



BEA WebLogic Commerce Server BEA WebLogic Personalization Server

Performance Tuning Guide

WebLogic Commerce Server 3.2
WebLogic Personalization Server 3.2
Document Edition 3.2.1
January 2001

Copyright

Copyright © 2001 BEA Systems, Inc. All Rights Reserved.

Restricted Rights Legend

This software and documentation is subject to and made available only pursuant to the terms of the BEA Systems License Agreement and may be used or copied only in accordance with the terms of that agreement. It is against the law to copy the software except as specifically allowed in the agreement. This document may not, in whole or in part, be copied, photocopied, reproduced, translated, or reduced to any electronic medium or machine readable form without prior consent, in writing, from BEA Systems, Inc.

Use, duplication or disclosure by the U.S. Government is subject to restrictions set forth in the BEA Systems License Agreement and in subparagraph (c)(1) of the Commercial Computer Software-Restricted Rights Clause at FAR 52.227-19; subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013, subparagraph (d) of the Commercial Computer Software--Licensing clause at NASA FAR supplement 16-52.227-86; or their equivalent.

Information in this document is subject to change without notice and does not represent a commitment on the part of BEA Systems. THE SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. FURTHER, BEA Systems DOES NOT WARRANT, GUARANTEE, OR MAKE ANY REPRESENTATIONS REGARDING THE USE, OR THE RESULTS OF THE USE, OF THE SOFTWARE OR WRITTEN MATERIAL IN TERMS OF CORRECTNESS, ACCURACY, RELIABILITY, OR OTHERWISE.

Trademarks or Service Marks

BEA, WebLogic, Tuxedo, and Jolt are registered trademarks of BEA Systems, Inc. How Business Becomes E-Business, BEA WebLogic E-Business Platform, BEA Builder, BEA Manager, BEA eLink, BEA WebLogic Commerce Server, BEA WebLogic Personalization Server, BEA WebLogic Process Integrator, BEA WebLogic Collaborate, BEA WebLogic Enterprise, and BEA WebLogic Server are trademarks of BEA Systems, Inc.

All other product names may be trademarks of the respective companies with which they are associated.

Performance Tuning Guide

Document Edition	Date	Software Version
3.2	December 2000	WebLogic Commerce Server 3.2
3.2.1	January 2001	WebLogic Commerce Server 3.2

Contents

About This Document

What You Need to Know	v
e-docs Web Site	v
How to Print the Document	vi
Related Information	vi
Contact Us!	vi
Documentation Conventions	vii

Performance Tuning Guide

Precompile JSPs	2
Specifying a Different Java Compiler	4
Adjust the Intervals for Checking JSP and Servlet Modifications	5
About the Page-Check Interval Properties	6
About the Reload-Servlet Interval Property	7
To Adjust the Intervals	7
For More Information	7
Adjust Database Connections Available at Startup	8
For More Information	8
Set the Reload Policy for Rules	9
For More Information	9
Adjust Caching	10
Adjust and Use the Session and Global Caches	10
For More Information	11
Enabling the Caches	11
JSP Tags for Accessing HttpSession and the Session and Global Caches	12

An API for Accessing HttpSession and the Session and Global Caches	12
Guidelines for Placing Data in HttpSession, Session Cache, or Global Cache	13
Adjust Caching for Content Management.....	14
For More Information.....	16
Enable Property Caching.....	16
Property Caching in a Clustered Environment.....	16
To Enable Property Caching	17
For More Information.....	18
Enable Group Caching	19
Group Caching in a Clustered Environment	19
To Set Up the Group Cache Table	20
To Enable and Configure the Group Cache	20
To Access Data in the Group Cache Table	21
Adjust Portal and Portlet Settings While Load Testing.....	21
For More Information.....	22
Display Metadata, Sort and Query Explicit Metadata	23
For More Information.....	23
Use LDAP for Authentication Only	23
For More Information.....	23
Use the DocumentManager EJB.....	24
Use the HotSpot Server Virtual Machine	24
Deactivate the HotSpot Server VM on Windows.....	25
Activate the Hotspot Server VM on UNIX	25

About This Document

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. This document describes how to tune BEA WebLogic Commerce Server and Personalization Server for a production environment.

What You Need to Know

This document is intended mainly for Web-site administrators who configure properties for WebLogic Server and WebLogic Commerce Server and Personalization Server. It assumes a familiarity WebLogic Commerce Server and Personalization Server configuration files as well as the WebLogic Server platform and J2EE specifications.

e-docs Web Site

BEA product documentation is available on the BEA corporate Web site. From the BEA Home page, click on Product Documentation or go directly to the “e-docs” Product Documentation page at <http://e-docs.bea.com>.

