



BEA WebLogic Server™

Programming WebLogic Management Services with JMX

Release 7.0
Revised: March 18, 2004

Copyright

Copyright © 2002 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, Jolt, Tuxedo, and WebLogic are registered trademarks of BEA Systems, Inc. BEA Builder, BEA Campaign Manager for WebLogic, BEA eLink, BEA Manager, BEA WebLogic Commerce Server, BEA WebLogic Enterprise, BEA WebLogic Enterprise Platform, BEA WebLogic Express, BEA WebLogic Integration, BEA WebLogic Personalization Server, BEA WebLogic Platform, BEA WebLogic Portal, BEA WebLogic Server, BEA WebLogic Workshop and How Business Becomes E-Business are trademarks of BEA Systems, Inc.

All other trademarks are the property of their respective companies.

Programming WebLogic Management Services with JMX

| Part Number | Document Revised | Software Version |
|--------------------|-------------------------|------------------------------------|
| N/A | March 14, 2004 | BEA WebLogic Server Version 7.0 |

Contents

About This Document

| | |
|--------------------------------|------|
| Audience..... | viii |
| e-docs Web Site..... | viii |
| How to Print the Document..... | viii |
| Contact Us!..... | ix |
| Documentation Conventions..... | ix |

1. Overview of WebLogic JMX Services

| | |
|--|------|
| WebLogic Server Managed Resources and MBeans..... | 1-2 |
| Basic Organization of a WebLogic Server Domain..... | 1-3 |
| MBeans for Configuring Managed Resources..... | 1-3 |
| Local Replicas of Configuration MBeans..... | 1-4 |
| The Life Cycle of Configuration MBeans..... | 1-5 |
| Replication of MBeans for Managed Server Independence..... | 1-8 |
| Documentation for Configuration MBean APIs..... | 1-8 |
| MBeans for Viewing the Runtime State of Managed Resources..... | 1-9 |
| Documentation for Runtime MBean APIs..... | 1-11 |
| Security MBeans..... | 1-12 |
| Non-WebLogic Server MBeans..... | 1-13 |
| MBean Servers and the MBeanHome Interface..... | 1-13 |
| Local MBeanHome and the Administration MBeanHome..... | 1-15 |
| Notifications and Monitoring..... | 1-17 |
| The Administration Console and the weblogic.Admin Utility..... | 1-17 |
| The Administration Console..... | 1-18 |
| The weblogic.Admin Utility..... | 1-18 |

2. Accessing WebLogic Server MBeans

| | |
|---|------|
| Accessing MBeans: Main Steps | 2-1 |
| Determining Which Interfaces to Use | 2-2 |
| Accessing an MBeanHome Interface | 2-4 |
| Using the Helper APIs to Retrieve an MBeanHome Interface..... | 2-4 |
| Example: Retrieving a Local MBeanHome Interface | 2-5 |
| Using JNDI to Retrieve an MBeanHome Interface..... | 2-6 |
| Example: Retrieving the Administration MBeanHome from an External Client..... | 2-8 |
| Example: Retrieving a Local MBeanHome from an Internal Client .. | 2-9 |
| Using the Type-Safe Interface to Access MBeans | 2-10 |
| Retrieving a List of All MBeans | 2-10 |
| Retrieving MBeans By Type and Selecting From the List..... | 2-12 |
| Walking the Hierarchy of Local Configuration and Runtime MBeans.... | 2-14 |
| Using the MBeanServer Interface to Access MBeans..... | 2-18 |
| Using WebLogicObjectName for WebLogic Server MBeans..... | 2-20 |
| Using weblogic.Admin to Find the WebLogicObjectName | 2-24 |

3. Accessing and Changing Configuration Information

| | |
|--|-----|
| Example: Using weblogic.Admin to View the Message Level for Standard Out.. | 3-2 |
| Example: Configuring the Message Level for Standard Out..... | 3-3 |

4. Accessing Runtime Information

| | |
|--|------|
| Example: Determining the Active Domain and Servers..... | 4-2 |
| Using weblogic.Admin to Determine Active Domains and Servers..... | 4-4 |
| Example: Viewing and Changing the Runtime State of a WebLogic Server Instance..... | 4-5 |
| Using a Local MBeanHome and getRuntimeMBean() | 4-5 |
| Using the Administration MBeanHome and getMBeansByType() | 4-8 |
| Using the Administration MBeanHome and getMBean()..... | 4-10 |
| Using the MBeanServer Interface | 4-12 |
| Example: Viewing Runtime Information About Clusters | 4-14 |
| Viewing Runtime Information for EJBs..... | 4-16 |
| Example: Retrieving Runtime Information for All Stateful and Stateless EJBs | 4-20 |

5. Using WebLogic Server MBean Notifications and Monitors

| | |
|--|------|
| How Notifications are Broadcast and Received | 5-2 |
| Monitoring Changes in MBeans..... | 5-3 |
| Best Practices: Listening Directly Compared to Monitoring | 5-6 |
| Best Practices: Commonly Monitored Attributes..... | 5-6 |
| Listening for Notifications from WebLogic Server MBeans: Main Steps..... | 5-9 |
| WebLogic Server Notification Types | 5-10 |
| Creating a Notification Listener..... | 5-11 |
| Creating a Notification Filter | 5-14 |
| Adding Filter Classes to the Server Classpath..... | 5-15 |
| Registering a Notification Listener and Filter..... | 5-16 |
| Listening for Configuration Auditing Messages: Main Steps..... | 5-19 |
| Notification Listener for Configuration Auditing Messages | 5-20 |
| Notification Filter for Configuration Auditing Messages..... | 5-21 |
| Registration Class for Configuration Auditing Messages..... | 5-22 |
| Using Monitor MBeans to Observe Changes: Main Steps..... | 5-24 |
| Choosing a Monitor MBean Type..... | 5-24 |
| Monitor Notification Types..... | 5-25 |
| Error Notification Types | 5-26 |
| Creating a Notification Listener for a Monitor MBean..... | 5-27 |
| Instantiating the Monitor and Listener..... | 5-28 |
| Example: Monitoring an MBean on a Single Server | 5-29 |
| Example: Monitoring Instances of an MBean on Multiple Servers.. | 5-32 |
| Configuring CounterMonitor Objects | 5-36 |
| Configuring GaugeMonitor Objects..... | 5-38 |
| Configuring StringMonitor Objects | 5-39 |
| Sample Monitoring Scenarios | 5-39 |
| JDBC Monitoring | 5-40 |



About This Document

This document describes how to use the BEA WebLogic Server™ management APIs to configure and monitor WebLogic Server domains, clusters, and server instances.

