BEA WebLogic Commerce Server
BEA WebLogic Personalization Server

Performance Tuning Guide

WebLogic Commerce Server 3.1
WebLogic Personalization Server 3.1
Document Edition 1.0
November 2000

Copyright

Restricted Rights Legend

Trademarks or Service Marks

Performance Tuning Guide

Document Edition	Date	Software Version
1.0	November 2000	WebLogic Commerce Server 3.1.1

Contents

Adjust the Intervals for Checking JSP and Servlet Modifications 2
About the Page-Check Intervals Properties 3
About the Reload-Servlet Interval Property 4
To Adjust the Intervals 4
For More Information 4
Adjust Database Connections Available at Startup 5
Set the Reload Policy for Rules 6
Adjust Caching 7
Adjust and Use the Session and Global Caches 7
Enabling the Caches 8
JSP Tags for Accessing HttpSession and the Session and Global Caches 9
An API for Accessing HttpSession and the Session and Global Caches 9
Guidelines for Placing Data in HttpSession, Session Cache, or Global Cache 10
Adjust Caching for Content Management 10
Enable Property Caching 13
Property Caching in a Clustered Environment 13
To Enable Property Caching 14
Enable Group Caching 15
Group Caching in a Clustered Environment 16
To Set Up the Group-Cache Table 16
To Enable and Configure the Group Cache 17
To Access Data in the Group Cache Table 17
Adjust Portal and Portlet Settings While Load Testing 18
Display Metadata, Sort and Query Explicit Metadata 19

Use LDAP for Authentication Only 19
Use the DocumentManager EJB..... 20

Performance Tuning Guide

When you first install BEA WebLogic Commerce Server and Personalization Server, it is configured to support Web-site developers and administrators. For example some caching mechanisms are disabled so developers can see the results of their modifications immediately.

When you are ready to make your Web site available to customers, refer to *WebLogic Server Performance Tuning Guide* for information about tuning WebLogic Server.

Then, for information about tuning WebLogic Commerce Server and Personalization Server performance, refer to the following topics in this document:

- Adjust the Intervals for Checking JSP and Servlet Modifications
- Adjust Database Connections Available at Startup
- Set the Reload Policy for Rules
- Adjust Caching
  - Adjust and Use the Session and Global Caches
  - Adjust Caching for Content Management
  - Enable Property Caching
  - Enable Group Caching
- Adjust Portal and Portlet Settings While Load Testing
- Display Metadata, Sort and Query Explicit Metadata
- Use LDAP for Authentication Only
- Use the DocumentManager EJB

Adjust the Intervals for Checking JSP and Servlet Modifications

By default, each time a Web browser requests a JSP, Commerce Server checks for any modifications to the JSP source file. Likewise, each time Commerce Server sends a request to a servlet, it checks for any modifications to the servlet class files.

For your production Web site, you can decrease the amount of time in which Commerce Server serves JSPs and processes requests to servlets by increasing the intervals at which the server checks for modifications.

Although Commerce Server performs faster with higher values for the modification-check intervals, the higher values reduce sensitivity to changes in your source files. For example, you can set the server to check for JSP modifications every 10 minutes. After you change a JSP, it will take up to 10 minutes for the server to see the modifications.

This section includes the following topics:

- About the Page-Check Intervals Properties
- About the Reload-Servlet Interval Property
- To Adjust the Intervals
- For More Information

About the Page-Check Intervals Properties

Two properties determine the interval at which WebLogic Server checks to see if JSP files have changed and need recompiling:

- The `pageCheckSeconds` property in `WL_COMMERCE_HOME\weblogic.properties`, which applies only to servlets that WebLogic deploys in the default servlet context.

  The following excerpt from `weblogic.properties` shows the property in boldface text with its default value:

  ```
  weblogic.httpd.register.*.jsp=\
  weblogic.servlet.JSPServlet

  weblogic.httpd.initArgs.*.jsp=\
  pageCheckSeconds=0,\

  packagePrefix=jsp,\
  compileCommand=d:/bin/jikes.exe,\
  workingDir=d:/weblogic/myserver/classfiles,\
  verbose=false,\
  keepgenerated=true
  ```

- The `weblogic.jsp.pageCheckSeconds` context parameter in a web app's deployment descriptor (`web.xml` file), which applies only to the servlets that WebLogic Server deploys in the context of the web app.