How to Print the Document

You can print a copy of this document from a Web browser, one file at a time, by using the File—>Print option on your Web browser.

A PDF version of this document is available on the WebLogic Commerce Server and Personalization Server documentation Home page on the e-docs Web site (and also on the documentation CD). You can open the PDF in Adobe Acrobat Reader and print the entire document (or a portion of it) in book format. To access the PDFs, open the WebLogic Commerce Server and Personalization Server documentation Home page, click the PDF files button and select the document you want to print.

If you do not have the Adobe Acrobat Reader, you can get it for free from the Adobe Web site at <http://www.adobe.com/>.

Related Information

The following BEA documents describe configuring WebLogic Server and WebLogic Commerce Server and Personalization Server for optimal performance in a production environment:

- [BEA WebLogic Server Performance Tuning Guide](#)

Contact Us!

Your feedback on the BEA WebLogic Commerce Server and Personalization Server documentation is important to us. Send us e-mail at docsupport@bea.com if you have questions or comments. Your comments will be reviewed directly by the BEA professionals who create and update the WebLogic Commerce Server and Personalization Server documentation.

In your e-mail message, please indicate that you are using the documentation for the BEA WebLogic Commerce Server and Personalization Server 3.2 release.

If you have any questions about this version of BEA WebLogic Commerce Server and Personalization Server, or if you have problems installing and running BEA WebLogic Commerce Server and Personalization Server, contact BEA Customer Support through BEA WebSupport at www.bea.com. You can also contact Customer Support by using the contact information provided on the Customer Support Card, which is included in the product package.

When contacting Customer Support, be prepared to provide the following information:

- Your name, e-mail address, phone number, and fax number
- Your company name and company address
- Your machine type and authorization codes
- The name and version of the product you are using
- A description of the problem and the content of pertinent error messages

Documentation Conventions

The following documentation conventions are used throughout this document.

Convention	Item
boldface text	Indicates terms defined in the glossary.
Ctrl+Tab	Indicates that you must press two or more keys simultaneously.
<i>italics</i>	Indicates emphasis or book titles.

Convention	Item
monospace text	<p>Indicates code samples, commands and their options, data structures and their members, data types, directories, and file names and their extensions. Monospace text also indicates text that you must enter from the keyboard.</p> <p><i>Examples:</i></p> <pre>#include <iostream.h> void main () the pointer psz chmod u+w * \tux\data\ap .doc tux.doc BITMAP float</pre>
monospace boldface text	<p>Identifies significant words in code.</p> <p><i>Example:</i></p> <pre>void commit ()</pre>
<i>monospace</i> <i>italic</i> text	<p>Identifies variables in code.</p> <p><i>Example:</i></p> <pre>String <i>expr</i></pre>
UPPERCASE TEXT	<p>Indicates device names, environment variables, and logical operators.</p> <p><i>Examples:</i></p> <pre>LPT1 SIGNON OR</pre>
{ }	<p>Indicates a set of choices in a syntax line. The braces themselves should never be typed.</p>
[]	<p>Indicates optional items in a syntax line. The brackets themselves should never be typed.</p> <p><i>Example:</i></p> <pre>buildobjclient [-v] [-o name] [-f file-list]... [-l file-list]...</pre>
	<p>Separates mutually exclusive choices in a syntax line. The symbol itself should never be typed.</p>

Convention	Item
------------	------

...	Indicates one of the following in a command line: <ul style="list-style-type: none">■ That an argument can be repeated several times in a command line■ That the statement omits additional optional arguments■ That you can enter additional parameters, values, or other information The ellipsis itself should never be typed. <p><i>Example:</i></p> <pre>buildobjclient [-v] [-o name] [-f file-list]... [-l file-list]...</pre>
-----	--

.	Indicates the omission of items from a code example or from a syntax line. The vertical ellipsis itself should never be typed.
.	
.	



Performance Tuning Guide

When you first install BEA WebLogic Commerce Server and WebLogic 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 the [WebLogic Server Performance Tuning Guide](#) for information about tuning WebLogic Server.

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

- Precompile JSPs
- 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
- Use the HotSpot Server Virtual Machine

Precompile JSPs

By default, WebLogic Commerce Server and Personalization Server Web applications deactivate the JavaServer Page (JSP) precompile option. With this option deactivated, the server starts quickly but must compile each new or modified JSP when you access it, causing a significant delay the first time you request a new or modified JSP.

When you activate the precompile option, the server startup process checks for new or modified JSPs in the Web application and compiles them. Activating the precompile option can cause a significant delay in server startup if you have modified or added JSPs but avoids delays when you access a new or modified JSP for the first time.