The document is organized as follows:

- [Chapter 1, “Overview of WebLogic JMX Services”](#) describes the WebLogic Server management interface and provides overviews of WebLogic Server MBeans, MBean home interfaces, and the distributed management architecture.
- [Chapter 2, “Accessing WebLogic Server MBeans,”](#) describes how to access interfaces for working with WebLogic Server MBeans.
- [Chapter 3, “Accessing and Changing Configuration Information,”](#) provides examples of retrieving and modifying the configuration of WebLogic Server resources.
- [Chapter 4, “Accessing Runtime Information,”](#) provides examples of retrieving and modifying runtime information about WebLogic Server domains and server instances.
- [Chapter 5, “Using WebLogic Server MBean Notifications and Monitors,”](#) describes how to observe and respond to changes in the values of WebLogic Server MBean attributes.

Note: The WebLogic Security Service provides MBeans and tools for generating additional MBeans that manage security on a WebLogic Server. These MBeans are called Security MBeans and their usage model is different from the one described in this document. For information on Security MBeans, refer to *Developing Security Providers for WebLogic Server*.

Audience

This document is written for independent software vendors (ISVs) and other developers who are interested in creating custom applications that use BEA WebLogic Server facilities to monitor and configure applications and server instances. It assumes that you are familiar with the BEA WebLogic Server platform and the Java programming language, but not necessarily with Java Management Extensions (JMX).

While the document describes how to access and use the Managed Beans (MBeans) that WebLogic Server provides, it does not describe how to create your own, additional MBeans. For information about creating and using MBeans in addition to the ones that WebLogic Server provides, refer to the JMX 1.0 specification, which you can download from <http://jcp.org/aboutJava/communityprocess/final/jsr003/index.html>

e-docs Web Site

BEA product documentation is available on the BEA corporate Web site. From the BEA Home page, click on Product Documentation.

How to Print the Document

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

A PDF version of this document is available on the WebLogic 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 Server documentation Home page, click Download Documentation, and select the document you want to print.

Adobe Acrobat Reader is available at no charge from the Adobe Web site at <http://www.adobe.com>.

Contact Us!

Your feedback on BEA 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 documentation.

In your e-mail message, please indicate the software name and version you are using, as well as the title and document date of your documentation. If you have any questions about this version of BEA WebLogic Server, or if you have problems installing and running BEA WebLogic Server, contact BEA Customer Support through BEA WebSupport at <http://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 | Usage |
|-------------------|--------------------------------|
| Ctrl+Tab | Keys you press simultaneously. |
| <i>italics</i> | Emphasis and book titles. |

| Convention | Usage |
|---|--|
| monospace text | Code samples, commands and their options, Java classes, data types, directories, and file names and their extensions. Monospace text also indicates text that the user is told to enter from the keyboard. <i>Examples:</i> import java.util.Enumeration; chmod u+w * config/examples/applications .java config.xml float |
| <i>monospace</i> <i>italic</i> text | Placeholders. <i>Example:</i> String <i>CustomerName</i> ; |
| UPPERCASE MONOSPACE TEXT | Device names, environment variables, and logical operators. <i>Examples:</i> LPT1 BEA_HOME OR |
| { } | A set of choices in a syntax line. |
| [] | Optional items in a syntax line. <i>Example:</i> java utils.MulticastTest -n <i>name</i> -a <i>address</i> [-p <i>portnumber</i>] [-t <i>timeout</i>] [-s <i>send</i>] |
| | Separates mutually exclusive choices in a syntax line. <i>Example:</i> java weblogic.deploy [list deploy undeploy update] password {application} {source} |
| ... | Indicates one of the following in a command line: <ul style="list-style-type: none"> ■ An argument can be repeated several times in the command line. ■ The statement omits additional optional arguments. ■ You can enter additional parameters, values, or other information |

| Convention | Usage |
|-------------------|--------------|
|-------------------|--------------|

- | | |
|---|--|
| . | Indicates the omission of items from a code example or from a syntax line. |
| . | |
| . | |
-



1 Overview of WebLogic JMX Services

WebLogic Server implements the Sun Microsystems, Inc. Java Management Extensions (JMX) 1.0 specification to provide open and extensible management services. WebLogic Server adds its own set of convenience methods and other extensions to facilitate working in the WebLogic Server distributed environment.

All WebLogic Server resources are managed through these JMX-based services, and third-party services and applications that run within WebLogic Server can be managed through them as well. You can build your own management utilities that use these JMX services to manage WebLogic Server resources and applications.

The following sections provide an overview of the WebLogic Server JMX services:

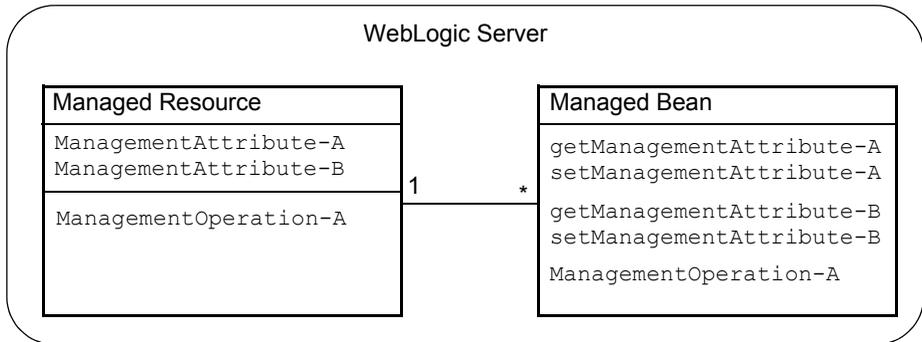
- “WebLogic Server Managed Resources and MBeans” on page 1-2
- “MBean Servers and the MBeanHome Interface” on page 1-13
- “Notifications and Monitoring” on page 1-17
- “The Administration Console and the `weblogic.Admin` Utility” on page 1-17

To view the JMX 1.0 specification, download it from <http://jcp.org/aboutJava/communityprocess/final/jsr003/index.html>. The API documentation is included in the archive that you download.

WebLogic Server Managed Resources and MBeans

Subsystems within WebLogic Server (such as JMS Provider and JDBC Container) and the items that they control (such as JMS servers and JDBC connection pools) are called **WebLogic Server managed resources**. Each managed resource includes a set of attributes that can be configured and monitored for management purposes. For example, each JDBC connection pool includes attributes that define its name, the name of its driver, its initial capacity, and its cache size. Some managed resources provide additional methods (operations) that can be used for management purposes. The WebLogic JMX services expose these management attributes and operations through one or more managed beans (MBeans). An **MBean** is a concrete Java class that is developed per JMX specifications. It can provide getter and setter operations for each management attribute within a managed resource along with additional management operations that the resource makes available. (See [Figure 1-1](#).)

