to its configuration.

For the three web engines in [Distributed Session Servers - Failover Structure](#).

- WebEngine(server1)
 - Session routing ID: domain1/server1
 - Session routing ID of the backup server: domain1/server2
- WebEngine(server2)
 - Session routing ID: domain1/server2
 - Session routing ID of the backup server: domain1/server3
- WebEngine(server3)
 - Session routing ID: domain1/server3
 - Session routing ID of the backup server: domain1/server1

The following diagram shows the failover structure of session tracking that uses the session routing ID assigned to each distributed session server.



- Request Session Key : XXX/domain1/server1
- Reponse Session Key : XXX/domain1/server3

Distributed Session Servers - Failover Structure

The following describes the failover structure in [Distributed Session Servers - Failover Structure](#).

1. The web server checks the session routing ID and attempts to send the request to web engine server1, but server1 has failed.
2. The web server arbitrarily selects a different web engine and sends the request. In this example, server3 is selected.

server3, which received the request from the web server, checks the session routing ID and determines that server1 contains the session object for the request, and that the backup exists in server2.

3. server3, which is a distributed session server, attempts to retrieve the session object by connecting to server1, but the request fails due to the failure of server1.
4. Because the request failed, server3 attempts to connect to the backup server, server2. Server2 returns the backup of the requested session object to server3.
5. Server3, which has successfully retrieved the session object, processes the client request and sends the response message to the client. A new session key is created and sent, the client's session key is modified, and future requests are forwarded to server3. When a new session key is created, only the session routing ID is changed.

Web servers cannot provide session routing in situations where the request cannot be sent because the web engine has failed or there is no connection to the target web engine.

The following describes the scenario illustrated in [Distributed Session Servers - Failover Structure](#) for distributed session servers where session routing is not used.

1. Without session routing, the session ID does not exist. The web server tries to send a request to an arbitrary web engine. In this example, web engine server1 is selected.

2. The web server detects that server1 has failed and selects another web engine, server3, and sends the request again.
3. Because server3 does not know the routing information of the session, it sends a request to the currently connected distributed session server, server1, but the attempt fails.
4. The distributed session server sends the request to server2, and server2 returns the session.
5. Server3, which received the session, sends the unmodified session key to the client. As a result, the next request may not be sent to server3 that contains the session.



The routing information is given as 'domain1/server1' for the reader, but this information is encoded during operation.

A session is an object used for a request. Session specifications do not guarantee session manipulation in another JVM or thread. The manipulation must be guaranteed by an application, which is not recommended.

If an application needs to send concurrent requests, the requests must be either sent to different sessions or must not use a session. This is required for a normal operation. However, some recent applications are implemented for multiple requests using a shared session.

The following issues may occur when multiple requests are processed using sessions.

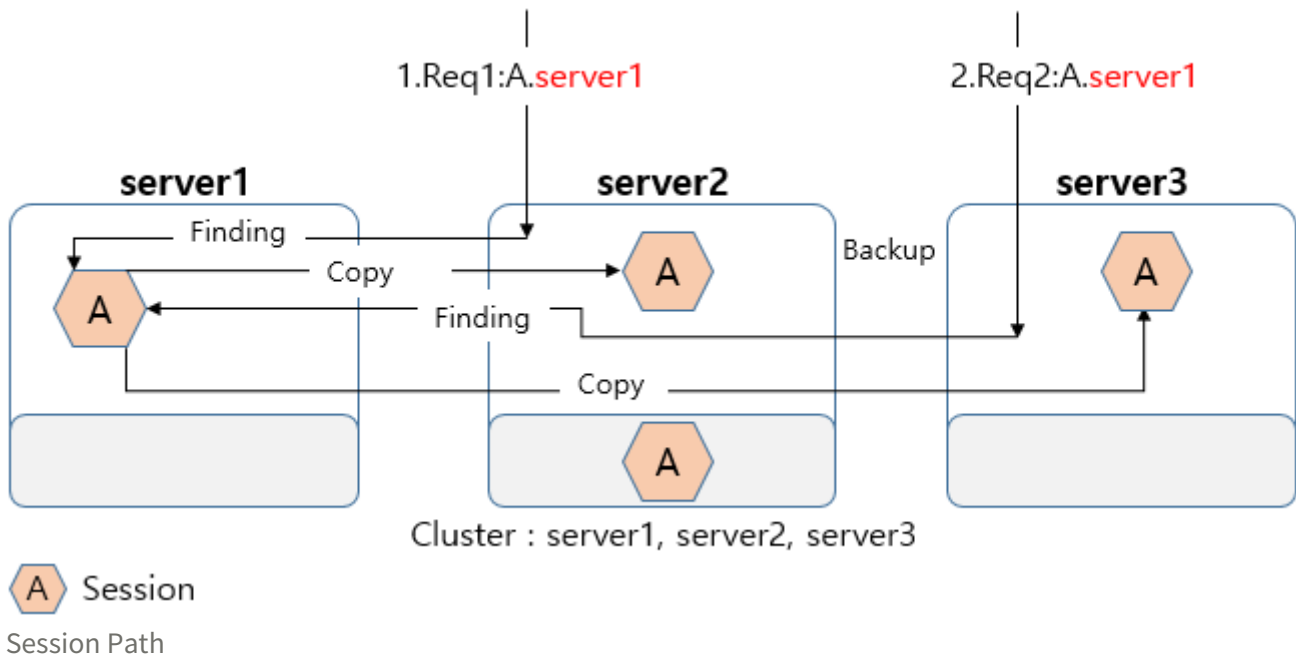
1. Multiple sessions can be created at the same time. Since only one of them is maintained using a cookie, memory is wasted.