  The following excerpt from `WL_COMMERCE_HOME\server\webapps\web-inf\web.xml` shows the `weblogic.jsp.pageCheckSeconds` context parameter in boldface text with the default value:

  ```
  <context-param>
       <param-name>weblogic.jsp.pageCheckSeconds</param-name>
       <param-value>0</param-value>
  </context-param>
  ```

Note: Neither page-check interval determines the frequency with which Commerce Server checks for updated content that is stored in the database and in a content management system. Instead, the `ttl` (time to live) settings for various caches determine the refresh rate for content. For example, if you set the page-check intervals to once a second, and you set the `ttl` for the content cache to 10 minutes, it can take up to 10 minutes for the server to see the new content, even though it is checking for new JSP source code every second. For information on setting `ttl` properties for caches, refer to "Adjust Caching" on page 7.

About the Reload-Servlet Interval Property

The `weblogic.servlet.reloadCheckSecs` context parameter in a web app's deployment descriptor (`web.xml` file) specifies the interval in seconds that the web app checks for modified servlet classes.

The following excerpt from `WL_COMMERCE_HOME\server\webapps\web-inf\web.xml` shows the `weblogic.servlet.reloadCheckSecs` context parameter in boldface text with the default value:

```
<context-param>
     <param-name>weblogic.servlet.reloadCheckSecs</param-name>
     <param-value>600</param-value>
</context-param>
```

To Adjust the Intervals

To determine the optimal page-check and reload-servlet intervals for your production Web site do the following:

1. Establish performance baselines by testing Commerce Server performance with all three intervals set to `-1` (which specifies that the server never checks for modifications).
2. Test the performance with the intervals set to various numbers of seconds. For example, set the intervals to `600` seconds (10 minutes) and test the performance. Then set the intervals to `900` seconds and test the performance.
3. Choose intervals that provide the best performance while checking for modifications to JSP files and servlet classes at a satisfactory rate.

For More Information

For more information about configuring JSP and servlet options for WebLogic Server, see the following topics on the WebLogic Server documentation Web site:

- "Writing a Web Application"
- "Using WebLogic JSP"

Adjust Database Connections Available at Startup

To optimize the database pool performance for your production web site, open `$WL_COMMERCE_HOME/weblogic.properties` and modify the values for the following `weblogic.jdbc.connectionPool.commercePool` properties:

- `loginDelaySecs`. Change to `0`
- `initialCapacity`. Change to `maxCapacity`
- `allowShrinking`. Change to `false`
- `testConnsOnReserve`. Change to `false`

For example:

```
weblogic.jdbc.connectionPool.commercePool=\
url=jdbc:oracle:thin:@server:port:instance,\
driver=oracle.jdbc.driver.OracleDriver,\
loginDelaySecs=0,\
initialCapacity=maxCapacity,\
maxCapacity=20,\
capacityIncrement=1,\
allowShrinking=false,\
shrinkPeriodMins=15,\
testConnsOnReserve=false,\
props=user=user;password=pwd,\
refreshMinutes=5
```

FOR MORE INFORMATION

For more information on database connection pools, see "Creating and Using Connection Pools" on the WebLogic Server documentation Web site and "Setting Up Connection Pools" under "Creating and Managing Content" in *Personalization Server User's Guide*.

Set the Reload Policy for Rules

You can determine the frequency with which Personalization Server checks for changes to rules by doing the following:

1. Open `$WL_COMMERCE_HOME/lib/rulesservice.jar`.
2. In `ejb-jar.xml` (which is in `rulesservice.jar`), modify the value for `rulesetReloadInterval`. The value expresses the number of milliseconds that Personalization Server waits before checking for changes to rules. For example:

   <env-entry>
   <env-entry-name>**rulesetReloadInterval**</env-entry-name>
   <env-entry-type>java.lang.String</env-entry-type>
   <env-entry-value>**300000**</env-entry-value>
   </env-entry>

Personalization Server performs faster with a higher value for the reload interval, however, the higher reload value reduces sensitivity to rule changes.

