BEA WebLogic Portal™

Performance Tuning Guide

Version 4.1
Document Date: May 2002

Performance Tuning Guide

Document Edition	Date	Software Version
4.1	March 2002	WebLogic Portal 4.0

Contents

About This Document

When you install BEA WebLogic Portal™, it is configured to support a development environment.

When you are ready to make your Web site available to customers, refer to this document for information about tuning WebLogic Portal performance for your testing and production environments.

This document includes the following topics:

- Chapter 1, “Load Testing Your Installation.”
- Chapter 2, “Factors Affecting Performance.”
- Chapter 3, “Estimating Baseline Capacity Requirements.”
- Chapter 4, “Using Caches.”
- Chapter 5, “Tuning JSPs and Servlets.”
- Chapter 6, “Additional Tuning Recommendations.”

All of the recommendations in this document are in addition to the recommendations in the *BEA WebLogic Server Performance and Tuning* guide.

What You Need to Know

This document is intended mainly for System Administrators and Database Administrators who configure properties for WebLogic Server and WebLogic Portal. It assumes a familiarity with WebLogic Portal, the WebLogic Server platform, J2EE specifications, as well as the database management system that your organization uses.

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 Portal documentation Home page on the e-docs Web site. A PDF version of this document is also available in the documentation kit on the product CD. Or you can download the documentation kit from the WebLogic Portal portion of the BEA Download site. 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 Portal 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 documents provide background and additional information that you may need to deploy WebLogic Server and WebLogic Portal:

- *Java™ 2 Platform Enterprise Edition Specification, v1.3*
- *BEA WebLogic Server Administration Guide*
- *Developing WebLogic Server Applications*

Contact Us!

Your feedback on the BEA WebLogic Portal 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 Portal documentation.

In your e-mail message, please indicate that you are using the documentation for the BEA WebLogic Portal **Product Version: 4.0** release.

If you have any questions about this version of BEA WebLogic Portal, or if you have problems installing and running BEA WebLogic Portal, 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.

Convention	Item
italics	Indicates emphasis or book titles.

`monospace text`	Indicates code samples, commands and their options, data structures and their members, data types, directories, and filenames and their extensions. Monospace text also indicates text that you must enter from the keyboard.
	Examples:
	`#include <iostream.h> void main ( ) the pointer psz`
	`chmod u+w *`
	`\tux\data\ap`
	`.doc`
	`tux.doc`
	`BITMAP`
	`float`

`monospace boldface text`	Identifies significant words in code. *Example*: `void` **`commit`** `( )`
`monospace italic text`	Identifies variables in code. *Example*: `String` *`expr`*
UPPERCASE TEXT	Indicates device names, environment variables, and logical operators. *Examples*: LPT1 SIGNON OR
`{ }`	Indicates a set of choices in a syntax line. The braces themselves should never be typed.

`[ ]`	Indicates optional items in a syntax line. The brackets themselves should never be typed.
	Example:
	`buildobjclient [-v] [-o name ] [-f` *`file-list`*`]...`
	`[-l` *`file-list`*`]...`

Convention	Item
|	Separates mutually exclusive choices in a syntax line. The symbol itself should never be typed.

`...`	Indicates one of the following in a command line:
	■ 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.
	Example:
	`buildobjclient [-v] [-o name ] [-f` *`file-list`*`]...`
	`[-l` *`file-list`*`]...`

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

CHAPTER

1 Load Testing Your Installation

This topic explains why load testing of the installation is important, and provides an outline of the steps necessary to complete this testing.

This topic includes the following sections:

- Rationale
- Steps for Load Testing Your Installation
  - General Architecture
  - WebLogic Portal

Rationale

BEA recommends that you establish an environment where you can load test the installation for the following reasons:

- Testing your prototype under load will help you validate design decisions that may significantly alter the performance of your application, while it is still early in the development cycle.
- Any tier within an n-tier architecture can dramatically affect application performance (hardware, database, clustering environment, application tuning parameters, and so on). Load testing your application whenever design changes

are made provides a way to narrow down performance problems to a particular area.

- Testing early and often increases the likelihood that your site implementation will be successful and scalable.

Steps for Load Testing Your Installation

The recommended approach for load testing is to start with the simplest aspect of the installation and then move into areas of increased complexity. If you observe slow behavior in any portion of this testing process, you should begin a more thorough investigation into its causes.

General Architecture

First, perform the following steps to identify performance issues with your network, database, or other software that is independent of the WebLogic Portal.

1. Test your database (independent of any Web components) to determine how well your schema and SQL work. Note any areas where the schema or SQL may not be optimized for performance. For more information about configuring database connection pools, see “Connection Pools” at http://e-docs.bea.com/wls/docs60/adminguide/jdbc.html
2. Test your network for sufficient bandwidth, and check that the TCP/IP parameters on the server’s operating system can sufficiently handle the application load you expect. It is quite possible that the network is the slowest aspect of your deployment.
3. Test your Web server, ensuring that it has sufficient capacity to serve static HTML pages when many concurrent threads are running.
4. Most large applications are clustered, but keep in mind that a clustering environment requires resources to perform load-balancing tasks (that is, the HTTPD proxy plug-in). Ensure that you have enough resources available to meet

application requirements. For more information, see “Configuring WebLogic Servers and Clusters” at
http://e-docs.bea.com/wls/docs60/adminguide/config.html

5. Test your servlet engine by running a load test against a trivial servlet such as a HelloWorld servlet. If this simple servlet does not perform and scale horizontally (meaning that as you add Java Virtual Machines, performance increases accordingly), the performance problems you encounter may be related to an infrastructure or resource issue.

WebLogic Portal

Now, perform the following steps to identify performance issues with WebLogic Portal:

1. Verify that your WebLogic Server database configuration is optimal. WebLogic Portal makes extensive use of the database. Check that your connection pool is large enough, and verify that your database handles connection failures in an efficient manner. For example, you may want to increase the number of connections at start up, the wait time before requesting new connections, whether your pool can shrink, and so on.
2. Verify that each portlet is optimized for speed as follows:
    a. Avoid using forms in a portlet that update the data within the portlet. This causes the entire portal to refresh its data, which can be very time consuming.
    b. Place items that require heavy processing in an edit page or a maximized URL. If you do not, the portal must wait for the portlet to process, and this considerably slows down the eventual rendering of the portal.
    c. Avoid large amounts of data retrieval that can take significant time to process.
3. Test your application’s components, starting from the data access layer. Then proceed toward the GUI one step at a time. Pay attention to performance and scalability differences at each component and between each layer of your application. Finally, do end-to-end testing from a browser-based load-testing tool.
4. Test the behavior and performance of your application under simulated, real-world conditions. (Many tools are available to help you do this.) Be sure to use both anonymous and logged-in users simultaneously.

2 Factors Affecting Performance

Application throughput is the amount of work processed by a system in a given period of time. A typical measurement of application throughput is the number of transactions (usually, requests) processed per second. This topic describes factors that can influence how much system hardware capacity will be required to support throughput requirements.

This topic includes the following sections:

- General Web Application Factors
- WebLogic Server Factors
- Campaign Factors
- WebLogic Commerce Services Factors
- WebLogic Personalization Services Factors

General Web Application Factors

The following paragraphs describe general factors that could impact your Web application's performance.

Complex Page Layouts

Intricate table layouts and other complex HTML can cause a perceived wait on the client after the files are transmitted and the browser determines how to render the page. Simple pages render more quickly.

Image Content

More images require more downloading from the server, lengthening the time it takes for a Web page to complete rendering. The size of each image file is also a factor affecting performance. Although there may only be a few, large image files can slow down delivery of a Web page. Keep the number of images you use on a page to a minimum, and be sure those you do use are a reasonable size. Additionally, the location of those images from a network perspective is important. For instance, images should not need to traverse firewalls before arriving at a user's browser.

Clustered Session Replication

Presently, a conservative approach is used for failover of HTTP session data. All client states are contained in the HTTP session, which provides a high degree of failover. In other words, the failure of a server in a cluster will only abort the current transaction, and the user session will be continued with no loss of data. (However, if the HttpSession contains an attribute that is not serializable, then the replication will not happen.)