This issue occurs when requests are sent to different JVMs or processed at the same time. Since a new session is created by default if there is no available session, concurrent requests result in creating multiple sessions. The sessions are stored in cookies of a user browser. Since the cookies are managed within one scope, the last session information overwrites all the other session information. Therefore, the other sessions cannot be accessed and removed after a timeout period.

2. A session may not be maintained according to a method for maintaining the session in multiple JVMs. This issue is related to sticky session routing and occurs only when requests are sent to multiple servers or web engines. When multiple requests are sent to multiple JVMs at the same time, sessions may be lost according to a session maintaining method.
3. Multiple sessions may not be updated if they are tried to be updated at the same time.

This issue occurs only when requests are sent to multiple servers or web engines like in the above issue. Since one request must be sent to update one session, updating multiple sessions has the similar issue that occurs when creating multiple sessions.

As described above, no issue occurs when concurrent requests are sent to the same server or web engine. However, if an application sends concurrent requests to multiple servers or web engines, sessions are lost as in the above figure. To prevent this, JEUS 9 uses the Copy & Update approach.



Although concurrent requests are sent to multiple servers, sessions are not migrated and copied for local manipulation. Manipulation results are updated in a server with ownership of each session. Disabling migration prevents losing sessions and has no issue when used to simply check session values.



Since a session is a temporary object used to save and manage simple information, it must not be used like a database which exists in all systems. Use a session only to refer to an object value. It is not recommended to use a structure or configuration in which multiple requests are sent to multiple servers concurrently.

2.5. Main Features

This section describes the main features of distributed session server.

2.5.1. Duplicate Login Prevention

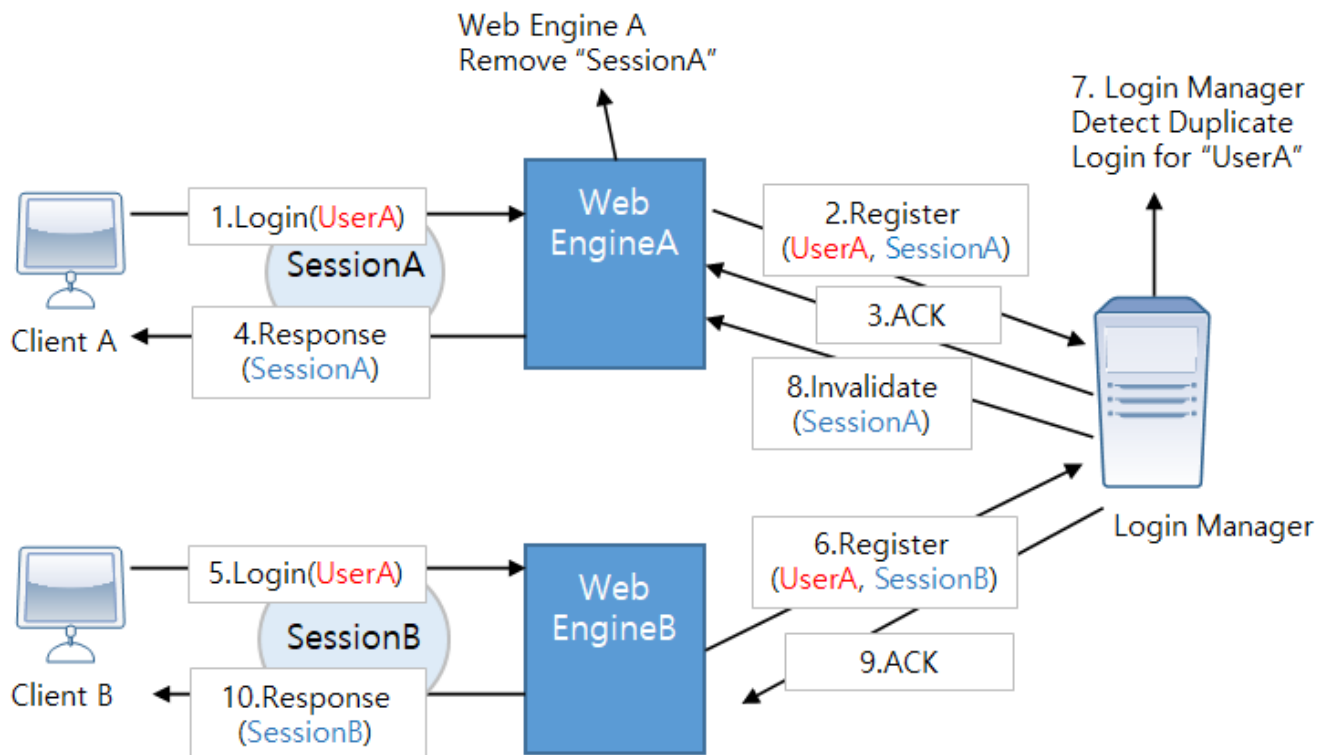
Distributed session servers provide duplicate login prevention function that include the basic duplicate login management functions provided by applications.