Figure 1-1 Managed Resources and Managed Beans



WebLogic Server MBeans that expose attributes and operations for configuring a managed resource are called **Configuration MBeans** while MBeans that provide information about the runtime state of a managed resource are called **Runtime MBeans**. The functions of configuring resources and viewing data about the runtime state of resources in a WebLogic Server domain are different enough that Configuration MBeans and Runtime MBeans are distributed and maintained differently.

The following sections describe how WebLogic Server distributes and maintains MBeans:

- [“Basic Organization of a WebLogic Server Domain”](#) on page 1-3
- [“MBeans for Configuring Managed Resources”](#) on page 1-3
- [“MBeans for Viewing the Runtime State of Managed Resources”](#) on page 1-9
- [“Security MBeans”](#) on page 1-12
- [“Non-WebLogic Server MBeans”](#) on page 1-13

Basic Organization of a WebLogic Server Domain

A WebLogic Server administration **domain** is a logically related group of WebLogic Server resources. Domains include a special WebLogic Server instance called the **Administration Server**, which is the central point from which you configure and manage all resources in the domain. Usually, you configure a domain to include additional WebLogic Server instances called **Managed Servers**. You deploy applications, EJBs, and other resources developed onto the Managed Servers and use the Administration Server for configuration and management purposes only.

Using multiple Managed Servers enables you to balance loads and provide failover protection for critical applications, while using single Administration Server simplifies the management of the Managed Server instances. For more information about domains, refer to ["Overview of WebLogic System Administration"](#) in the *WebLogic Server Administration Guide*.

MBeans for Configuring Managed Resources

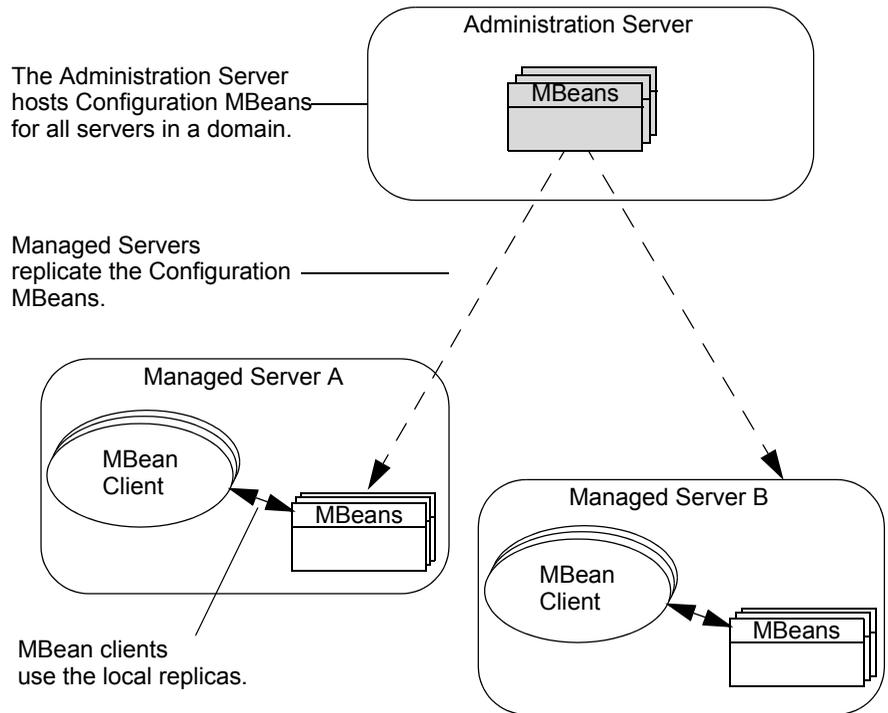
To support the WebLogic Server model of centralizing management responsibilities on the Administration Server, the Administration Server hosts Configuration MBeans for all managed resources on all server instances in the domain. In addition, the Administration Server saves changes to configuration data so that it is available when you shut down and restart a server instance.

To change the configuration of a WebLogic Server resource, you modify the values in the Configuration MBeans on the Administration Server.

Local Replicas of Configuration MBeans

To enhance performance and to support some clustering features, each Managed Server creates local replicas of all Configuration MBeans in a domain. WebLogic Server subsystems and applications that interact with MBeans use the replicas on the local server instead of initiating remote calls to the Administration Server. (See [Figure 1-2](#).)

Figure 1-2 MBean Replication



The Configuration MBeans on the Administration Server are called **Administration MBeans**, and the replicas on the Managed Servers are called **Local Configuration MBeans**.

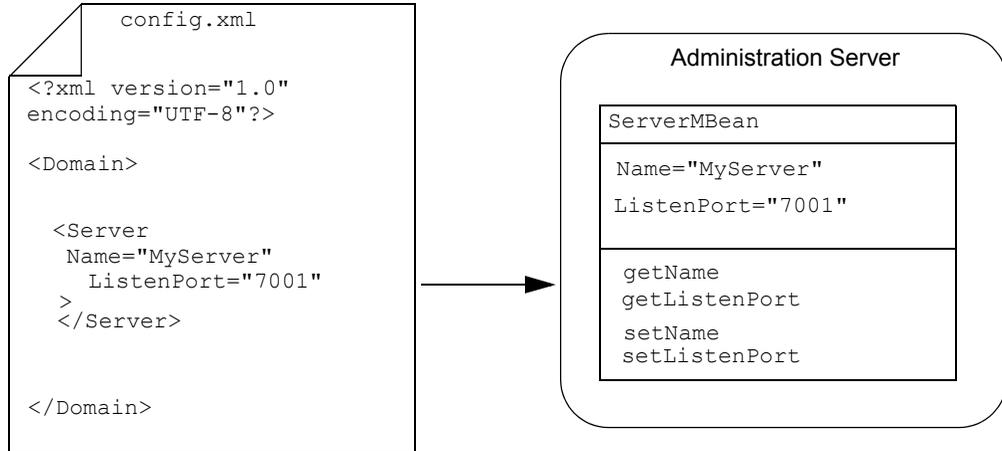
Note: In addition to hosting Administration MBeans, the Administration Server hosts the Local Configuration MBeans that are used by its own subsystems and by any applications that are deployed on the Administration Server.

The Life Cycle of Configuration MBeans

This section describes how Administration MBeans and Local Configuration MBeans are initialized, how changes to configuration data are propagated throughout the WebLogic Server system, and how attribute values can be changed so that they are available when you restart server instances:

1. The life cycle of a Configuration MBean begins when you start the Administration Server. During its startup cycle, the Administration Server initializes all the Administration MBeans for the domain with data from the domain's `config.xml` file. (See [Figure 1-3](#).)

Figure 1-3 Initializing Configuration MBeans



The Administration Server reads data from the `config.xml` file only during its startup cycle.