Secure Sockets Layer (SSL)

The use of Secure Sockets Layer (SSL) in communication from a user's Web browser to a server or from server to server can affect overall throughput. SSL should be used when encryption of sensitive data is required while in transit or when strong server authentication is required. However, SSL should not be used unless it is absolutely needed.

WebLogic Server Factors

Because the WebLogic Portal runs on the BEA WebLogic Server, it is expected that factors impacting the performance of WLS will also impact the performance of WebLogic Portal.

Campaign Factors

The following are factors affecting performance of campaigns and related items.

Referencing Events

Always make scenario rules dependent on a particular event. This allows optimizations based on the event types referenced in the scenario rules.

Avoiding Firing Extraneous Events

Whenever possible, avoid firing any extraneous events. The campaign services must listen to all events. Use events to signify important occurrences on the site.

WebLogic Commerce Services Factors

The following paragraphs describe the factors specific to the BEA WebLogic Commerce Services that could potentially impact performance.

Cache Settings

The BEA WebLogic product catalog uses two in-memory caches for products and categories. The settings assigned to these caches can greatly affect the performance of an installation.

In general, most catalogs (except the largest) should cache all of the items and categories that will be accessed with any degree of frequency. For the purposes of capacity planning, the cache settings affect the amount of RAM allocated directly to these cache instances.

Note: For more information about these settings, see “Improving Catalog Performance by Optimizing the Catalog Cache” within “Catalog Administration Tasks” at http://edocs.bea.com/wlp/docs40/catalog/admin.htm.

Products

For exact product memory requirements, browse the `WLCS_PRODUCT` table in the database to determine the average number of characters used by each field in the database.

Then, multiply the result by the number of products in the database. The final result is the number of bytes of RAM that are required to cache all of the products. For a conservative estimate based on middle-of-the-road usage for every field in the schema, use 3200 bytes per product. For an example of these calculations, see “Example of Calculating the Necessary RAM”.

Note: Use four characters for dates and multiply character fields by two to account for Unicode encoding used by Java.

Categories

The fields considered in the calculations of category memory requirements are located in the `category` table in the database. However, the caching scheme for categories is more complex. In addition to the category data, the category’s position in the hierarchy is also cached. This makes it more important to cache categories than items. Specifically, the following information is stored about each category and is cached:

- Products in the category

- Sibling categories (peers in the hierarchy)
- Subcategories
- Ancestor categories
- Parent category

Note: Only primary keys are cached.

To arrive at the RAM requirement for category caching, add the data associated with the category as well as the space required by the hierarchy information (that is, the number of entries multiplied by the size of the appropriate key). For an example of these calculations, see “Example of Calculating the Necessary RAM”.

Example of Calculating the Necessary RAM

Suppose a site has 100,000 product items. The items are in a hierarchy with 15 top-level categories, 225 second-level categories (each category has 15 subcategories), and 3375 third-level categories (again, each category has 15 subcategories) for a total of 3615 categories. For simplicity, assume that the products are scattered evenly across all 3615 categories, and that each product exists in two different categories (56 items/category). Further, assume that product keys and category keys are each 10 characters long and will therefore occupy 20 bytes of RAM each. Lastly, each category’s database data occupies 1000 bytes, and each product item occupies 3000 bytes.

Product RAM Calculation

```
3000 bytes/item * 100000 items = 300 MB RAM
```

Therefore, 300 MB RAM is required to cache the whole catalog (which is recommended if possible).

Category RAM Calculation

```
1000 bytes/category * 3615 categories = 3.6 MB RAM
```

Therefore, 3.6MB RAM is required to cache the data.

Category Hierarchy RAM Calculation

- *Items*: 4.0 MB
  - `56 products/category * 20 bytes/key * 3615 categories = 4.0 MB`
- *Siblings*: approximately 230 MB
  - `15 categories * 14 siblings /category * 20 bytes/key = 4200 bytes` (Top level)
  - `225 categories * 224 siblings /category * 20 bytes/key = 1MB` (2nd level)
  - `3375 categories * 3374 siblings/category * 20 bytes/key = 229 MB` (3rd level)
- *Subcategories*: 72,300 bytes
  - `1 root category * 15 subcategories * 20 bytes/key = 300 bytes` (Root level)
  - `15 categories * 15 subcategories/category * 20 bytes/key = 4500 bytes` (Top level)
  - `225 categories * 15 subcategories/category * 20 bytes/key = 67,500 bytes` (2nd level)
- *Ancestors*: 211,800 bytes
  - `15 categories * 1 ancestor/category * 20 bytes/key = 300 bytes` (Top level)
  - `225 categories * 2 ancestors /category * 20 bytes/key = 9000 bytes` (2nd level)
  - `3375 categories * 3 ancestors/category * 20 bytes/key = 202,500 bytes` (3rd level)
- *Parents*: 72,300 bytes
  - `3615 categories * 20 bytes/key = 72,300 bytes`
- *Hierarchy Total*: **233 MB**.

RAM Totals

`300 MB` (product) + `3.6 MB` (category data) + `233 MB` (hierarchy) = `536.6 MB`

Therefore, approximately **537 MB** is required for catalog caching.

Catalog Size

The number of product items in the catalog tables and their corresponding attributes can have a significant effect on response time, especially when querying the catalog and making product recommendations. Because searching for and recommending products are key aspects of the browsing experience, it is important to ensure that your database is large enough to handle this product information; a slow database can limit performance of the whole application.

Catalog Update Frequency

When updating the product catalog directly in the database, it is recommended that the product item cache be disabled to prevent stale data. This is particularly true if category information is being changed. Running a server with the catalog cache disabled will place a greater burden on the commerce services database. This performance factor should be measured and planned for, based on the frequency of category and product updates. The cache can be left enabled if the updates are done through the server APIs, but the performance of that approach should be tested with a fully populated catalog.

Payment Settings

Settlement of commerce transactions can be done either in real-time or in batch mode (via the Administration Tools). The business model typically dictates which approach is taken. The real-time (online) approach will result in longer response times because the settlement process typically requires a network connection to a payment service. However, this approach guarantees that customers only pay for goods received. Batch settlement enables faster response to customers, but requires some back office processing to perform settlement.

Pipeline Session Data

At some point, your application will probably require that objects be stored in a Pipeline session. A large amount of data stored in the session (for example, search results) may affect performance and reduce the overall scalability of the application for clustered environments.

Deployment

In general, networked applications should be near (in a network sense) to the resources they utilize. If possible, for example, the WebLogic Server instances should be on the same network segment as the database.

Clustering your deployment is preferred, both for failover and for good performance. There are, however, some factors that can heavily influence the effectiveness of a clustered deployment. In particular, the location of the Web server's proxy plugin (relative to the cluster) is the most important factor.

In the preferred configuration, the Web servers between the external and internal firewalls (the DMZ) proxy requests for dynamic content served by WLS from within the internal network. The HTTP servers—not the application servers behind the firewalls—serve static content (HTML, GIFs, and so on).

This deployment is the highest performing as well as the most secure configuration. The network traffic between the proxy plugin is coarse-grained HTTP instead of the fine-grained RMI traffic used by EJBs, so traffic across the firewall is minimized. The fact that the plugin is running in the DMZ also minimizes the amount of logic executing in this area, and makes the system more secure.

Note: Cluster configurations other than the one described here will work, but with significant security and performance implications.

Location of Java Virtual Machines (JVMs)

Although clustered environments offer benefits in terms of both load balancing and failover, it is important to consider the location of clustered nodes as they relate to application scalability. If a single machine is assigned multiple IP addresses, scalability is improved because replication of HTTP session data does not require traversing the network. However, this obviously is less desirable where failover is critical (for example, in situations where customers should never lose their shopping cart). If you choose to run Java Virtual Machines (JVMs) on different machines to ensure failover, the scalability of your application might be negatively affected.

WebLogic Personalization Services Factors

The following paragraphs describe the factors specific to the BEA WebLogic Personalization Services that can have an impact on performance.