Duplicate login prevention of distributed session servers prevents duplicate login using a single application ID. When the same ID is used to log into two different sessions, the existing session is removed to prevent duplicate login.

To prevent duplicate login, the login manager is configured similar to the existing central session server. A server that stores the login information is specified among the servers in the cluster, and a secondary server is configured to prepare for failure.

Duplicate login prevention can be configured for each Session Storage, and each login information is managed by Session Storage. To use this feature, set login-manager to true with the property of Session Storage.

The following figure illustrates the process of duplicate login prevention.



The Process of Duplicate Login Prevention

The following is description of [The Process of Duplicate Login Prevention](#).

1. Client A using SessionA tries to log into Web EngineA with the ID, UserA.
2. Web EngineA registers the login information in the Login Manager.
3. The Login Manager sends ACK for the login to Web EngineA.
4. Web EngineA sends a response to Client A.
5. Client B that is using SessionB tries to log into Web EngineB with the ID UserA.
6. Web EngineB registers login information in the Login Manager.
7. The Login Manager detects a duplicate login.
8. The Login Manager tells Web EngineA to discard the existing session SessionA and Web EngineA deletes the session.
9. The Login Manager sends ACK for the login to Web EngineB.
10. Web EngineB sends a response to Client B.



Since login is an event in an application, there are restrictions for sending a login event from the JEUS Login Manager. JEUS Login Manager sets the method of retrieving LoginID with the user-info in the property of Session Storage. If removeAttribute is executed with the same key, logout occurs and the invalidated