To activate the precompile option for a Web application that is deployed as an expanded directory hierarchy, do the following:

1. From the Web application's `WEB-INF` directory, open the `web.xml` file in a text editor and insert the following element:

```
<context-param>
  <param-name>weblogic.jsp.precompile</param-name>
  <param-value>true</param-value>
</context-param>
```

Note: Some of the `web.xml` files shipped with the product already have this element in the file, but the value is set to `false`; in this case, change the `<param-value>` to `true`.

2. Save the file and restart the server.

To activate the precompile option for a Web application that is deployed as a `.war` file, do the following:

1. Make a backup copy of the `.war` file.
2. Create a temporary directory and copy the `.war` file to the directory.

3. In the temporary directory, unjar the .war file by entering the following command:

```
pathname\jar -xf WarFileName
```

For example:

```
c:\jdk1.3\bin\jar -xf tools.war
```

4. Under the temporary directory, open WEB-INF\web.xml in a text editor and insert the following element:

```
<context-param>  
  <param-name>weblogic.jsp.precompile</param-name>  
  <param-value>>true</param-value>  
</context-param>
```

Note: Some of the web.xml files shipped with the product already have this element in the file, but the value is set to `false`; in this case, change the `<param-value>` to `true`.

5. Save web.xml.
6. Under the temporary directory, if the WEB-INF directory contains a subdirectory named `_tmp_war`, delete the `_tmp_war` directory. This directory contains compiled JSPs and you must remove them before you rejar the .war file to ensure that WebLogic Commerce Server and Personalization Server recompile all JSPs the next time you start the server.
7. Remove the old .war file from the temporary directory.
8. Create a new .war file for the Web application by entering the following command:

```
pathname\jar -cf WarFileName *.*
```

For example:

```
c:\jdk1.3\bin\jar -cf tools.war *.*
```

-
9. Move the new `.war` file back to its original directory.
 10. Remove any other files in the original directory that may have been left over from previous `.war` extractions. For example, there may be a `WEB-INF` directory remaining from the last time you ran the Web application from the `.war` file.
 11. Restart the server.

The server console logs a message for each file it compiles. Ignore any `[JSP Enum] no match` messages. These are displayed for files that do not match the `.jsp` file extension.

Specifying a Different Java Compiler

The `weblogic.properties` file specifies a Java compiler for the resources WebLogic Server contains. You can override this property for your Web application by adding the following element to the Web application's `web.xml` file:

```
<context-param>
  <param-name>weblogic.jsp.compileCommand</param-name>
  <param-value>c:/jdk1.3/bin/javac.exe</param-value>
</context-param>
```

Change the `<param-value>` to specify the pathname of the Java compiler that you want to use for the Web application. For information on modifying a Web application's `web.xml` file, refer to “Precompile JSPs” on page 2.

Note: Some of the `web.xml` files shipped with the product already have this element in the file. In this case, change the `<param-value>`.

Adjust the Intervals for Checking JSP and Servlet Modifications

By default, each time a Web browser requests a JSP, WebLogic Commerce Server checks for any modifications to the JSP source file. Likewise, each time WebLogic 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 WebLogic Commerce Server serves JSPs and processes requests to servlets by increasing the intervals at which the server checks for modifications.

Although WebLogic 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 topic includes the following sections:

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

About the Page-Check Interval 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 application's deployment descriptor (`web.xml` file), which applies only to the servlets that WebLogic Server deploys in the context of the Web application.

The following excerpt from `WL_COMMERCE_HOME\server\Web apps\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 properties determine the frequency with which WebLogic 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 10.

About the Reload-Servlet Interval Property

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

The following excerpt from `WL_COMMERCE_HOME\server\Web apps\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 WebLogic 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 the *WebLogic Personalization Server Developer’s Guide*.