Accessing the Database

The number of times the database is accessed while generating a portlet or Web page can have a significant effect on application performance. If this number is kept low, performance might benefit. Alternatively, numerous queries to the database can hurt performance.

Application Complexity

Complex applications have the potential to incur performance bottlenecks. For example, an application that must access a database, two mainframes, an LDAP server over SSL, and a secondary Web server is going to have more potential for degraded performance than an application that simply queries a single, local database. Designers of complex applications must carefully consider all potential bottlenecks and address them with appropriate solutions, such as a data caching strategy.

CHAPTER

3 Estimating Baseline Capacity Requirements

This topic explains how you can use the information described in prior topics to obtain a baseline environment that meets your specific needs.

This topic includes the following sections:

- Estimate the Maximum Number of Simultaneous Users
- Estimate Per-User Memory Requirements
- Estimate Application Throughput Requirements
- Estimate Maximum Acceptable Response Times
- Calculate a Baseline Estimate
  - Throughput Requirements
  - Memory Requirements
  - Latency Requirements
- Adjusting Your Estimate

Estimate the Maximum Number of Simultaneous Users

The first step in calculating a baseline environment for your application is to estimate the maximum simultaneous user load that your application will be expected to handle. This estimate depends on the nature of your application and on the behavior of your customers. For example, your requirements may stipulate that your application be able to support 1000 concurrent client connections.

Estimate Per-User Memory Requirements

Each simultaneous user connection requires memory to store information on a per-user basis. For example, an online commerce site must store the contents of each user’s shopping cart. Ideally, we would like to be able to store each shopping cart in memory rather than in the database. Therefore, where possible, allocate enough memory so that your application stores all concurrent client data in memory.

Estimate Application Throughput Requirements

Next, determine the maximum rate at which clients will make requests to your application. This estimate will depend on the nature of the content served by your application and on the behavior of your users. In the case mentioned above, the application must be able to support 1000 concurrent clients. Suppose you expect users to make new requests every 25 seconds. In this case, the application must be able to process 1000/25 = 40 transactions per second (TPS), on average. However, the peak application throughput requirements will be larger than this average figure. Assume that peak workload is 150% of the average figure. This translates into a peak throughput requirement of 60 TPS.

Estimate Maximum Acceptable Response Times

Finally, an application response time goal should be associated with the application throughput goal. This figure would typically be inclusive of wide area network latency, implying a lower application response time goal.

Calculate a Baseline Estimate

The estimates of the maximum simultaneous users, per-user memory requirements, application throughput requirements, and maximum response times can be used to calculate a baseline environment for your specific needs as described in the following sections.

Throughput Requirements

To satisfy the throughput requirements of your application, choose the appropriate hardware deployment configuration such that:

```
Maximum throughput requirement = TPS1 + TPS2 + … + TBSn
```

where `TPS1`, `TPS2`, …`TBS`*n* can be estimated using the throughput observations found in the results section of each test scenario. It is important to use the observations from the scenarios and parameters that best mimic your application. If your application uses Commerce and Personalization services and campaign features, it is best to use the lower of the two TPS observations as a baseline estimate.

Memory Requirements

Use the following formula to determine the minimum amount of memory that must be allocated to each of your *n* WebLogic Commerce and Personalization services processes:

```
Server minimum + Per-user memory requirements * (Maximum number of users)/n
```

Latency Requirements

When trying to satisfy latency requirements, it is important to factor in the effect of the network and deployment configurations on application latency. For example, it is extremely important to optimize the network connection between your database server(s) and your application server(s). Furthermore, the addition of firewalls and proxy services (such as the cluster proxy plugin) can increase application latency. System throughput tests that include response times can help you optimize your physical architecture such that latency times are minimized.

Adjusting Your Estimate

The advice provided in this Guide should get you moving in the right direction for your particular capacity planning efforts. However, you will need to perform tests in your own environment and adjust these estimates according to your results before deploying your application. Environment-specific testing is especially important in determining exact scalability numbers. By performing your own tests, you can be confident that your application will perform as well as it possibly can for your users.

CHAPTER

4 Using Caches

WebLogic Portal provides a single framework for configuring, accessing, monitoring, and maintaining caches. If configured properly, the caches can reduce the time needed to retrieve frequently used data.

Many WebLogic Portal services use preconfigured caches that you can tune to meet your performance needs. Some services use internally configured caches that you cannot configure or access. If you extend or create additional services, you can use the cache framework to define and use your own set of caches.

This topic contains the following sections:

- Overview of the Cache Framework
- Configuring Caches with MBeans
- Accessing Caches
- Reconfiguring Active Caches
- Monitoring and Managing Caches
- Recommended Settings

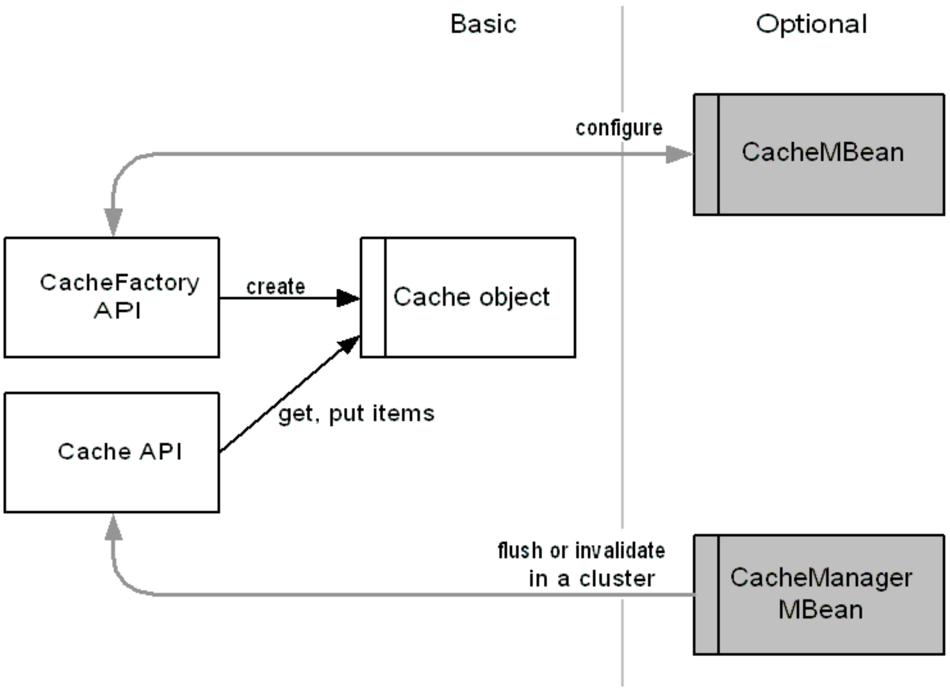
All of the recommendations in this document are in addition to the recommendations in the *BEA WebLogic Server Performance and Tuning* guide.

Overview of the Cache Framework

The most basic components of the cache framework are the `com.bea.p13n.cache.CacheFactory` and `com.bea.p13n.cache.Cache` classes. These classes provide facilities for creating, accessing, and managing caches.

In addition to these components, WebLogic Portal provides a set of MBeans (Java Beans for Management) that you can use to configure and administer the caches. (MBeans are part of the specification for Java Management Extensions, or JMX.) See Figure 4-1.

Figure 4-1 The Cache Framework



Comparison of Configuring Caches

Using MBeans to configure caches is optional. This section compares configuring caches with and without MBeans.

Using MBeans to configure caches provides the following advantages:

- A cache's configuration persists when you restart a server instance.
- You can use the WebLogic Server Administration Console (or an API) to dynamically modify a cache configuration.
- The configuration is available to all instances of the cache on all nodes of a cluster. Any changes that you make to this configuration via the WebLogic Server Administration Console are propagated to all instances of the cache on all cluster nodes.

Instead of using MBeans to configure caches, you can use a default configuration that is specified in `com.bea.p13n.cache.CacheDefaults`. You cannot use the WebLogic Server Administration Console to modify the configuration of caches that you create in this way. (Hence, any changes that you make to the configuration of such a cache are not propagated across a cluster.)