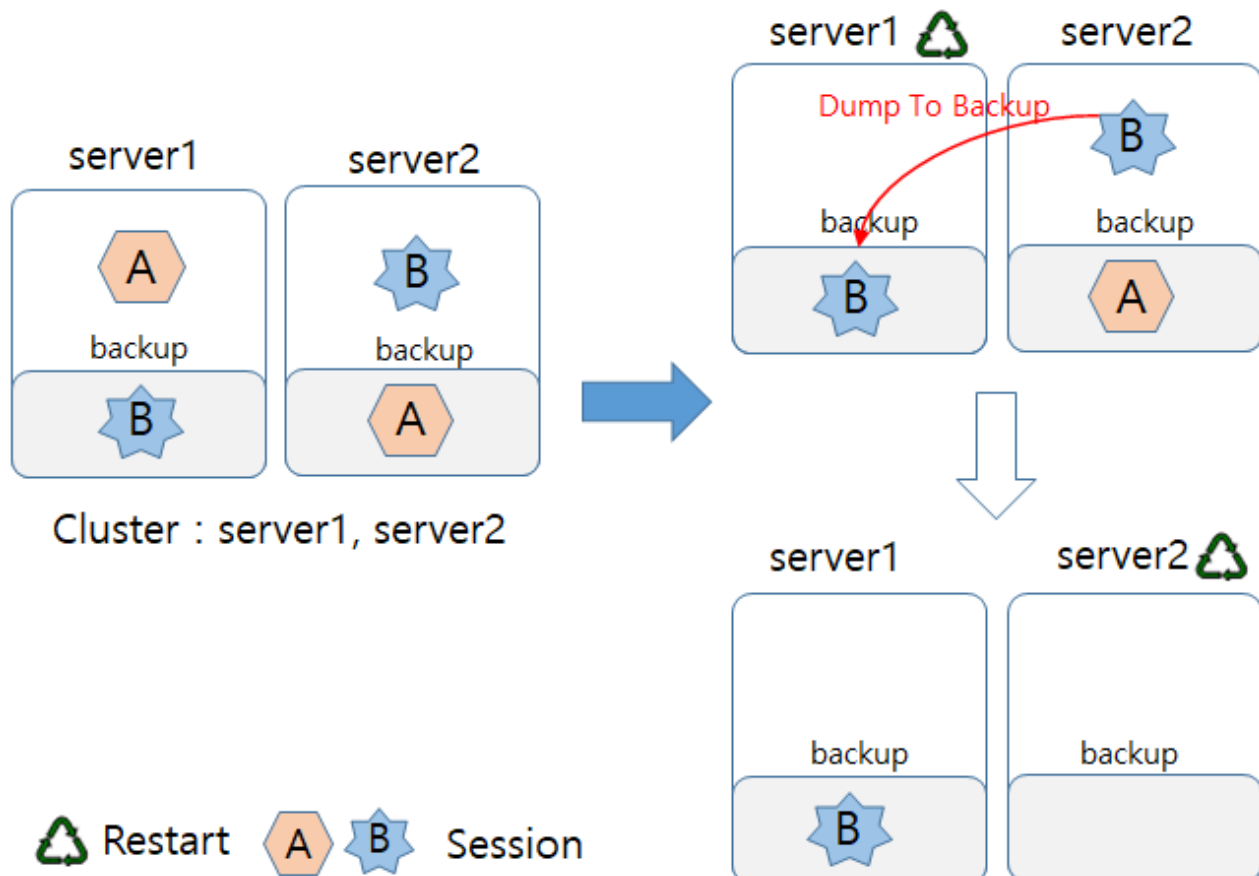
session is automatically logged out.

2.5.2. Failback

This section describes failback for distributed session servers. Failback is a basic function for session servers. It allows for a service to continuously operate even during a failure situation. That is, although a failure occurs in a server, a session is not lost and continuously provides a service because the session's backup is dumped to another session.

Failback provided by JEUS is for not only one failure but also consecutive failures, such as multiple servers' consecutive restart, and recovery of a server with a failure. The following explains this with a scenario that two servers restart consecutively.

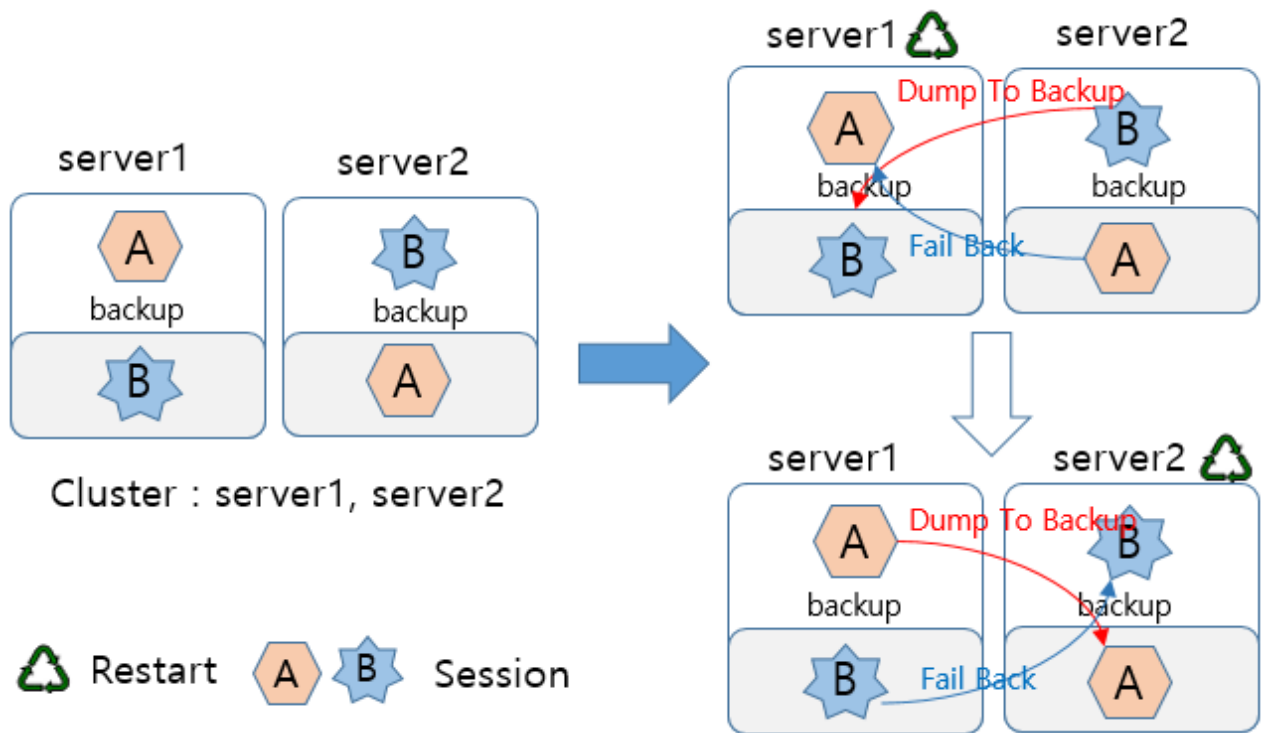
The following figure shows the scenario when failback is not supported.