If you shut down and restart your production servers when you make changes to your site, you can boost performance by setting the reload policy in `ejb-jar.xml` (which is in `rulesservice.jar`) to `reloadNever`. For example:

```
<env-entry>
<env-entry-name>rulesetReloadPolicy</env-entry-name>
<env-entry-type>java.lang.String</env-entry-type>
<env-entry-value>reloadNever</env-entry-value>
</env-entry>
```

If you set the reload policy to reload`Never`, Personalization Server does not recognize changes to rules until you restart the server.

For More Information

For more information about rules, see "Creating and Managing Rules" in *Personalization Server User's Guide*.

Adjust Caching

To adjust caching for production Web site, complete the following tasks:

- Adjust and Use the Session and Global Caches
- Adjust Caching for Content Management
- Enable Property Caching

Adjust and Use the Session and Global Caches

In a clustered environment, you can improve scalability and performance by minimizing the use of `HttpSession` objects. (`HttpSession` is part of the JDK session-tracking mechanism, which servlets use to maintain state about a series of requests from the same user.)

To minimize using `HttpSession`, each server in the WebLogic Commerce Server and Personalization Server cluster provides the following caches:

- `session` cache, which stores data in memory about each session. The function of the session cache is the same as `HttpSession`, however, unlike `HttpSession`, it is not replicated across the cluster.
- `global` cache, which stores data in memory that multiple sessions can use. For example, sessions for anonymous users can access data from the global cache. Like the session cache, it is not replicated across the cluster.

This section discusses the following topics:

- Enabling the Caches
- JSP Tags for Accessing HttpSession and the Session and Global Caches
- An API for Accessing HttpSession and the Session and Global Caches
- Guidelines for Placing Data in HttpSession, Session Cache, or Global Cache

FOR MORE INFORMATION

For more information about how WebLogic Commerce Server and Personalization server process HTTP requests, refer to "Foundation Classes and Utilities" in *Personalization Server Developer's Guide*. For more information about `HttpSession`, see http://java.sun.com/products/servlet/2.2/javadoc/javax/servlet/http/HttpSession.html. For more information about WebLogic Server clusters, see *Using WebLogic Server Clusters*.

Enabling the Caches

To enable the session and global caches, add the following properties to `$WL_COMMERCE_HOME/weblogiccommerce.properties`:

```
_sessionCache.ttl=900000
_sessionCache.capacity=10000
_sessionCache.enabled=true


_globalCache.ttl=600000
_globalCache.capacity=1000
_globalCache.enabled=true
```

The `ttl` (time to live) property determines the number of milliseconds that the server maintains the cache. The `capacity` property determines the maximum number of objects in the cache. (Both `session` and `global` are in-memory caches.) The `enabled` property determines whether the cache is activated. A `false` value deactivates the cache and obviates the `ttl` and `capacity` properties; `true` activates it.

You can increase or decrease values for `ttl` and `capacity` based on the amount of available memory and the level of performance you desire.

Note: Each server in a cluster maintains its own set of caches, each of which must be configured separately by modifying the server's `weblogiccommerce.properties` file. Because the session and global caches are not replicated across servers in the cluster, if a server fails, the data in its caches is inaccessible. For guidelines about which types of data to place in the session and global caches, see "Guidelines for Placing Data in HttpSession, Session Cache, or Global Cache" on page 10.

JSP Tags for Accessing HttpSession and the Session and Global Caches

Use the following JSP tags from the FlowManager tag library to place, retrieve, and remove data from `HttpSession` as well as the session and global caches:

- `<fm:getCachedAttribute>`
- `<fm:setCachedAttribute>`
- `<fm:removeCachedAttribute>`
- `<fm:getSessionAttribute>`
- `<fm:setSessionAttribute>`
- `<fm:removeSessionAttribute>`

For information about these tags, refer to "JSP Tag Library Reference" in *Personalization Server Developer's Guide*.

An API for Accessing HttpSession and the Session and Global Caches

Use the following methods of the `com.beasys.commerce.foundation.flow.helper.FlowManagerHelper` API to place, retrieve, and remove data from `HttpSession` and the session and global caches:

- getCachedValue
- setCachedValue
- removeCachedValue
- getGlobalCachedValue
- setGlobalCachedValue
- removeGlobalCachedValue
- getSessionAttribute
- setSessionAttribute
- removeSessionAttribute

For information about these methods, refer to the documentation for `com.beasys.commerce.foundation.flow.helper.FlowManagerHelper` in *WebLogic Personalization Server Javadoc*.