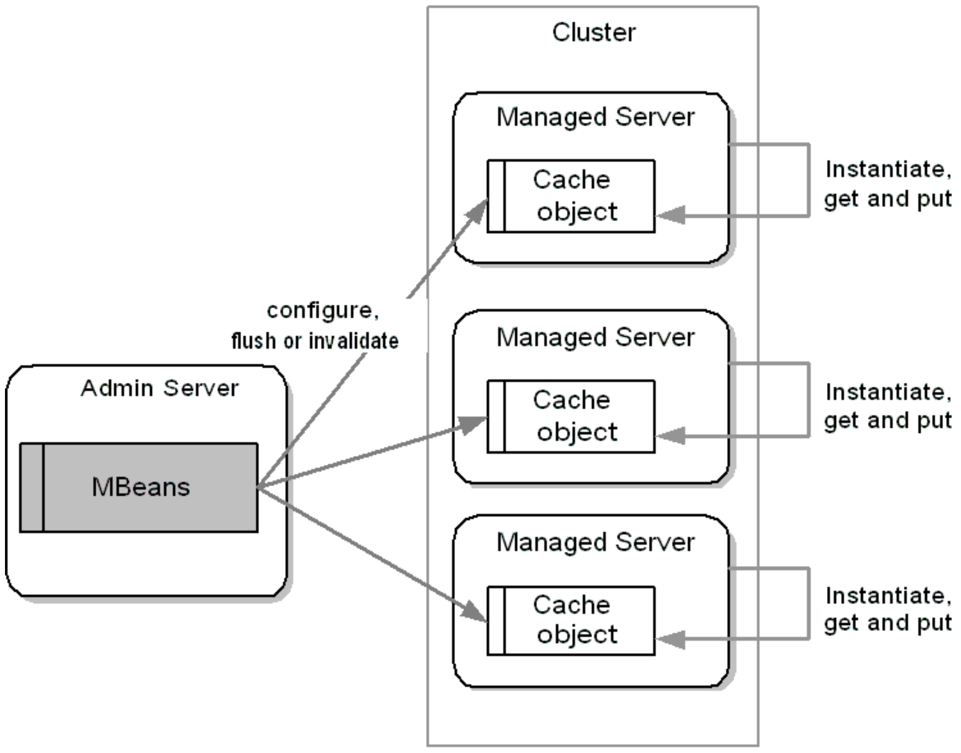
Caches in a Clustered Environment

In a clustered environment, entries within caches are not replicated across nodes.

If you use MBeans to configure a cache, the CacheMBean on the Administration Server propagates the cache configuration to all nodes in the cluster. If you modify the configuration, the CacheMBean propagates the changes to all instances of the cache throughout the cluster, thus maintaining a consistent view of the cache configuration parameters. (See Figure 4-2.)

Note: If you want to dynamically reconfigure an active, single instance of a cache without propagating the changes throughout a cluster, refer to "Reconfiguring Active Caches" on page 4-11.

Figure 4-2 Caches in a Cluster



In addition, for caches that are configured by MBeans, you can use the WebLogic Server Administration Console to do the following:

- Flush (remove) all items from all instances of the cache throughout a cluster.
- Invalidate a single item in a cache and propagate the invalidation to all instances of the cache throughout a cluster.

For example, a customer logs in to node A and receives a discount while shopping. The Discount Service on node A uses the `discount` cache to store data. A different customer logs in to node B and receives a discount while shopping. The Discount Service on node B creates its own instance of the `discount` cache to store its data. In the meantime, your supply of one of your discounted items unexpectedly runs low. You use the E-Business Control Center to deactivate the discount for the item, and to make sure that the discount is deactivated immediately (as opposed to waiting for the cache TTL to time out), you use the WebLogic Server Administration Console to flush the `discount` cache on all nodes in the cluster.

Configuring Caches with MBeans

Before you can access the cache MBeans, you must do the following:

- Assemble and deploy an enterprise application with a deployment descriptor that declares the default cache MBeans. For information on setting up a deployment descriptor for your enterprise application, refer to “Create Deployment Descriptors” under “Assembling and Deploying Enterprise Applications” in the *Deployment Guide*.
- Start the server instance onto which you deploy your enterprise application.

Configuring Default Caches

Many WebLogic Portal services use preconfigured caches that you can tune to meet your performance needs.

If you want to re-configure these default caches, do the following:

1. Start the WebLogic Server Administration Console by entering the following URL in a Web browser:

   `http://`*`hostname`*`:`*`port-number/`*`console`

   For example, you started a server on a host named bonnie and it uses port 7001 as a listen-on port. Enter the following URL:

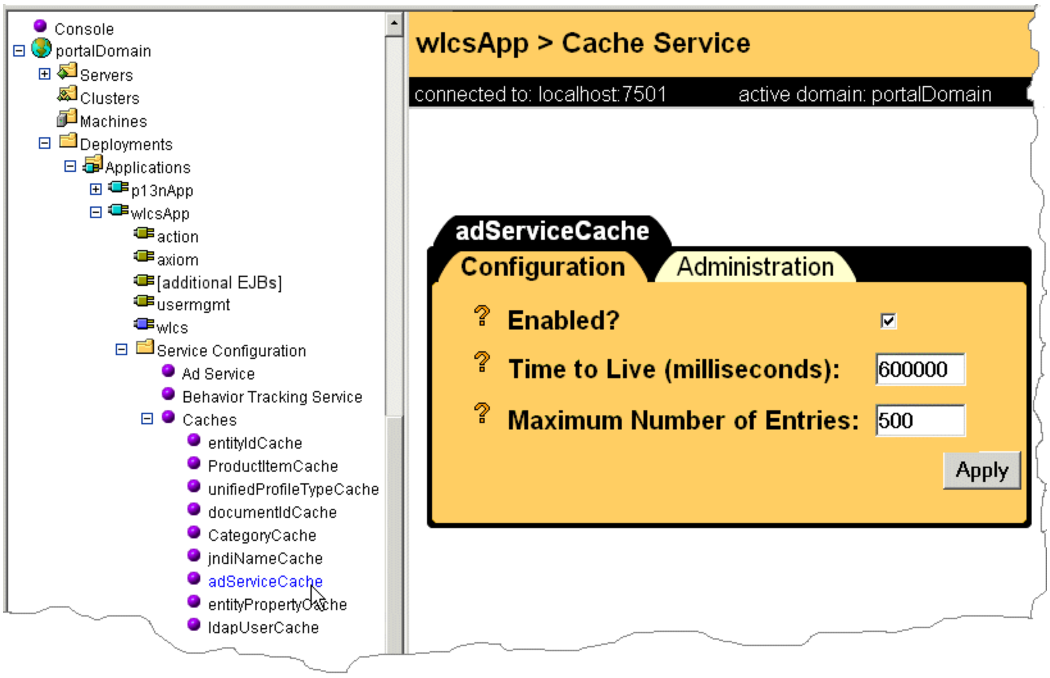
   `http://bonnie:7001/console`

2. The WebLogic Server Administration Console prompts you to log in with a user account that has administrator privileges.

3. After you log in, in the left pane, click Deployments → Applications → *MyApplication* → Service Configuration → Caches.

4. Click the cache that you want to configure. (See Figure 4-3.)

Figure 4-3 Configuring a Cache MBean



connected to: localhost:7501 active domain: portalDomain

Enabled? ☑

Time to Live (milliseconds): 600000

Maximum Number of Entries: 500

5. On the Cache Service page, click the Configuration tab and modify any of the following items:

Name	**Description**
Enabled	Determines whether the cache can be instantiated and accessed. Clearing this check box also flushes the cache and causes the fetch, add, and remove methods to do nothing.
Time to Live	Determines the number of milliseconds that items in the cache remain valid. For information on setting TTL values for individual items in a cache, refer to "Putting and Getting Items in a Cache" on page 4-10.
Maximum Number of Entries	Determines the maximum number of items that the cache maintains. If a cache already contains the maximum number of entries and a service tries to add an item, the cache does one of the following: If the item already exists, the cache replaces the entry. If the item does not already exist, it is added to the cache and the least recently used element is removed to accommodate the new element. Caches remove items to accommodate new ones only if the cache is at capacity.