When Failback Is Not Supported

If Server1 restarts, session A that belongs to the Server1 exists only in Server2 as backup. If Server2 restarts later, its backup will be removed, which means that the session A is removed from the entire system. If there is a request for the session A, a new session is created. This is because session backup is not processed in previous distributed session servers.

The following figure shows the scenario when failback is supported.



When Failback Is Supported

When Server1 restarts, DumpToBackup for session B is the same as in the case that failback is not supported. Additionally, session A in Server1 is recovered by performing failback with backup of session A in Server2.

Even after Server2 restarts later, all sessions are maintained.



Even though failback is not supported, a session is maintained if there is an access to the session when a server failure occurs (failover). This is possible because a backup session exists. The accessed session is backed up again to another server (DumpToBackup), it can operate normally. For this, some web servers try to maintain all sessions by accessing them.

Failback is executed using a reply-response approach determined by the configuration.

When server A is used as a backup server for server B, failback starts if the server A detects that the server B restarts.

1. Server A checks whether server B restarts and to have backup of server B's sessions.
2. If failback is set to true, the server A transfers backup of the server B's sessions to the server B.



If OOM occurs, failback is not recommended. If too many sessions are backed up, failback may cause OOM continuously. Failback is supported for rolling patches because servers need to be restarted consecutively without a session disconnection.

2.6. Session Cluster Modes

This section describes JEUS session cluster modes.

Session clustering is used to share sessions. Sessions must be shared within the same scope and must not be shared in different scopes. A session scope is determined according to a cluster mode.

JEUS supports the following three session cluster modes.

- [Default Session Cluster Mode](#)
- [Domain-Wide Session Cluster Mode](#)
- [Session Storage Scope Cluster Mode](#)

2.6.1. Default Session Cluster Mode

JEUS provides the default session cluster mode.

When servers are clustered, this mode is supported automatically and session data can be shared and used by the servers. Since this mode complies with servlet's basic session scope specifications, each application can manage a session independently and basic session clustering functions, such as failback and load balancing, are supported. The default scope is an application. Clustered servers can be a scope by using the Scope setting for Session Storage provided by a web engine session manager.

2.6.2. Domain-Wide Session Cluster Mode

In the domain-wide session cluster mode, all applications deployed in servers within a domain are regarded as a scope.

The default session cluster mode is restricted (targets to be deployed in a server) by a cluster structure because the mode depends on the cluster. In JEUS 6, different applications are deployed in each container and all containers' sessions can be shared.

The following sample scenario describes more details.

- There are two nodes and each node has two containers.
- A different application is deployed in each container. That is, a total of four applications exist.
- As a result, all sessions in the four containers can be shared.

The above scenario does not match the server cluster concept. Since all applications must be shared in a cluster, an application is forcibly deployed in all servers in the cluster. In a reliable environment, forcibly deployed applications may waste resources. However, they prevent the occurrence of a service not being available because a deployed application only physically exists in a specific server even though the server is included in a cluster. Therefore, a server must not be clustered to deploy a different application in each server.

Use the DOMAIN_WIDE mode when using session clustering without participating the server cluster.