Guidelines for Placing Data in HttpSession, Session Cache, or Global Cache

In general, place only the following in `HttpSession`:

- Items that are required for replication across the cluster.
- Any keys that are required to look up information. When you enable session replication for WebLogic Server, `HttpSession` is replicated on all machines in a cluster. Placing information in `HttpSession` while session replication is enabled provides a backup for data lookups. For example, you place query parameters for a search in `HttpSession` and the search results in the `session` cache. While returning the search results the server fails. Another server can recreate the search by referring to the parameters that are stored in the `HttpSession` replica.

Place any information that multiple users require (either within the same application or across multiple applications) in the global cache.

Place all other session-related information in the session cache.

Adjust Caching for Content Management

To optimize content-management performance for your production Web site, configure Personalization Server as follows:

- For the `cm:select`, `cm:selectById`, `pz:contentQuery`, and `pz:contentSelector` JSP tags, use the `useCache` attribute whenever possible. Doing so avoids a call to DocumentManager and, in the case of `pz:ContentSelector`, to the RuleService.

  For information on using the `useCache` attribute, refer to "JSP Tag Library Reference" in *Personalization Server Developer's Guide*.

To clear cached content when user and/or document attributes change, use the `remove` method of `com.beasys.commerce.content.ContentCache`. For more information, see the JavaDoc for `com.beasys.commerce.content.ContentCache`.

For an example of a JSP file that uses the `remove` method, see

`WL_COMMERCE_HOME/server/public_html/examples/content/cache-control.jsp`

- For the `cm:select`, `cm:selectById`, `pz:contentQuery`, and `pz:contentSelector` JSP tags, set the `cacheScope` attribute to `application` whenever possible. For example:

```
<cm:select id="myDocs" query="riskFactor = 'Low'"
useCache="true" cacheId="myDocs"
cacheScope="application"
max="10" cacheTimeout="300000" />
```

  The application cache type is global instead of per-user and should speed up queries by avoiding a call to the DocumentManager EJB.

Note: For `pz:contentSelector`, set the `cacheScope` attribute to **`application`** only when you want to select **shared** content. For example, in exampleportal, the Acme Promotion portlet uses an application-scoped cache to select content for non-authenticated users. Because it uses the application scope, all non-authenticated users see the same content. For authenticated users, Acme Promotion provides personalized content by switching to a session scoped cache.

- Whenever you can predict the next document that users will view based on the document that they are currently viewing, load the next document into the cache before users request it. This “forward caching” will greatly improve the speed at which Personalization Server responds to user requests (assuming that your prediction is correct; forward caching a document that no one requests will only degrade performance and scalability).

The following JSP fragment is an example of forward caching a document:

```
<%-- Get the first set of content --%>

<cm:select id="myDocs" query="riskFactor = 'Low'"
useCache="true" cacheId="myDocs"
cacheScope="application"
max="10" cacheTimeout="300000" />

<%-- Generate a query from each content's relatedDocId --%>

<% String query = null; %>
<es:forEachInArray array="<%=myDocs%>" id="myDoc"
type="com.beasys.commerce.axiom.content.Content">

<% String relId = (String)myDoc.getProperty("relatedDocId",
null); %>
<es:notNull item="<%=relId%>">

<%
if (query != null)
query += " || ";
else
query = "";
query += "identifier = '" +
ExpressionHelper.toStringLiteral(relId) + "'";

%>

</es:notNull>
</es:forEachInArray>

<%-- Load the related content into the cache via cm:select
--%>

<es:notNull item="<%=query%>">

<cm:select query="<%=query%>" id="foo" useCache="true"
cacheId="relatedDocs"
cacheScope="session" max="10" cacheTimeout="300000" />

</es:notNull>
```

For More Information

For more information about content management, see "Creating and Managing Content" in *Personalization Server User's Guide*.

For more information about JSP tags for content management, see "JSP Tag Library Reference" in *Personalization Server Developer's Guide*.

Enable Property Caching

The WebLogic Server Configurable Entity and Entity Property Manager provide several in-memory caches that you can enable for WebLogic Commerce Server and Personalization Server. The caches decrease the amount of time needed to access user, group, and other properties, but introduce the possibility of stale data.