6. Click Apply. Your modifications take effect immediately.

The WebLogic Server Administration Console saves these values to the `application-config.xml` file, which is located in the `META-INF` directory of your enterprise application.

Configuring Custom Caches

If you create a cache to support one of your extensions or additional services, you can use MBeans to configure the cache. To use MBeans to configure a custom cache, do the following:

1. In a text editor, open `myApplication/META-INF/application-config.xml`.

   Note: Do not use the WebLogic Server Administration Console for your domain while you are editing `application-config.xml` in the text editor. The WebLogic Server Administration Console writes its changes to `application-config.xml`.

2. In `application-config.xml`, within the `<CacheManager>` element, add the following subelement:

   ```
   <Cache
      Name="cache-name"
      TimeToLive="time-in-milliseconds"
      MaxEntries="numerical value">
   </Cache>
   ```

   For example,

   ```
   <Cache
       Name="MyCache"
       TimeToLive="360000"
       MaxEntries="100">
   </Cache>
   ```

3. Restart the server or redeploy your application.

To access this cache, use `CacheFactory.getCache(cache-name)`.

Accessing Caches

To access (instantiate and retrieve) a cache, you use `CacheFactory.getCache()`. This section contains the following subsections:

- Accessing a Cache
- Putting and Getting Items in a Cache

For information about the advantages of each configuration method, refer to "Comparison of Configuring Caches" on page 4-3. For more information about `com.bea.p13n.cache.CacheFactory.getCache()`, refer to the WebLogic Portal *Javadoc*.

Accessing a Cache

Use the following syntax to access a cache:

```
CacheFactory.getCache(String cacheName)
```

With this command, `getCache()` does the following:

- Determines whether a cache identified by the *`cacheName`* value has already been instantiated. If it has, then it retrieves the existing cache.
- If it has not, then `CacheFactory` looks for an instance of a CacheMBean with a name attribute that matches *`cacheName`*. If it finds such an MBean, then it creates a cache using the MBean configuration parameters.
- If `CacheFactory` does not find an CacheMBean that matches *`cacheName`*, then `getCache()` configures the cache with default parameters from `com.bea.p13n.cache.CacheDefaults`:
  ```
  MaxEntries=100
  TimeToLive=360000
  Enabled=true
  ```

For example, the following command returns a cache named MyCache:

```
Cache cache = CacheFactory.getCache("MyCache");
```

Putting and Getting Items in a Cache

After you retrieve a cache, you use the following methods to put and get items:

- `Cache.put(`*`Object key`*`, `*`Object value`*`)` to add items.
- `Cache.put (`*`Object key`*`, `*`Object value`*`, `*`long ttl`*`)` to add and specify a time-to-live (in milliseconds) for an item. This item-specific TTL does not persist and does not propagate to other instances of the cache in a cluster. The TTL for other items in the cache is determined by the cache-wide TTL, which you specify when you create the cache.
- `Cache.get(`*`Object key`*`)` to retrieve items.

For example, the following commands retrieve a cache named MyCache, add three items that are identified as Element 1, Element 2, and Element 3, and then retrieves the value for Element 2.

```
Cache cache = CacheFactory.getCache("MyCache");
cache.put("Element 1", "stream");
cache.put("Element 2", "mountain");
cache.put("Element 3", "stars");
String s = (String)cache.get("Element 2");
```

The following command specifies a TTL of 6000 milliseconds for the Element 2 item,

```
cache.put ("Element 2", "mountain", 6000);
```

For a description of all `Cache` API methods, refer to the WebLogic Portal *Javadoc*. for `com.bea.p13n.cache.Cache`.

Reconfiguring Active Caches

If you configured a cache with MBeans, you can use the WebLogic Server Administration Console to reconfigure the cache and all of its instances across a cluster. For more information, refer to “Configuring Caches with MBeans” on page 4-5.

If you want to reconfigure a single instance of any currently active cache, invoke `Cache.set` methods. These configuration changes do not use MBeans and are not propagated to other instances the cache in a cluster.

For more information, refer to the WebLogic Portal *Javadoc*. for `com.bea.p13n.cache.CacheFactory.Cache`.

Monitoring and Managing Caches

You can use the APIs to monitor cache activity and the WebLogic Server Administration Console or APIs to perform management tasks such as removing items.

This section contains the following subsections:

- Monitoring a Cache
- Flush or Invalidate Items in a Cache That Is Configured with MBeans
- Clear or Remove Items from a Cache

Monitoring a Cache

Each `cache` object maintains and reports information about itself. The following sections describe how to monitor an individual instance of a cache:

- Viewing the Number of Items in a Cache
- Viewing the Hit Count
- Viewing the Hit Rate
- Viewing the Miss Count
- Example of Monitoring Caches

Note: Cache statistics, such as hit count and hit rate, describe activity for individual instances of the cache. You cannot view them from the WebLogic Server Administration Console. Compare this to cache attributes, such as `enabled`, which apply to all instances of a named cache in a cluster and are available to view or modify from the WebLogic Server Administration Console.

Viewing the Number of Items in a Cache

To view the number of items in a cache, use the following API:

```
com.bea.p13n.cache.Cache.size()
```

For information about this method, refer to the documentation for `com.bea.p13n.cache.Cache` in the WebLogic Portal *Javadoc*.

Viewing the Hit Count

To view the hit count, which is the number of `Cache.get()` calls that successfully returned data, use the following API:

```
com.bea.p13n.cache.getHitCount()
```

For information about this method, refer to the documentation for `com.bea.p13n.cache.CacheStats` in the WebLogic Portal *Javadoc*.

Viewing the Hit Rate

To view the hit rate, which is the percentage of `Cache.get()` calls that successfully returned data, use the following API:

```
com.bea.p13n.cache.getHitRate()
```

For information about this method, refer to the documentation for `com.bea.p13n.cache.CacheStats` in the WebLogic Portal *Javadoc*.

Viewing the Miss Count

To view the miss count, which is the number of `Cache.get()` calls that did not return data, either because an entry has expired or was not in the cache, use the following API:

```
com.bea.p13n.cache.getMissCount()
```

For information about this method, refer to the documentation for `com.bea.p13n.cache.CacheStats` in the WebLogic Portal *Javadoc*.

Example of Monitoring Caches

The following commands retrieve a cache named FooCache and return the statistics that the `cache` object keeps. The final line resets the statistics:

```
Cache cache = CacheFactory.getCache(“FooCache”);
//do stuff; put, get items
int hitRate = cache.getHitRate();
int hitCount = cache.getHitCount();
int missCount = cache.getMissCount();

com.bea.p13n.cache.resetStats();
```

Disable a Cache

If you want to make a cache unavailable for putting and getting items, you can disable it. Disabling also flushes a cache and causes the fetch, add, and remove methods to do nothing.

If you use the WebLogic Server Administration Console to configure a cache, do the following to disable the cache:

1. Start the WebLogic Server Administration Console by entering the following URL in a Web browser:

   `http://hostname:port-number/console`

   For example, you started a server on a host named bonnie and it uses port 7001 as a listen-on port. Enter the following URL:

   `http://bonnie:7001/console`

2. The WebLogic Server Administration Console prompts you to log in with a user account that has administrator privileges.
3. After you log in, in the left pane, click Deployments → Applications → *MyApplication* → Service Configuration → Caches.
4. Click the cache that you want to disable. (See Figure 4-3.)
5. On the Cache Service page, click the Configuration tab.
6. Clear the Enabled check box and click Apply.

Note: The `Cache.disable()` API also disables caches, but we recommend that you do not use this method for caches that are configured with MBeans.

If you do not use the WebLogic Server Administration Console and MBeans to configure a cache, use `Cache.setEnabled(false)` to disable a cache.

Flush or Invalidate Items in a Cache That Is Configured with MBeans

For any cache that is configured with MBeans, you can flush or invalidate items cluster-wide. Flushing removes all items from all instances of the cache throughout a cluster. Invalidating removes a single item from all instances of the cache throughout a cluster.