2. When a Managed Server starts, it contacts the Administration Server for its configuration data. By default, it creates replicas of the Administration MBeans that configure resources in the domain. However, you can use arguments in the server's startup command to override values of the Administration MBeans.

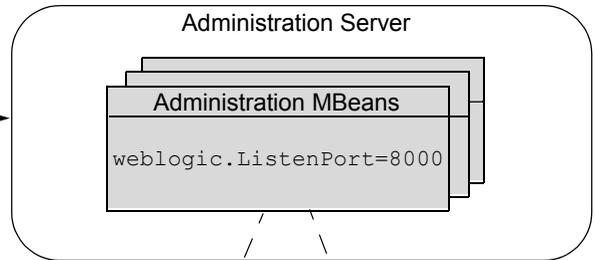
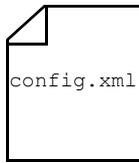
For example, for Managed Server A, the `config.xml` file states that its listen port is 8000. When you use the `weblogic.Server` command to start Managed Server A, you include the `-Dweblogic.ListenPort=7501` startup option to change the listen port for the current server session. The Managed Server creates a replica of the Administration MBeans, but substitutes 7501 as the value of its

1 Overview of WebLogic JMX Services

listen port. When you restart Managed Server A, it will revert to using the value from the `config.xml` file, 8000. (See [Figure 1-4](#).)

Figure 1-4 Overriding Administration MBean Values

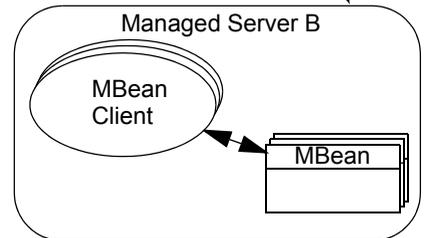
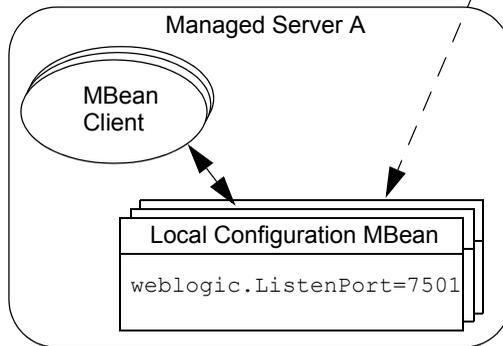
1. At startup, the Administration Server initializes Administration MBeans with data from the `config.xml` file.



2. At startup, Managed Servers replicate the Administration MBeans.

Startup options override the values from the Administration MBeans.

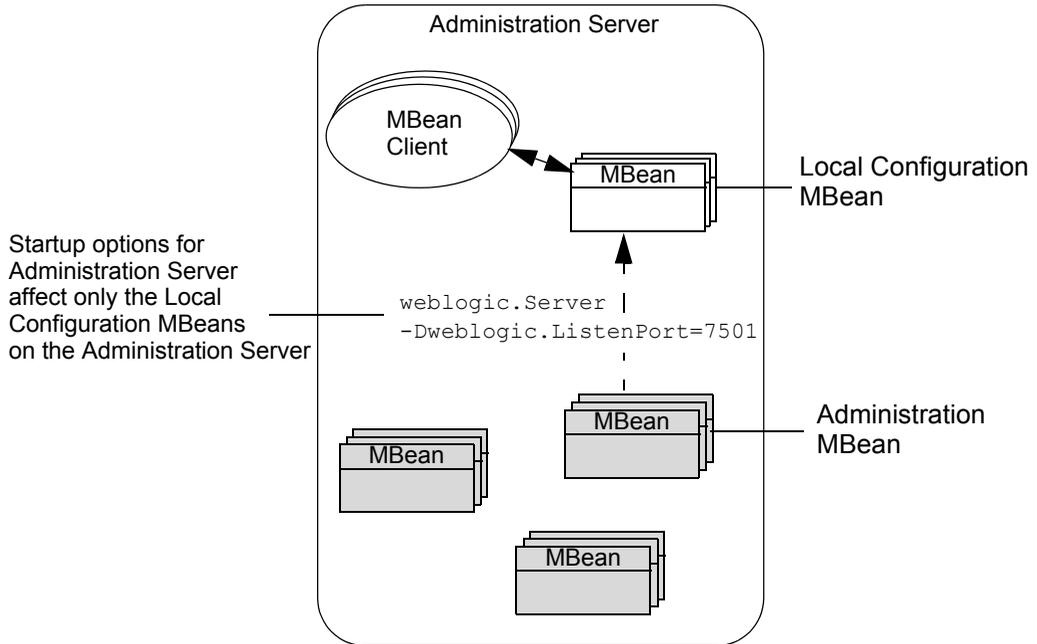
```
weblogic.Server  
-Dweblogic.ListenPort=7501
```



When you start an Administration Server, any startup command arguments that you use to override the values in `config.xml` affect only the values of the Local Configuration MBeans on the Administration Server. The command arguments

do not affect the values of the Administration MBeans and therefore do not affect subsequent server sessions. (See [Figure 1-5](#).)

Figure 1-5 Overriding Values on the Administration Server



3. If you change a value in an Administration MBean, and if the corresponding Managed Server is running, the Administration Server propagates the change to the Local Configuration MBean. Depending on the attribute, the underlying resource might not be able to accept the new value until it restarts. The WebLogic Server Javadoc indicates whether a managed resource can accept new values for an attribute during the current session. Even if a managed resource can accept new values, depending on the frequency with which the resource checks for configuration changes, the resource might not use the updated value immediately.

Note: BEA recommends that you change only the values of Administration MBean attributes. Do not change attribute values in Local Configuration MBeans. When a Managed Server replicates the data of other Managed Servers, it uses the values that are stored in Administration MBeans. Communication problems can occur if the values in Administration MBeans and Local Configuration MBeans differ.

1 Overview of WebLogic JMX Services

4. Periodically, the Administration Server determines whether Administration MBeans have been changed and writes any changes back to `config.xml`. Changes also are written to `config.xml` when the Administration Server shuts down or when MBean attributes are modified by a WebLogic Server utility such as the Administration Console or `weblogic.Admin`.
5. Local Configuration MBeans are destroyed when you shut down Managed Servers. Administration MBeans are destroyed when you shut down the Administration Server.

Replication of MBeans for Managed Server Independence

Managed Server Independence (MSI) is a feature that enables a Managed Server to start if the Administration Server is unavailable. If a Managed Server is configured for MSI, in addition to its Local Configuration MBeans, it also contains a copy of all Administration MBeans for the domain.

Do not interact with these Administration MBeans on a Managed Server. They reflect the last known configuration for the domain and are used only for starting the Managed Server in MSI mode. Modifying an Administration MBean on a Managed Server can cause the Managed Server's configuration to be inconsistent with the Administration Server, which will lead to unpredictable results. In addition, Managed Servers are not aware of the Administration MBeans on other Managed Servers.