The mode does not comply with domain structure specifications and is supported for user convenience. Note the following restrictions.

- The mode is not related to EJB and only affects web applications.
- If a server cluster as well as the mode are configured, the server cluster configurations will be ignored.
- If the mode is set in domain.xml of a domain, it is applied to all servers in the domain. Some servers in the domain cannot be specified as a target of the mode.



The domain-wide session cluster mode overrides server clusters.

2.6.3. Session Storage Scope Cluster Mode

Applications or clusters in the storage are considered being in the same scope.

The session storage scope cluster mode groups certain applications or clusters for sharing sessions, independently of the basic cluster setup. This mode is used when multiple groups exist. It differs from the basic session cluster or domain-wide session cluster mode in that the latter are generally used when only one application group exists. The session cluster has a different scope from distributed session servers, but it operates in the same way as the central/distributed session servers. Through the session storage scope mode, sessions can be shared between server clusters, and multiple scopes can be configured for a single server. This can be used in place of the existing specific-scope session cluster mode.

2.7. Session Server Configuration

This section describes the basic configuration of a session cluster.

A session cluster is basically used to share sessions. In addition to the cluster, to share sessions, a session server operates in a web engine and supports related functions. The operations and functions can be configured.

Settings for the session server can be configured in domain.xml and console.

```
<session-server>
  <cluster-mode>DEFAULT</cluster-mode>
  <session-storage>
    <name>jeus-session-storage</name>
    <session-manager-provider>JEUS</session-manager-provider>
    <scope>
      <name>jeus-distributed-scope</name>
      <jeus-session>DISTRIBUTED</jeus-session>
    </scope>
    <session-config>
      <timeout>30</timeout>
    </session-config>
  </session-storage>
</session-server>
```

```

    <max-session-count>-1</max-session-count>
    <reload-persistent>>false</reload-persistent>
    <tracking-mode>
      <cookie>>true</cookie>
      <url>>false</url>
      <ssl>>false</ssl>
    </tracking-mode>
    <session-cookie>
      <cookie-name>JSESSIONID</cookie-name>
      <url-cookie-name>jsessionid</url-cookie-name>
      <version>0</version>
      <max-age>-1</max-age>
      <path>/</path>
      <secure>>false</secure>
      <http-only>>true</http-only>
      <same-site>Disable</same-site>
      <partitioned>>false</partitioned>
    </session-cookie>
  </session-config>
  <target-cluster>cluster1</target-cluster>
</scope>
<scope>
  <name>jeus-central-scope</name>
  <jeus-session>CENTRAL</jeus-session>
  <session-config>
    <timeout>30</timeout>
    <max-session-count>-1</max-session-count>
    <reload-persistent>>false</reload-persistent>
    <tracking-mode>
      <cookie>>true</cookie>
      <url>>false</url>
      <ssl>>false</ssl>
    </tracking-mode>
    <session-cookie>
      <cookie-name>JSESSIONID</cookie-name>
      <url-cookie-name>jsessionid</url-cookie-name>
      <version>0</version>
      <max-age>-1</max-age>
      <path>/</path>
      <secure>>false</secure>
      <http-only>>true</http-only>
      <same-site>Disable</same-site>
      <partitioned>>false</partitioned>
    </session-cookie>
  </session-config>
  <target-application>sessionTest.war</target-application>
</scope>
</session-storage>
<property>
  <key>encoding-rule</key>
  <value>BASE64</value>
</property>
<jeus-login-manager>
  <login-manager-type>REDIS</login-manager-type>
  <primary>server1</primary>
  <secondary>server2</secondary>
  <property>
    <key>redis-nodes</key>
    <value>redis://localhost:6379</value>
  </property>

```

```

    </property>
  </jeus-login-manager>
  <jeus-central-session-server>
    <primary>server1</primary>
    <secondary>server2</secondary>
  </jeus-central-session-server>
</session-server>

```

• Basic Configuration

The following describes common settings applied in the entire domain. You can edit each item in **domain.xml**.