Flushing or Invalidating From the WebLogic Server Administration Console

To flush or invalidate from the WebLogic Server Administration Console, do the following:

1. Start the WebLogic Server Administration Console by entering the following URL in a Web browser:

   `http://`*`hostname`*`:`*`port-number/`*`console`

   For example, you started a server on a host named bonnie and it uses port 7001 as a listen-on port. Enter the following URL:

   `http://bonnie:7001/console`

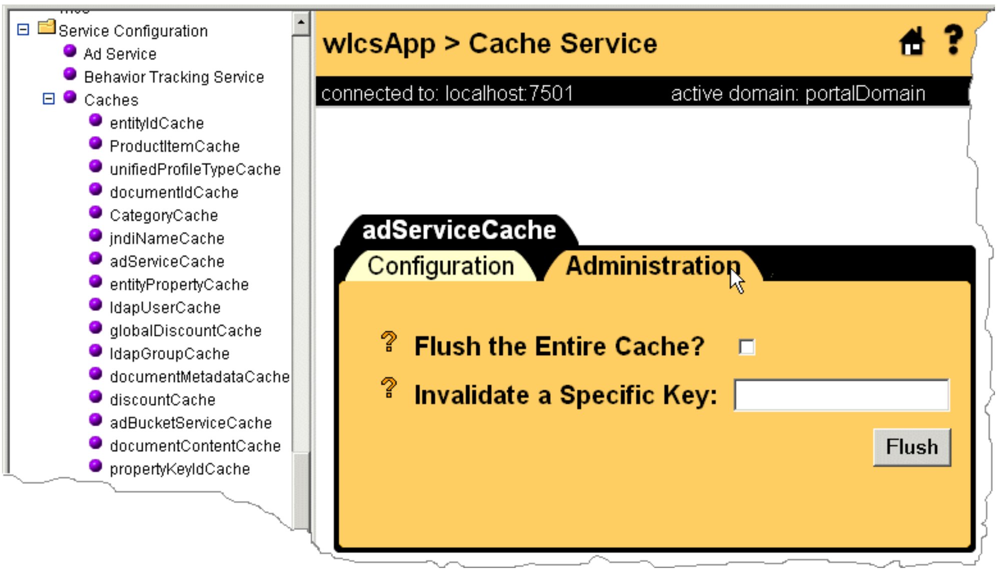
2. The WebLogic Server Administration Console prompts you to log in with a user account that has administrator privileges.

3. After you log in, in the left pane, click Deployments → Applications → *MyApplication* → Service Configuration → Caches.

4. Click the cache that you want to flush or invalidate.

5. On the Cache Service page, click the Administration tab. (See Figure 4-4.)

Figure 4-4 Cache Administration



6. To flush all items from a cache, select the Flush the Entire Cache check box. Then click Flush.

   To invalidate an item, enter the item’s key in the Invalidate a Specific Key box. Then click Flush.

Clear or Remove Items from a Cache

For any cache that is currently active, you can clear or remove items. Clearing removes all items from a cache; removing destroys only the items that you specify. Both of these operations apply only to the current instance of a cache. They do not affect other instances on other nodes of a cluster.

To clear all items from a cache, retrieve the cache and then use the `cache.clear` method:

```
CacheFactory.getCache(String name);
cache.clear()
```

For example, the following two commands retrieve a cache named MyCache and clear all items from it:

```
CacheFactory.getCache(“MyCache”);
cache.clear()
```

To remove items from a cache, retrieve the cache and then use the `cache.remove` method:

```
CacheFactory.getCache(String name);
cache.remove(Object key)
```

For example, the following two commands retrieve a cache named MyCache and remove an entry named Element 2:

```
Cache cache = CacheFactory.getCache(“MyCache”);
cache.remove(“Element 2”)
```

Recommended Settings

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

- Adjust Caching for Content Management
- Property Caching in a Clustered Environment
- Adjust Group Membership TTL in the Caching Realm
- Adjust Caching for the Discount Service

Adjust Caching for Content Management

To optimize content-management performance for your production Web site, the Content Manager uses the caching framework to configure and manage the following caches:

- `documentContentCache`
- `documentMetadataCache`
- `documentIdCache`

The content management JSP tags provide an additional set of caches, which you can access by doing the following:

- 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 Rules Manager.

  For information on using the `useCache` attribute, refer to “JSP Tag Library Reference” in the *Guide to Building Personalized Applications*.

  To clear cached content when user and/or document attributes change, use the `remove` method of `com.bea.p13n.content.ContentCache`. For more information, see the WebLogic Portal *Javadoc*. for `com.bea.p13n.content.ContentCache`.

- For the `cm:select`, `cm:selectById`, `pz:contentQuery`, and `pz:contentSelector` JSP tags, set the `cacheScope` attribute to `application` whenever possible. This `application` scope applies to the Web application, not to the enterprise application.

  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, you create an application that 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, your application 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 Portal 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.bea.p13n.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 *Guide to Building Personalized Applications*.

For more information about JSP tags for content management, see “JSP Tag Library Reference” in the *Guide to Building Personalized Applications*.

Property Caching in a Clustered Environment

To decrease the amount of time needed to access user, group, and other properties data, the WebLogic Server Configurable Entity and Entity Property Manager use the cache framework to configure and manage the following caches:

- `ldapGroupCache`
- `ldapUserCache`
- `entityPropertyCache`
- `entityIdCache`
- `unifiedProfiletypeCache`
- `propertyKeyIdCache`

By default, these property caches are enabled.

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 caches 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.

If a user and an administrator are pinned to different servers in the cluster and the administrator changes a user's properties, the user may not see the changes during the current session. You can mitigate this situation by specifying a small Time-To-Live (TTL) setting. For information on adjusting the TTL setting for a cache, refer to “Configuring Caches with MBeans” on page 4-5.

If you require multiple servers in a cluster to have immediate access to modified properties, disable property caching. For information on disabling a cache, refer to “Disable a Cache” on page 4-14.

For More Information

For more information about property sets, see “Creating and Managing Property Sets” in the *Guide to Building Personalized Applications*.

For more information about JSP tags for managing property sets, see “JSP Tag Library Reference” in the *Guide to Building Personalized Applications.*

Adjust Caching for the Discount Service

To reduce the amount of time the Order and Shopping Cart services need to calculate order and price information that include discounts, the Discount Service uses the caching framework to create and manage the following caches:

- `discountCache`, which contains data for campaign discounts. Campaign discounts are targeted to specific customers or customer segments, and are available only in the context of a campaign.
- `globalDiscountCache`, which contains data for global discounts. Global discounts apply to all customers, regardless of customer properties or customer segments.

When a customer adds an item to the shopping cart, removes an item from the shopping cart, checks out, or confirms an order, the Pricing Service is responsible for determining the price of the items in the cart. To calculate the effect of discounts on the shopping cart, the Pricing Service requests the Discount Service to retrieve information about all global discounts and about any campaign discounts that apply to the current customer.

The first request for information about discounts requires a separate call to the database for each discount that applies. For example, if you have defined one global discount and if a customer is eligible for two campaign-related discounts, the Discount Service makes three calls to the database. To decrease the response time for any subsequent requests, the Discount Service uses the caches.

This section contains the following subsections:

- Adjusting the discountCache
- Adjusting the globalDiscountCache
- Discount-Service Caches in Clustered and Non-Clustered Environments

Adjusting the discountCache

The `discountCache` contains data for campaign discounts. For maximum performance, set the capacity to the number of campaign discounts that are currently deployed. A larger capacity will potentially use more memory than a smaller capacity.

The Time-To-Live (TTL) property determines the number of milliseconds that the Discount Service keeps the information in the cache. After the cache value times out, the next request for the value requires the Discount Service to call the database to retrieve the information and then cache the value. A longer TTL decreases the number of database calls made over time when requesting cached objects. In a clustered environment, the TTL is the maximum time required to guarantee that any changes to global discounts are available on all servers.