For more information on MSI, refer to "[Starting a Managed Server When the Administration Server Is Not Accessible](#)" in the *Creating and Configuring WebLogic Server Domains* guide.

Documentation for Configuration MBean APIs

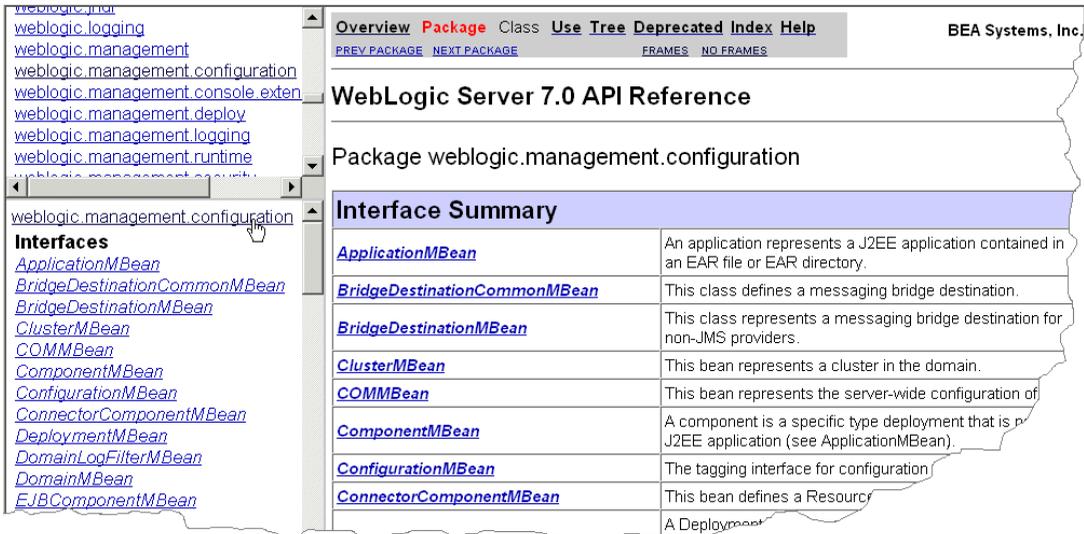
To view the documentation for Configuration MBeans:

1. Open the [WebLogic Server Javadoc](#).
2. In the top left pane of the Web browser, click `weblogic.management.configuration`.

The lower left pane displays links for the package.
3. In the lower left pane, click `weblogic.management.configuration` again.

The right pane displays the package summary. (See [Figure 1-6](#).)

Figure 1-6 Javadoc for the configuration Package



4. Click on an interface name to view its API documentation.

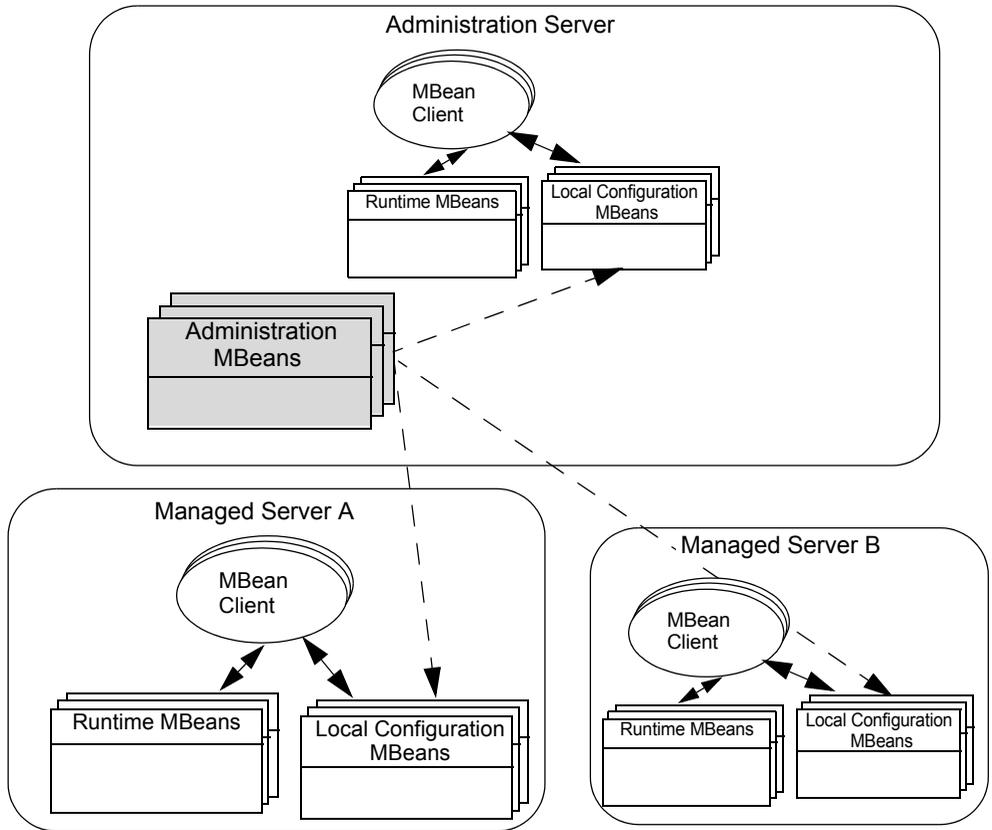
MBeans for Viewing the Runtime State of Managed Resources

WebLogic Server managed resources provide performance metrics and other information about their runtime state through one or more Runtime MBeans. Runtime MBeans are not replicated like Configuration MBeans, and they exist only on the same server instance as their underlying managed resources.

Because Runtime MBeans contain only transient data, they do not save their data in the `config.xml` file. When you shut down a server instance, all runtime statistics and metrics from the Runtime MBeans are destroyed.