This section discusses the following topics:

- Property Caching in a Clustered Environment
- To Enable Property Caching

Property Caching in a Clustered Environment

With property caching enabled in a clustered environment, each server in a cluster maintains its own cache; the cache is not replicated on other servers. In this environment, when properties that are stored in the `defaultPropertyCache`, `entityPropertyCache`, `directPropertyManager`, or `ldapPropertyCache` change on one server, they may not change on another server in a timely fashion.

In most cases, immediate or quick access to properties on another server is not necessary: user sessions are pinned to a single server, and even with caching enabled, users immediately see changes they make to their own settings on the server.

However, if a server fails and loses the data in its caches, modifications to properties may be lost, depending on the longevity of the property cache. In addition, if an administrator changes a user's properties, the user may not see the changes during her session if she and the administrator are pinned to different servers in the cluster.

You can mitigate these situations by specifying a small `ttl` (time-to-live) setting when you enable the caches. The small `ttl` setting provides performance gains by caching data, but the short-lived caches increase the rate at which property changes are replicated across servers.

If you require multiple servers in a cluster to have immediate access to modified properties, disable property caching by adding the entries described in “To Enable Property Caching” and specifying `false` for the `unifiedProfileTypeCache.enabled` value.

To Enable Property Caching

To enable property caching, add the following entries to `WL_COMMERCE_HOME\weblogiccommerce.properties`, adjusting the values based on the number of properties in your property sets and the frequency with which you want the data updated:

Note: These entries enable in-memory caching. Caches that grow exceedingly large may degrade performance.

- To create a cache of unified profile types that lives for 1 hour and contains 100 entries, add:

```
unifiedProfileTypeCache.ttl=3600000
unifiedProfileTypeCache.capacity=100
unifiedProfileTypeCache.enabled=true
```

- To create a cache of default schema properties that lives for 10 minutes and contains 500 entries, add:

```
defaultPropertyCache.ttl=600000
defaultPropertyCache.capacity=500
defaultPropertyCache.enabled=true
```

- To create a cache of entity properties that lives for 10 minutes and contains 500 entries, add:

```
entityPropertyCache.ttl=600000
entityPropertyCache.capacity=500
entityPropertyCache.enabled=true
```

- To create a cache of LDAP entity properties that lives for 10 minutes and contains 500 entries, add:

```
ldapEntityPropertyCache.ttl=600000
ldapEntityPropertyCache.capacity=500
ldapEntityPropertyCache.enabled=true
```

- To create a cache of entity ids that lives for 1 hour and contains 500 entries, add:

```
entityIdCache.ttl=3600000
entityIdCache.capacity=500
entityIdCache.enabled=true
```

- To create a cache of explicit properties that lives for 10 minutes and contains 100 entries, add:

```
directPropertyManager.ttl=600000
directPropertyManager.capacity=100
directPropertyManager.enabled=true
```

- To create a cache of ConfigurableEntity methods that lives for 1 hour and contains 100 entries, add:

```
ConfigurableEntityMethodCache.ttl=3600000
ConfigurableEntityMethodCache.capacity=100
ConfigurableEntityMethodCache.enabled=true
```

FOR MORE INFORMATION

For more information about property sets, see "Creating and Managing Property Sets" in *Personalization Server User's Guide*.

For more information about JSP tags for managing property sets, see "JSP Tag Library Reference" in *Personalization Server Developer's Guide*.

Enable Group Caching

In systems with a deep group hierarchies, you can improve performance using group caching, which precalculates group membership information and stores the calculation results in a new database table, WLCS_USER_GROUP_CACHE. Any queries that are submitted while group caching is recalculating data return the old, previously committed data.

With group caching, you exchange faster performance for the risk of stale or inconsistent data. To balance performance with data consistency, you can configure the interval at which the caching mechanism recalculates and updates the table.

This section contains the following topics:

- Group Caching in a Clustered Environment
- To Set Up the Group-Cache Table
- To Enable and Configure the Group Cache
- To Access Data in the Group Cache Table

Group Caching in a Clustered Environment

To improve performance of group caching in a cluster, you can establish one cache as the master. The server with the master cache periodically updates its WLCS_USER_GROUP_CACHE table. All other servers in the cluster read this master table; they do not update the table or maintain their own copy. For information on setting up a master cache, refer to “To Enable and Configure the Group Cache” on page 17.