Adjusting the globalDiscountCache

The Maximum Number of Entries property for global caches does not need to be modified.

The time-to-live property determines the number of milliseconds that the Discount Service keeps information in the global-discount cache. After the Time-To-Live (TTL) expires, the next request for global discount information requires the Discount Service to call the database to retrieve the information and then cache the value. A longer TTL decreases the number of database calls made over time when requesting cached objects. In a clustered environment, the TTL is the maximum time required to guarantee that any changes to campaign discounts are available on all servers.

Discount-Service Caches in Clustered and Non-Clustered Environments

In either environment (clustered or non-clustered), when you change a discount priority, end date, or its active/inactive state, WebLogic Portal flushes the discount from the appropriate cache. Changes to a campaign discount flush only the specific discount from the campaign-discount cache. Changes to a global discount flush all discounts from the global-discount cache.

For example, you log in to a WebLogic Portal host named `bread` and deactivate a campaign discount named `CampaignDiscount1`. WebLogic Portal flushes the `CampaignDiscount1` from the campaign-discount cache on `bread`.

In a clustered environment, other machines in the cluster continue to use their cached copy of the discount until the TTL for that discount expires.

Adjust Group Membership TTL in the Caching Realm

The WebLogic Server Caching realm stores the results of both successful and unsuccessful realm lookups. It does not use the WebLogic Portal caching framework.

The Caching realm manages separate caches for Users, Groups, permissions, ACLs, and authentication requests. It improves the performance of WebLogic Server by caching lookups, thereby reducing the number of calls into other security realms.

WebLogic Portal enables the Caching realm by default. While all of the caches in the Caching realm can improve performance, the Time-To-Live (TTL) value for the Group Membership Cache in particular can affect the performance of WebLogic Portal.

In addition, note that if you delete a user from the system without first removing the user from a group, then the system continues to recognize the user until the TTL for the Group Membership Cache expires.

For information on adjusting the Group Membership TTL, refer to “Configuring the Caching Realm” under “Managing Security” in the *WebLogic Server Administration Guide*.

CHAPTER

5 Tuning JSPs and Servlets

JSPs are the front end of your application. When a customer requests a page on your e-business Web site, WebLogic Portal compiles the corresponding JSP into a servlet. In addition to servlets that come from compiled JSPs, WebLogic Portal provides a set of servlets for exchanging information between various components of the system.

For suggestions on tuning the compiling and updating for JSPs and servlets, refer to the following sections:

- Precompile JSPs
- Specifying a Java Compiler for a Web Application
- Adjust the Intervals for Checking JSP and Servlet Modifications

All of the recommendations in this document are in addition to the recommendations in the *BEA WebLogic Server Performance and Tuning* guide.

Precompile JSPs

For each of your Web applications that you deploy, you can determine when WebLogic Portal compiles JSPs:

- You can specify that the application server compiles (precompiles) all JSPs when you start the server.

  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.

- You can specify that the application server compiles JSPs only when they are requested.

  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.

By default, the sample Web applications for WebLogic Portal deactivate the JavaServer Page (JSP) precompile option.

To precompile JSPs 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 `weblogic.xml` file in a text editor and find the following element:

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

   Note: `web.xml` was changed to `weblogic.xml` in this release.

   Note: A patch for WebLogic Server 6.1 is required for this functionality to work. The patches are included in `60sp2rp2`, `60sp2rp1`,and `61sp2`.

2. In the `<param-value>` element, replace `false` with `true`. For example, `<param-value>true</param-value>`

3. Save the file and restart the server.

For information, refer to "Starting and Shutting Down the Server" in the *Deployment Guide*.

To precompile JSPs 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 find the following element:

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

5. In the `<param-value>` element, replace `false` with `true`. For example, `<param-value>true</param-value>`
6. Save `web.xml`.
7. 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 re-jar the `.war` file to ensure that WebLogic Portal recompile all JSPs the next time you start the server.
8. Remove the old `.war` file from the temporary directory.
9. 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 *.*
   ```

10. Move the new `.war` file back to its original directory.

11. 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.

12. Restart the server.

    For information on shutting down and starting the server, refer to “Starting and Shutting Down the Server” in the *Deployment Guide*.

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 Java Compiler for a Web Application

The WebLogic Server Administration Console specifies a Java compiler for each server configuration. All applications that you deploy on a server use this compiler unless a Web application’s `weblogic.xml` file specifies a different compiler.

To review the current Java compiler for your server, in the left pane of the WebLogic Server Administration Console, click a server. In the right pane, on the Configurations tab, click the Compilers subtab.

To specify a Java compiler for a Web application, do the following:

1. Open Web application’s `WEB-INF/weblogic.xml` file in a text editor.

2. In `weblogic.xml`, find the following element:

```
<--
<jsp-param>
      <param-name>compileCommand</param-name>
      <param-value>java-compiler</param-value>
</jsp-param>
-->
```

3. Remove the `<--` and `-->` comment tags.

4. Change the `<param-value>` to specify the pathname of the Java compiler that you want to use for the Web application.
5. To deploy any modifications to this file, you must restart the server.

Adjust the Intervals for Checking JSP and Servlet Modifications

You can specify how frequently a server checks for modifications to JSPs and source files for other servlets in a Web application.

The sample Web applications check for modified JSPs each time a Web browser requests a JSP. Likewise, each time the server sends a request to a servlet in a sample Web application, 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 Portal serves JSPs and processes requests to servlets by increasing the intervals at which the server checks for modifications.

Although the 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
- To Adjust the Intervals

About the Page-Check Intervals Properties

The `pageCheckSeconds` attribute determines the interval at which a server checks to see if JSP files in a Web application have changed and need recompiling. Each Web application defines this property separately in its `WEB-INF\weblogic.xml` file. For example, the e-commerce sample Web application defines this property in the following file:
`PORTAL_HOME\applications\wlcsApp\wlcs\WEB-INF\weblogic.xml`

The following excerpt from the e-commerce sample Web application shows the `pageCheckSeconds` attribute with the default value in boldface text:

```
<jsp-param>
     <param-name>pageCheckSeconds</param-name>
     <param-value>1</param-value>
</jsp-param>
```

Note: The page-check interval does not determine the frequency with which a 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 Chapter 4, "Using Caches."

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 Portal performance with the interval set to `-1` (which specifies that the server never checks for modifications).

2. Test the performance with the interval set to various numbers of seconds. For example, set the interval to `600` seconds (10 minutes) and test the performance. Then set the interval to `900` seconds and test the performance.

3. Choose an interval that provides the best performance while checking for modifications to JSP files and servlet classes at a satisfactory rate.

CHAPTER

6 Additional Tuning Recommendations

This chapter recommends configurations for miscellaneous services and components:

- Adjust Database Connections Available at Startup
- Increase the Size of the Display-Count Buffer
- Display Metadata, Sort and Query Explicit Metadata
- Use LDAP for Authentication Only
- Use the HotSpot Virtual Machine

For information on tuning the Behavior Tracking service, refer to "Persisting Behavioral Tracking Data" in the *Guide to Events and Behavior Tracking*.

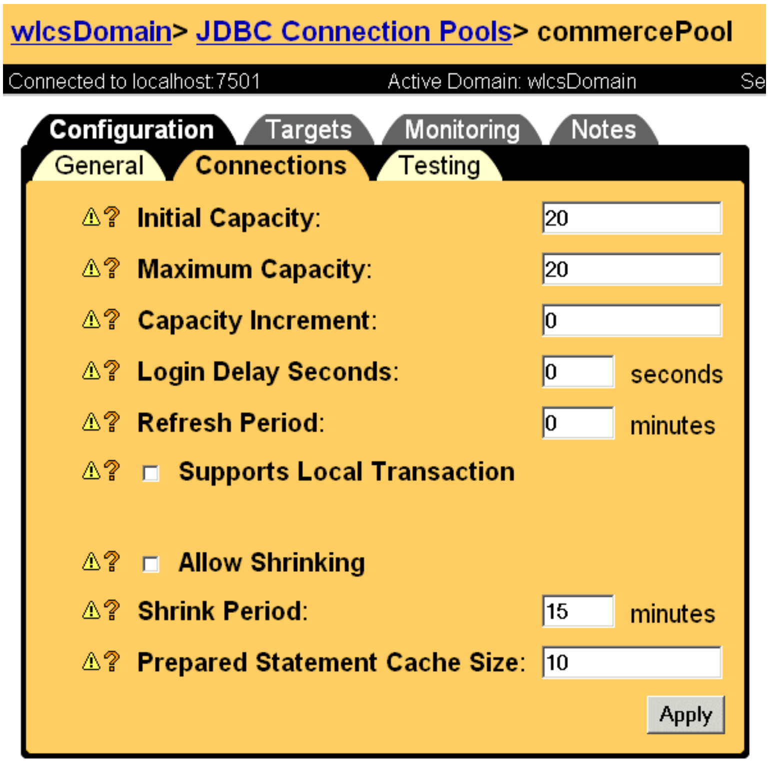
All of the recommendations in this document are in addition to the recommendations in the *BEA WebLogic Server Performance and Tuning* guide.

Adjust Database Connections Available at Startup

To optimize the database pool performance for your production Web site, do the following:

1. Start the WebLogic Server Administration Console for your domain.
2. In the WebLogic Server Administration Console, under JDBC, click Connection Pools.
3. On the JDBC Connection Pools page, in the Name column, click commercePool.
4. On the commercePool page, click the Connections tab and do the following (see Figure 6-1):
   a. Increase the value in Initial Capacity to match the value in Maximum Capacity.
   b. Change Login Delay Seconds to `0`.
   c. Clear the Allow Shrinking check box.
   d. Click Apply.
5. Click the Testing tab and clear the Test Reserve Connections check box.
6. Click Apply.
7. Restart the server.

Figure 6-1 Change Values on the commercePool Tab



wlcsDomain> JDBC Connection Pools> commercePool

Connected to localhost:7501 Active Domain: wlcsDomain Se

Initial Capacity:	20	
Maximum Capacity:	20	
Capacity Increment:	0	
Login Delay Seconds:	0	seconds
Refresh Period:	0	minutes
☐ Supports Local Transaction		
☐ Allow Shrinking		
Shrink Period:	15	minutes
Prepared Statement Cache Size:	10	

For More Information

For more information on database connection pools, refer to the WebLogic Server Administration Console online help.

Increase the Size of the Display-Count Buffer

The Campaign service uses display counts to determine whether a campaign has met its end goals. Each time an ad placeholder finds an ad to display as a result of a scenario action, the Campaign service updates the display count.

By default, the Campaign service does not update the display count in the database until an ad placeholder has found 10 ads to display as a result of one or more scenario actions. If the server crashes before the Campaign service flushes this display-count buffer to the database, you can lose display-count updates, up to the number of display counts that are in the buffer.

You can use the WebLogic Server Administration Console to determine the number of display counts that are stored in memory before the Campaign service updates the database:

1. Start the WebLogic Server Administration Console by entering the following URL in a Web browser:

   `http://`*`hostname`*`:`*`port-number/`*`console`

   For example, you started a server on a host named bonnie and it uses port 7001 as a listen-on port. Enter the following URL:

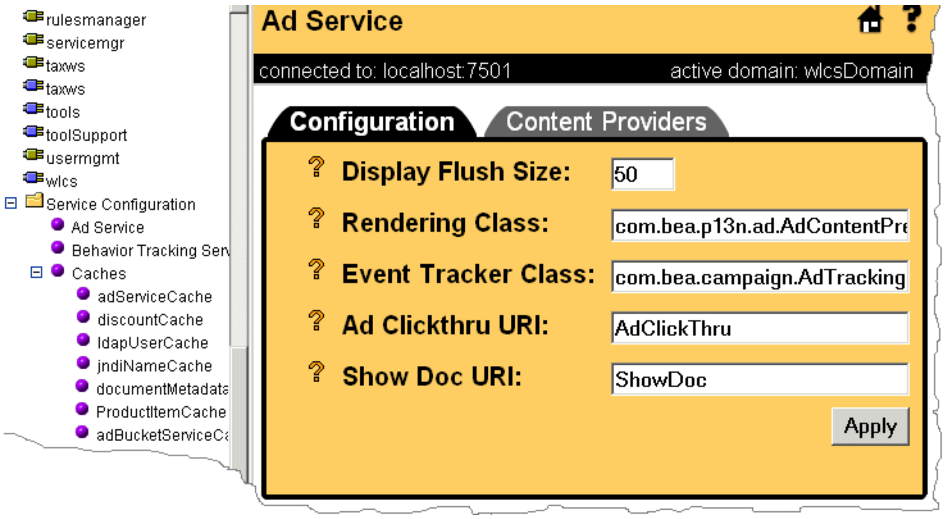
   `http://bonnie:7001/console`

2. The WebLogic Server Administration Console prompts you to log in with a user account that has administrator privileges.

3. After you log in, in the left pane, click Deployments → Applications → *MyApplication* → Service Configuration → Ad Service.

4. In the Display Flush Size box, change the value. For sites with high traffic, increase this number to a range of 50 to 100. (See Figure 6-2.)

Figure 6-2 Adjust the Ad Count



connected to: localhost:7501 active domain: wlcsDomain

Display Flush Size: 50
Rendering Class: com.bea.p13n.ad.AdContentPre
Event Tracker Class: com.bea.campaign.AdTracking
Ad Clickthru URI: AdClickThru
Show Doc URI: ShowDoc

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 about content management, see "Creating and Managing Content" in the *Guide to Building Personalized Applications*.

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 about changing LDAP settings, see "Using Other Realms" under "Creating and Managing Users" in the *Guide to Building Personalized Applications*.

Use the HotSpot Virtual Machine

Hot Spot enhances JDK 1.3 performance. It provides several implementations, which vary depending upon the operating system. HotSpot is an optimized VM with several variations.

For Windows:

WebLogic Portal supports the Client VM.

Note: If WebLogic Portal is configured to use the default Cloudscape database, the Classic VM is used. If WebLogic Portal is configured with any other database, the default is the Client VM using the -*hotspot* implementation.

To change this setting to the -*client* implementation of Client VM:

1. Navigate to `PORTAL_HOME\config\portalDomain`.
2. Open the `startPortal.bat` file for edit.

   Change:

   ```
   SET JAVA_VM=-hotspot
   ```

   To:

   ```
   SET JAVA_VM=-client
   ```

For Linux or Solaris:

WebLogic Portal is certified for both the Server VM and the Client VM, but the Server VM is the default setting.

To change the HotSpot VM:

1. Navigate to `PORTAL_HOME/bin/unix`.
2. Open the `set-environment.sh` file for edit.

For Linux change the following line under the LINUX|Linux heading of the VM Options section (the following is an example entry):

```
JAVA_VM_OPTIONS=”-hotspot -Xms128m - Xmx128”
```

To:

```
JAVA_VM_OPTIONS="-ms160m -mx300m -XX:MaxNewSize=100m
-XX:NewSize=100m -XX:MaxPermSize=128m"
```

For HP-UX or AIX:

WebLogic Portal does not specify this setting, so it defaults to the Client VM. Both Client and Server VMs are certified for WebLogic Portal.

The default `startPortal` startup script activates the HotSpot VM that is appropriate for each platform type. This script is located at `PORTAL_HOME/config/portalDomain`.

Deactivating HotSpot

To deactivate HotSpot, do the following:

For Windows:

1. Open the `startPortal.bat` file in `PORTAL_HOME\config\portalDomain` for edit.
2. Change the value of the `JAVA_VM` variable to `-classic`.
3. Restart the server using a startup script that refers to the `set-environment` file that you modified.

For UNIX:

1. Open the `set-environment.sh` file in `PORTAL_HOME/bin/unix` for edit.
2. Change the value of the `JAVA_VM_OPTIONS` variable to `-classic`.
3. Restart the server using a startup script that refers to the `set-environment` file that you modified.

Note: For information on shutting down and starting the server, refer to "Starting and Shutting Down the Server" in the Deployment Guide.