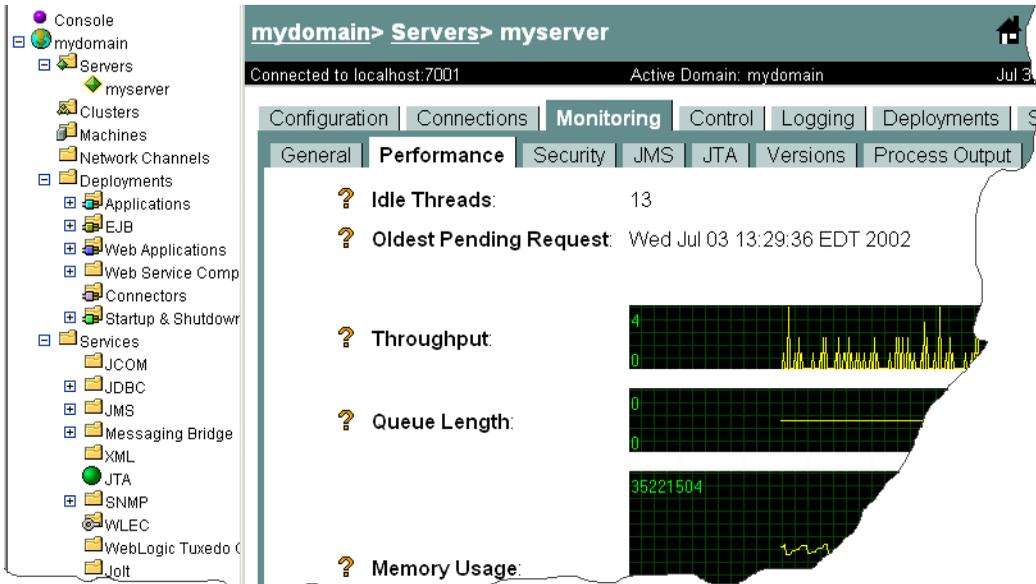
The following figure (Figure 1-7) illustrates how Runtime MBeans, Administration MBeans, and Local Configuration MBeans are distributed throughout a domain.

Figure 1-7 Distribution of MBeans



You can use the Administration Console, the `weblogic.Admin` utility, or MBean APIs to view the values. (See [Figure 1-8](#).)

Figure 1-8 Viewing Runtime Metrics from the Administration Console



You can also use these interfaces to change some runtime values. For example, the `weblogic.management.runtime.DeployerRuntimeMBean` activates and deactivates a deployed module by changing its runtime state.

Documentation for Runtime MBean APIs

To view the documentation for Runtime MBeans:

1. Open the [WebLogic Server Javadoc](#).
2. In the top left pane of the Web browser, click `weblogic.management.runtime`.
The lower left pane displays links for the package.
3. In the lower left pane, click `weblogic.management.runtime` again.
The right pane displays the package summary. (See [Figure 1-9](#).)

Figure 1-9 Javadoc for the runtime Package

The screenshot shows the Javadoc for the runtime package. The left pane shows a tree view with 'weblogic.management.runtime' selected. The main pane shows the 'Interface Summary' table:

| Interface Summary | |
|---|--|
| ApplicationRuntimeMBean | |
| CacheMonitorRuntimeMBean | |
| ClusterRuntimeMBean | This class is used for monitoring a server's view of the members of a Weblogic cluster within a Weblogic domain. |
| ConnectorConnectionPoolRuntimeMBean | This class is used for monitoring a Weblogic Connector Connection Pool |
| ConnectorConnectionRuntimeMBean | This class is used for monitoring individual Weblogic Connector connections |
| ConnectorServiceRuntimeMBean | This class is used for monitoring the Weblogic Connector Service |
| DeployerRuntimeMBean | This MBean is the user API for initiating deployment requests and exists only on an Admin Server |
| DeploymentTaskRuntimeMBean | Base interface for deployment tasks |

4. Click on an interface name to view its API documentation.

Security MBeans

The WebLogic Security Service provides MBeans and tools for generating additional MBeans that manage security on a WebLogic Server. These MBeans are called Security MBeans and their usage model is different from the one described in this document. For information on Security MBeans, refer to the *Developing Security Services for WebLogic Server* guide.

Non-WebLogic Server MBeans

WebLogic Server provides hundreds of MBeans, many of which are used to configure and monitor EJBs, Web applications, and other deployable J2EE modules. If you want to use additional MBeans to configure your applications or services, you can create your own MBeans.

Any MBeans that you create can take advantage of the full set of JMX 1.0 features, as defined by the JMX specification (which you can download from <http://jcp.org/aboutJava/communityprocess/final/jsr003/index.html>).

However, only MBeans that are provided by WebLogic Server can use the WebLogic Server extensions to JMX. For example, any MBeans that you create for your applications cannot save data in the `config.xml` file and they cannot use the type-safe interface as described in the next section, “[MBean Servers and the MBeanHome Interface](#).”

MBean Servers and the MBeanHome Interface

Within a WebLogic Server instance, the actual work of registering and providing access to MBeans is delegated to an MBean Server subsystem. The MBean Server on a Managed Server registers and provides access only to the Local Configuration MBeans and Runtime MBeans on the current Managed Server. The MBean Server on an Administration Server registers and provides access to the domain’s Administration MBeans as well as the Local Configuration MBeans and Runtime MBeans on the Administration Server.

Note: On a Managed Server that is configured for Managed Server Independence (MSI), the MBean Server also registers the Administration MBean replicas that the server uses to start if the Administration Server is not available. Do not interact with these Administration MBean replicas. For more information, refer to “[Replication of MBeans for Managed Server Independence](#)” on page 1-8.

1 Overview of WebLogic JMX Services

To access the MBean Server subsystem, you use the `weblogic.management.MBeanHome` interface. From `MBeanHome`, you can use any of the following interfaces to interact with the MBean Server and its MBeans (see [Figure 1-10](#)):

- `javax.management.MBeanServer`, which is the standard JMX interface for interacting with MBeans. You can use this interface to look up MBeans that are registered in an MBean Server, determine the set of operations available for an MBean, and determine the type of data that each operation returns. If you invoke MBean operations through the `MBeanServer` interface, you must use standard JMX methods. For example:

- `MBeanHome.getMBeanServer().getAttribute(MBeanObjectName, attributeName)`
- `MBeanHome.getMBeanServer().setAttribute(MBeanObjectName, attributeName)`
- `MBeanHome.getMBeanServer().invoke(MBeanObjectName, operationName, params, signature)`

For a complete list of `MBeanServer` APIs, refer to view the JMX 1.0 API documentation, which you can download from <http://jcp.org/aboutJava/communityprocess/final/jsr003/index.html>. The archive that you download includes the API documentation.

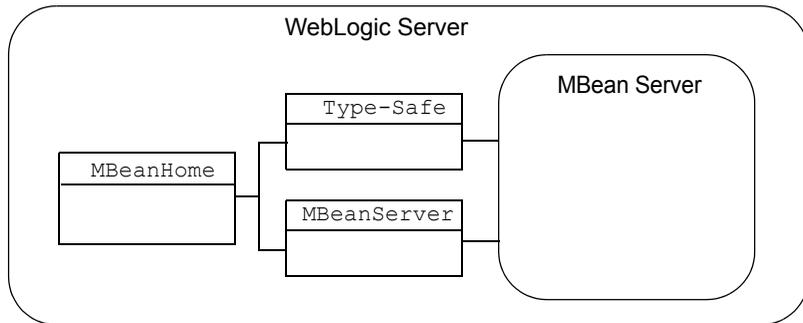
The `MBeanServer` interface is your only option for interacting with MBeans that you have created and registered (non-WebLogic MBeans).