Value	Description
cluster-mode	<ul style="list-style-type: none"> • DEFAULT: Supports a session cluster only when there is a server cluster. A session cluster in application units is supported by default. (Default value) • DOMAIN_WIDE: Supports a session cluster where sessions for all applications in all servers in a domain are shared. <p>Each session cluster mode has a different scope, but uses the same distributed session server configuration options. The method for configuring session storage is described below.</p>
property	<p>A property can contain two types of key.</p> <ul style="list-style-type: none"> • excluded-servers: Enumerates names of servers that are not to be included in a cluster, with comma-delimiters. • encoding-rule: Provides encoding rules for sticky routing. The following encoding modes are currently supported. <ul style="list-style-type: none"> ◦ BASE64 : Encodes information by using the BASE64 rule to avoid exposing engine or domain information through the session ID. (Default value) ◦ BASE64_WITHOUT_PADDING: Encodes information by using the BASE64 rule without PADDING ("=") to avoid exposing engine or domain information through the session ID. ◦ RAW: Does not encode information. This is used for debugging because it is difficult to determine which engine sent the request by using encoded information, or under a circumstance where no security issues exist even though an engine name is exposed. <p>The property can also be set through the console. For more information, refer to "set-sessionserver-property" in <i>JEUS Reference Guide</i>.</p>

• Session Storage

You can **add**, **modify**, and **delete** using **domain.xml** or **console commands**.

Refer to "Session Commands" in *JEUS Reference Guide*.

Value	Description
Name	Storage name.
Session Manager Provider	<p>Sets the implementation or a reserve word to be used for the session manager. Other than reserve words, the entire package name must be set.</p> <p>Four reserved words are supported for configuration. For descriptions of these reserved words, refer to the [Reserved Words] below the table.</p>
Property	Storage property. For information about valid keys for Property, refer to the description of [Valid Keys for Property] below the table.

[Reserved Words]

The following describes the reserved words that can be configured for the 'Session Manager Provider' in Session Storage settings.

- **JEUS**

Uses the JEUS implementation even if other implementations are provided. (Default value)

- **REDIS**

Uses Redis as the session storage. Must be version 5 or later. Configured in Property.

The following are valid keys.

- Enter the Redis architecture, either as 'standalone' or 'cluster'. (Default value: standalone)

Example

```
<key>redis-architecture</key>
<value>standalone</value>
```

- Enter the Redis node. If the architecture is cluster, list the cluster members with comma delimiters. (Default value: redis://localhost:6379)

Example

```
<key>redis-nodes</key>
<value>redis://127.0.0.1:7006,redis://127.0.0.1:7007,redis://127.0.0.1:7008</value>
```

- **HAZELCAST**

Uses Hazelcast as the session storage. Must be version 4.2. Configured in Property.

The following are valid keys.

- Enter the Hazelcast architecture, either as 'standalone' or 'cluster'. (Default value: standalone)

Example

```
<key>hazelcast-architecture</key>
<value>standalone</value>
```

- Enter the Hazelcast node. (Default value: localhost:5701)

Example

```
<key>hazelcast-nodes</key>
<value>localhost:5702</value>
```

- Enter the Hazelcast cluster. (Default value: dev)

Example

```
<key>hazelcast-cluster</key>
<value>dev</value>
```

• RUNTIME

Uses another implementation instead of JEUS-supported providers', if any. If no other implementation exists, one of those three JEUS-provided implementations will be set.

[Valid Keys for Property]

The following is an additional description for the '**Property**' section of the Session Storage configuration.

- When the **Provider is JEUS**

Value	Description
reserved-thread-num	Number of threads in the system thread pool. Only recommended for use if a service requires preallocated threads.
connection-timeout	Timeout value for opening a connection between session servers in the web engine.
read-timeout	Read timeout value for communication between session servers in the web engine. After sending data, this is the maximum amount of time to wait for a response.

Value	Description
backup-level	<p>Session standard used when backing up a session to a remote web container or a local database file.</p> <ul style="list-style-type: none"> ◦ access: Updates each session attribute and backs up specific attributes that calls setAttribute, putValue, removeAttribute, removeValue, getAttribute, and getValue. (Default value) ◦ set: When the session's setAttribute, putValue, removeAttribute, or removeValue method is called, the session is considered modified and the session object is backed up. ◦ get: When the session's setAttribute, putValue, removeAttribute, removeValue, getAttribute, or getValue method is called, it is considered modified and the session object is backed up. ◦ all: Backs up all sessions. When the session's HttpServletRequest.getSession() is called, it is considered modified and the session object is backed up.
backup-unit-size	Maximum number of transferred sessions to be backed up at the same time.
backup-queue-size	<p>Size of a queue that saves sessions that are not transferred normally to be backed up due to network delay.</p> <p>If there is no backup in a distributed session server, a session may be lost in a failure situation. Therefore, if the queue is full, a request is delayed so that a flow control can be performed without an issue.</p>
backup-flowcontrol-enabled	Option to allow a service to continue even if a backup queue is full. This is used to ignore session loss in order to continue a service.
failover-delay	<p>Amount of time to wait for recovery when an engine fails.</p> <p>This is the timeout for the other engines besides the failed one to reconnect to the cluster.</p>
restart-delay	When an engine shuts down normally, this is the timeout for the other engines to reconnect to the cluster. This setting enhances the performance for a restart which is the most frequent case of shutting down a web engine.
user-info	Path to the location from which the user information can be retrieved.
login-manager	<p>Option to use the login manager.</p> <ul style="list-style-type: none"> ◦ false (Default value) ◦ true