To Set Up the Group-Cache Table

To set up the table for group caching, issue the following SQL commands:

```
CREATE TABLE WLCS_USER_GROUP_CACHE ( USER_NAME VARCHAR2(100) NOT NULL,
GROUP_NAME VARCHAR2(100) NOT NULL );
ALTER TABLE WLCS_USER_GROUP_CACHE
ADD CONSTRAINT WLCS_USER_GROUP_CACHE_INDEX PRIMARY KEY ( USER_NAME,
GROUP_NAME );
```

To Enable and Configure the Group Cache

To enable the group cache, add all of the following lines to `$WL_COMMERCE_HOME/weblogic.properties`:

- `weblogic.system.startupClass.GroupCache=com.beasys.commerce.axiom.contact.security.GroupCache`
- `weblogic.system.startupArgs.GroupCache=updateDb=true`

In a clustered environment, to create a master cache, specify `weblogic.system.startupArgs.GroupCache=updateDb=`**`true`** for one server and `weblogic.system.startupArgs.GroupCache=updateDb=`**`false`** for all other servers in the cluster.

To configure the number of seconds that the server waits before calculating and updating the table, change the value for following WebLogic Server property: `weblogic.security.realm.cache.group.ttl.positive`

Note: You do not need to specify the size of the group cache.The depth of the group hiearchies determines the size of the group cache table.

To Access Data in the Group Cache Table

To access data in the group cache table, use any of the following:

- The new `UserManager` method of `getCachedGroupNamesForUser`
- The `static` method of the `GroupCache` object

For more information about these methods, refer to *WebLogic Personalization Server Javadoc*.

Adjust Portal and Portlet Settings While Load Testing

If you are testing the performance of the portal framework, do the following:

- Enable session and global caches as described in “Adjust and Use the Session and Global Caches” on page 7. (You do not need to add JSP tags or API methods that access the caches when testing the portal framework; the framework includes them by default.)
- Because slow portlets can severely slow a portal’s performance, remove all of the portlets from the portal except for Dictionary, Search, and Quote. These portlets do not invoke external activities such as database connections.
- Modify the framework’s Application Initialization Property Set as follows:
  - For `refreshWorkingDir`, increase the default number of seconds to prevent Personalization Sever from refreshing the working directory every five minutes (300 seconds) during a long load test.

    The working directory is the root of the portal pages and WebLogic Personalization Server pages hierarchy. You define the working directory in a JSP, and you can change it as needed without restarting the server. The `refreshWorkingDir` property determines how frequently the server checks to see if you have changed the working directory.

    The Application Initialization Property Set for the exampleportal defines the `refreshWorkingDir` property. If you base your portal on the exampleportal, it too will define the `refreshWorkingDir` property.
  - For `ttl`, increase the default number of milliseconds to prevent Personalization Sever from reloading properties every five minutes (300000 milliseconds) during a long load test.

For More Information

For more information about managing portals, see “Creating and Managing Portals” in *Personalization Server User’s Guide*.

For more information about developing portlets, see "Developing Portlets" in *Personalization Server Developer's Guide*.

Display Metadata, Sort and Query Explicit Metadata

If you used the BulkLoader to load document metadata into the reference implementation document database, you can improve document management performance when retrieving documents by doing the following:

- Display a document's metadata instead of the full document.
- Sort on explicit (system-defined) metadata attributes instead of implicit (user-defined) metadata attributes.
- Query on explicit metadata attributes instead of implicit metadata attributes.

FOR MORE INFORMATION

For more information about content management, see "Creating and Managing Content" in *Personalization Server User's Guide*.

Use LDAP for Authentication Only

For improved performance, use LDAP for authentication only; do not use it to retrieve user and group properties. Instead of retrieving properties from LDAP servers, configure your system to use properties stored in the RDBMS by minimizing the number of properties registered for retrieval from LDAP in the user management tools.

FOR MORE INFORMATION

For more information about changing LDAP settings, see "Using Other Realms" under "Creating and Managing Users" in *Personalization Server User's Guide*.

Use the DocumentManager EJB

Always use a DocumentManager EJB instead of Document EJB. Document EJBs are deprecated.