- `weblogic.management.RemoteMBeanServer`, which extends the `javax.management.MBeanServer` and `java.rmi.Remote` interfaces. Use the `RemoteMBeanServer` interface if you want to use standard JMX techniques to access WebLogic Server MBeans from remote JVMs or if you want to interact with non-WebLogic MBeans from a remote JVM.
- A WebLogic Server type-safe interface that makes it appear as though you can invoke an MBean's methods directly. You can use this interface to look up MBeans that are registered in an MBean Server and invoke `get`, `set`, and other operations on the MBean. For example:

```
wlMBean = MBeanHome.getMBean(WebLogicObjectName)
wlMBean.getAttribute
wlMBean.setAttribute
wlMBean.operationName
```

The type-safe interface extends the `java.rmi.Remote` interface, so you can use it to access WebLogic Server MBeans from remote JVMs.

Figure 1-10 MBeans Servers and Their Interfaces



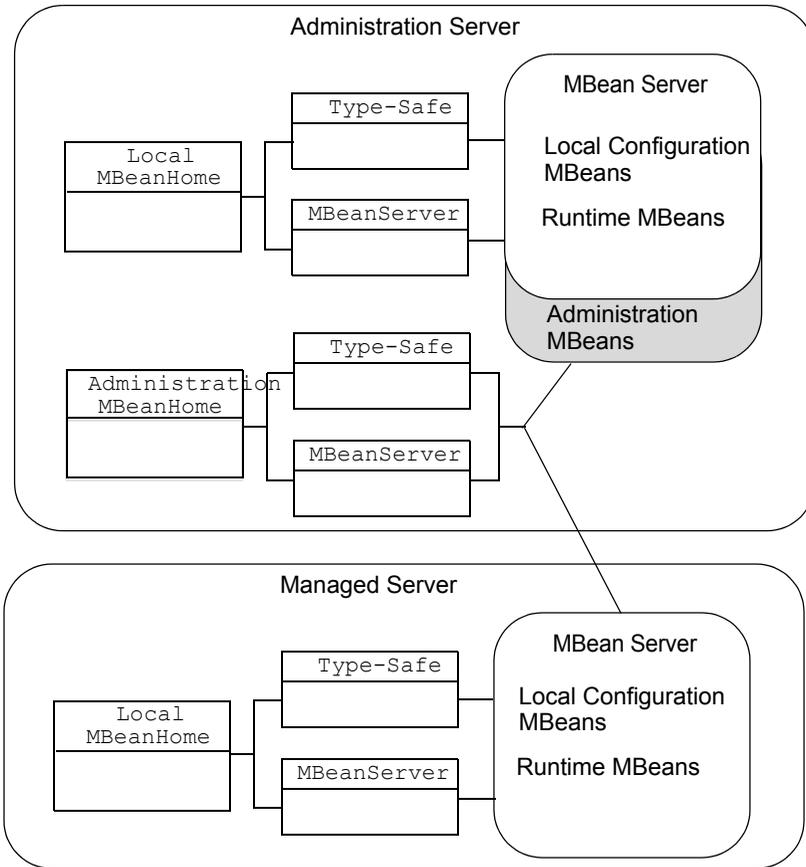
Local MBeanHome and the Administration MBeanHome

All instances of WebLogic Server provide a **local MBeanHome** interface through which you can access the MBeans that are hosted in the server instance's MBean Server.

For Managed Servers and Administration Servers, the local `MBeanHome` interface provides access to the Runtime MBeans for the current server only and to all Local Configuration MBeans in the domain.

The Administration Server provides an additional instance of the `MBeanHome` interface. This **Administration MBeanHome** provides access to Administration MBeans along with all other MBeans on all server instances in the domain. The Administration `MBeanHome` uses RMI to contact MBeans on Managed Servers, which uses more network resources and might take longer than using a local `MBeanServer` or `MBeanHome` interface. (See [Figure 1-11](#).)

Figure 1-11 Local and Administration MBeanHome Interfaces



The local `MBeanHome` and the Administration `MBeanHome` are two instances of the same interface class, so the APIs for the two types of `MBeanHome` differ only in the name of the `MBeanHome` instance and in the set of MBeans that you can access.

Notifications and Monitoring

Depending on your management needs, you can use MBean APIs to view MBean attributes only upon request, or you can use the WebLogic Server notification and monitoring facilities, which automatically broadcast reports (JMX notifications) when MBean attributes change.

To use these facilities:

- Create a JMX listener, which listens for and reports all attribute changes within an MBean that you specify. For example, you could use a listener with some additional logic to send an email to a System Administrator any time a user changes the configuration of a deployed component. For information about using listeners, refer to [Chapter 5, “Using WebLogic Server MBean Notifications and Monitors.”](#)
- Create a JMX monitor, which listens for and reports only the changes to specific MBean attributes that fall outside a set of parameters that you set. For example, you could use a monitor with some additional logic to send an email to a System Administrator when the number of open thread pools exceeds a specified limit. For more information, refer to [Chapter 5, “Using WebLogic Server MBean Notifications and Monitors.”](#)

The Administration Console and the `weblogic.Admin` Utility

The WebLogic Server Administration Console and the `weblogic.Admin` utility are examples of management utilities that use the WebLogic Server JMX services. You can use these interfaces to familiarize yourself with WebLogic Server management services before developing your JMX applications.

The Administration Console

The Administration Console is a Web application with servlets that invoke the WebLogic Server JMX APIs. Almost all of the values that the Administration Console presents are attributes of Administration MBeans and Runtime MBeans. Because the Administration Console does not read or write Local Configuration MBeans, it is possible that it reports a value that a server instance is not currently using. For example, if you use a `weblogic.Server` startup option to override the configured listen port, the Administration Console reports the value that is in the `config.xml` file, not the overriding value.

To determine which MBean attribute the Administration Console is presenting, click the question mark icon in the top banner. In the help window, click the Attributes link to see the MBean class and attribute that is associated with field on the Administration Console.

The caution icon (yellow triangle with an exclamation point) next to a field in the Administration Console indicates that an attribute is not dynamic. If you modify such an attribute, the underlying managed resource cannot use the new value until you restart the server.

If you modify a dynamic value from the Administration Console, the console updates the corresponding Administration MBean. For information on how this change is propagated to the Local Configuration MBean, refer to [“The Life Cycle of Configuration MBeans” on page 1-5](#).

The `weblogic.Admin` Utility

The `weblogic.Admin` utility provides several commands that create, get and set values for, invoke operations on, and delete instances of Administration and Configuration MBeans. It also provides commands to get values and invoke operations on Runtime MBeans. You could create shell scripts that use this utility instead of creating JMX applications to programmatically interact with the WebLogic Server management services, however, the performance of a JMX application is superior to a shell script that invokes command-line utilities.