Set the Reload Policy for Rules

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

1. Open `$WL_COMMERCE_HOME/lib/ruleservice.jar`.
2. In `ejb-jar.xml` (which is in `ruleservice.jar`), modify the value for `rulesetReloadInterval`. This value expresses the number of milliseconds that WebLogic 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>
```

WebLogic 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 `ruleservice.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 `reloadNever`, WebLogic 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 the *WebLogic Personalization Server User’s Guide*.

Adjust Caching

To adjust caching for a 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 WebLogic 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 topic includes the following sections:

- 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 WebLogic Personalization Server process HTTP requests, refer to “[Foundation Classes and Utilities](#)” in the *WebLogic 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 13.

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 the *WebLogic 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 the [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 WebLogic 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 the *WebLogic 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 WebLogic 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 the *WebLogic Personalization Server Developer’s Guide*.

For more information about JSP tags for content management, see [“JSP Tag Library Reference”](#) in the *WebLogic 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 WebLogic 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 topic discusses the following sections:

- 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 the *WebLogic Personalization Server User’s Guide*.

For more information about JSP tags for managing property sets, see [“JSP Tag Library Reference”](#) in the *WebLogic 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 topic contains the following sections:

- 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 20.

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.axio
m.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 the 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
hierarchies 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 the [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 10. (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 WebLogic Personalization Sever from refreshing the working directory every 5 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 `t11`, increase the default number of milliseconds to prevent WebLogic 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 the *WebLogic Personalization Server User's Guide*.

For more information about developing portlets, see “[Developing Portlets](#)” in the *WebLogic 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 the *WebLogic Personalization Server Developer'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 the *WebLogic Personalization Server User's Guide*.

Use the DocumentManager EJB

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

Use the HotSpot Server Virtual Machine

HotSpot enhances JDK 1.3 performance by using a just-in-time compiler (JIT) and other features. It provides two implementations: a client Virtual Machine (VM) and a server VM. WebLogic Commerce Server and Personalization Server do not support the client VM. Instead, they support the server VM, which is specially tuned to maximize peak operating speed and is intended for long-running server applications.

By default, WebLogic Commerce Server and Personalization Server activate the HotSpot server VM by including the `-server` parameter in the JVM startup script, `startCommerce`. Before starting the server on Windows, you must download the server VM from www.javasoft.com and install it per the instructions on the Javasoft Web site.

Because HotSpot uses a JIT, you cannot access thread dumps while it is active. If you require thread dumps while developing and debugging your application, deactivate HotSpot by doing one of the following:

- Deactivate the HotSpot Server VM on Windows
- Activate the Hotspot Server VM on UNIX

Deactivate the HotSpot Server VM on Windows

1. Shut down WebLogic Commerce Server and Personalization Server and WebLogic Server.
2. Open the following file in a text editor, where `%WL_COMMERCE_HOME%` is the directory in which you installed WebLogic Commerce Server and Personalization Server:

```
%WL_COMMERCE_HOME%\StartCommerce.bat
```

3. Remove the `-server` parameter from the command that starts the JVM. For example:

```
%JDK_HOME%\bin\java -ms64m -mx128m
-classpath %JAVA_CLASSPATH%
-Dweblogic.class.path=%WEBLOGIC_CLASSPATH%
-Dweblogic.system.name=%SYSTEM_NAME%
-Dweblogic.system.home=%SYSTEM_HOME%
-Dweblogic.home=%WEBLOGIC_HOME% -Djava.security.manager
-Djava.security.policy=%WEBLOGIC_HOME%\weblogic.policy
-Dcommerce.properties=%WL_COMMERCE_HOME%\weblogiccommerce.properties
-Dweblogic.properties=%WL_COMMERCE_HOME%\weblogic.properties
-Dpipeline.properties=%WL_COMMERCE_HOME%\pipeline.properties
-Dwebflow.properties=%WL_COMMERCE_HOME%\webflow.properties
weblogic.Server
```

4. Save `StartCommerce.bat` and run it from a shell.

For information on starting WebLogic Commerce Server and Personalization Server without using `StartCommerce.bat`, refer to “Starting the JVM” under “[Starting the Server](#)” in the *Deployment Guide*.

Activate the Hotspot Server VM on UNIX

1. Shut down WebLogic Commerce Server and Personalization Server and WebLogic Server.
2. Open the following file in a text editor, where `$WL_COMMERCE_HOME` is the directory in which you installed WebLogic Commerce Server and Personalization Server:

```
$WL_COMMERCE_HOME/StartCommerce.sh
```

-
3. Replace the `-server` option with `-classic` for the command that starts the JVM. For example:
- ```
%JDK_HOME%\bin\java -ms64m -mx128m
-classic
-classpath %JAVA_CLASSPATH%
-Dweblogic.class.path=%WEBLOGIC_CLASSPATH%
-Dweblogic.system.name=%SYSTEM_NAME%
-Dweblogic.system.home=%SYSTEM_HOME%
-Dweblogic.home=%WEBLOGIC_HOME% -Djava.security.manager
-Djava.security.policy=%WEBLOGIC_HOME%\weblogic.policy
-Dcommerce.properties=%WL_COMMERCE_HOME%\weblogiccommerce.properties
-Dweblogic.properties=%WL_COMMERCE_HOME%\weblogic.properties
-Dpipeline.properties=%WL_COMMERCE_HOME%\pipeline.properties
-Dwebflow.properties=%WL_COMMERCE_HOME%\webflow.properties
weblogic.Server
```

The `-classic` option deactivates the HotSpot client VM and the server VM.

4. Save `StartCommerce.sh` and run it from a shell.

For information on starting WebLogic Commerce Server and Personalization Server without using `StartCommerce.bat`, refer to “Starting the JVM” under “[Starting the Server](#)” in the *Deployment Guide*.