Value	Description
login-manager-strategy	Login manager operating strategy. <ul style="list-style-type: none"> ◦ invalidate-before: When duplicate login is detected, the previous session is invalidated. (Default value) ◦ invalidate-after: When duplicate login is detected, the later session is invalidated.

- When the **Provider is REDIS**

Value	Description
redis-nodes	Redis node. If the architecture is a cluster, list the members with comma delimiters. (Default value: redis://localhost:6379)
redis-architecture	Redis architecture. One of the following: <ul style="list-style-type: none"> ◦ standalone (Default value) ◦ cluster

- When the **Provider is HAZELCAST**

Value	Description
hazelcast-nodes	Hazelcast node. (Default value: localhost:5701)
hazelcast-cluster	Hazelcast cluster. (Default value: dev)
hazelcast-architecture	Hazelcast architecture. One of the following: <ul style="list-style-type: none"> ◦ standalone (Default value) ◦ cluster

- **Scope**

Value	Description
name	Scope name.
jeus-session	Setting for whether to use a distributed or centralized method for JEUS sessions. If the storage provider is Redis or Hazelcast, this setting is ignored. <ul style="list-style-type: none"> • DISTRIBUTED: Distributed session. • CENTRAL: Central session.
target	Target application or cluster to be included in the scope (only one can be configured). <ul style="list-style-type: none"> • target application: Specifies the application to be included in the scope. • target-cluster: Specifies the cluster to be included in the scope.

Value	Description
Session Config	Session configuration to be used by the context included in the scope. For information about detailed configuration, refer to Session Configuration .

◦ **jeus-central-session-server**

Configure the server to be used as JEUS's own session server. Configuration items can be modified using **domain.xml** or **console commands**. For more information, refer to "set-jeus-central-session-server" in *JEUS Reference Guide*.

Value	Description
primary	Sets the primary session server. Cannot be dynamically changed.
secondary	Sets the secondary session server. Cannot be dynamically changed.

◦ **jeus-login-manager**

Configuration for using the Jeus Login Manager. Configuration items can be modified using **domain.xml**.

Value	Description
login-manager-type	<p>Sets the implementation or a reserve word to be used as the Login Manager.</p> <p>The following three reserve words are supported.</p> <ul style="list-style-type: none"> ◦ JEUS: Uses the JEUS implementation even if other implementations are provided. ◦ REDIS: Uses Redis as the session storage. Must be version 5 or later. Configured in Property. ◦ HAZELCAST: Uses Hazelcast as the session storage. Must be version 4.2. Configured in Property.
primary	Sets the primary server for JEUS Login Manager. Cannot be dynamically changed. This setting is effective only when the ' Login Manager ' item is specified as 'JEUS'.
secondary	Sets the secondary server for JEUS Login Manager. Cannot be dynamically changed. This setting is effective only when the ' Login Manager ' item is specified as 'JEUS'.

Value	Description
property	<p>Sets properties for JEUS Login Manager.</p> <p>The following are valid keys for Property.</p> <ul style="list-style-type: none"> ◦ When 'Login Manager' is specified as 'REDIS' <ul style="list-style-type: none"> ◦ redis-nodes: Node for Redis. If the architecture is a cluster, enumerate cluster members with comma delimiters. (Default value: redis://localhost:6379) <p>Example</p> <pre><key>redis-nodes</key> <value>redis://127.0.0.1:7006, redis://127.0.0.1:7007,redis://127.0.0.1:7008</value></pre> ◦ When 'Login Manager' is specified as 'HAZELCAST' <ul style="list-style-type: none"> ◦ hazelcast-nodes: Node for Hazelcast. (Default value: localhost:5701) <p>Example</p> <pre><key>hazelcast-nodes</key> <value>localhost:5702</value></pre> ◦ hazelcast-cluster: Enter the cluster for Hazelcast. (Default value: dev) <p>Example</p> <pre><key>hazelcast-cluster</key> <value>dev</value></pre>