You can also use the `weblogic.Admin` utility to verify object names of MBeans and to get and set attributes from a command line before committing to writing JMX code. Subsequent sections in this document provide examples of using the `weblogic.Admin` utility as part of your JMX development.

For more information, refer to "[MBean Management Command Reference](#)" in the *WebLogic Server Administration Guide*.

1 *Overview of WebLogic JMX Services*

2 Accessing WebLogic Server MBeans

All JMX tasks—viewing or changing MBean attributes, using notifications, and monitoring changes—use the same process to access MBeans.

The following sections describe how to access WebLogic Server MBeans:

- [“Accessing MBeans: Main Steps” on page 2-1](#)
- [“Determining Which Interfaces to Use” on page 2-2](#)
- [“Accessing an MBeanHome Interface” on page 2-4](#)
- [“Using the Type-Safe Interface to Access MBeans” on page 2-10](#)
- [“Using the MBeanServer Interface to Access MBeans” on page 2-18](#)
- [“Using WebLogicObjectName for WebLogic Server MBeans” on page 2-20](#)
- [“Using weblogic.Admin to Find the WebLogicObjectName” on page 2-24](#)

Accessing MBeans: Main Steps

The main steps for accessing MBeans in WebLogic Server are as follows:

1. Use a `weblogic.management.MBeanHome` interface to access the MBean Server. See [“Accessing an MBeanHome Interface” on page 2-4](#).

2. Use one of the following interfaces to retrieve, look up, and invoke operations on MBeans:
 - A type-safe interface that WebLogic Server provides. This interface, which is a WebLogic Server extension to JMX, can retrieve and invoke operations only on the MBeans that WebLogic Server provides. See [“Using the Type-Safe Interface to Access MBeans” on page 2-10](#).
 - The standard JMX `javax.management.MBeanServer` interface, which can retrieve and invoke operations on WebLogic Server MBeans or on MBeans that you create. See [“Using the MBeanServer Interface to Access MBeans” on page 2-18](#).
 - The `weblogic.management.RemoteMBeanServer` interface, which extends the `javax.management.MBeanServer` and `java.rmi.Remote` interfaces.

In most cases, you use these interfaces to retrieve a list of MBeans and then filter the list to retrieve and invoke operations on a specific MBean. However, if you know the `WebLogicObjectName` of an MBean, you can retrieve an MBean directly by name.

Determining Which Interfaces to Use

When accessing MBeans, you must make two choices about which interfaces you use:

- Whether to use the `MBeanHome` interface on a local server instance or the `Administration MBeanHome` interface to access the MBean Server. The `MBeanHome` interface that you choose determines the set of MBeans you can access.

The following table lists typical considerations for determining whether to use the local `MBeanHome` interface or the `Administration MBeanHome` interface.

Table 2-1 Deciding Between the Local or Administration MBeanHome

| If your application manages... | Retrieve this MBeanHome interface... |
|--|--|
| Local Configuration MBeans or Runtime MBeans | <p>Administration MBeanHome or local MBeanHome</p> <p>The Administration MBeanHome provides a single, convenient interface from which to access all MBeans on all server instances in a domain. When you use this interface, you typically retrieve MBeans from multiple server instances and then iterate through the list to find an MBean for a specific server instance.</p> <p>A local MBeanHome provides access to the Runtime MBeans for the current server only and to all Local Configuration MBeans in the domain. The interface uses fewer network hops to access MBeans because it requires your client to establish a direct connection to the server instance.</p> <p>When using a local MBeanHome, you typically retrieve one of several top-level MBeans and use them to walk the MBean hierarchy. See “Walking the Hierarchy of Local Configuration and Runtime MBeans” on page 14.</p> |
| Administration MBeans | Administration MBeanHome |

- Whether to use the WebLogic Server type-safe interface, the standard JMX `MBeanServer` interface, or the WebLogic `RemoteMBeanServer` interface to access and invoke operations on MBeans.

The following table lists typical considerations for determining whether to use the type-safe interface or the `MBeanServer` interface.

Table 2-2 Deciding Between the Type-Safe Interface or the MBeanServer Interface

| If your application... | Use this interface... |
|---|---|
| Interacts only with WebLogic Server MBeans. | The WebLogic Server type-safe interface |

Table 2-2 Deciding Between the Type-Safe Interface or the MBeanServer Interface

| If your application... | Use this interface... |
|--|---|
| Might need to run on J2EE platforms other than WebLogic Server | <code>MBeanServer</code> If your client accesses MBeans that are running in a separate JVM, use <code>RemoteMBeanServer</code> . Your client code will still be portable to other J2EE servers, although you cannot on other J2EE servers you must substitute <code>RemoteMBeanServer</code> with some other interface that extends the standard <code>MBeanServer</code> interface. |
| Interacts with non-WebLogic Server MBeans | <code>MBeanServer</code> If your client accesses MBeans that are running in a separate JVM, use <code>RemoteMBeanServer</code> . |

Accessing an MBeanHome Interface

The simplest process for retrieving a local `MBeanHome` interface or an Administration `MBeanHome` interface is to use the `WebLogic Server Helper` class. If you are more comfortable with a standard J2EE approach, you can use the Java Naming and Directory Interface (JNDI) to retrieve `MBeanHome`.

Using the Helper APIs to Retrieve an MBeanHome Interface

WebLogic Server provides the `weblogic.management.Helper` APIs to simplify the process of retrieving `MBeanHome` interfaces.

To use the `Helper` APIs, collect the following information:

- The username and password of a WebLogic Server user who has permission to invoke MBean operations. For more information, refer to “[Protecting System Administration Operations](#)” in *WebLogic Server Administration Guide*.

- If you are accessing a local `MBeanHome` interface, the name of the target server (as defined in the domain configuration) and the URL of the target server.
- If you are accessing the Administration `MBeanHome`, the URL of the Administration Server.

After you collect the information, use one of the following APIs:

- To retrieve a local `MBeanHome`:
`Helper.getMBeanHome(java.lang.String user, java.lang.String password, java.lang.String serverURL, java.lang.String serverName)`
- To retrieve the Administration `MBeanHome`:
`Helper.getAdminMBeanHome(java.lang.String user, java.lang.String password, java.lang.String adminServerURL)`

For more information about the `Helper` APIs, refer to the [WebLogic Server Javadoc](#).

Example: Retrieving a Local MBeanHome Interface

The following example ([Listing 2-1](#)) is a class that uses the `Helper` API to obtain the local `MBeanHome` interface for a server named `MS1`.

Listing 2-1 Retrieving a Local MBeanHome Interface

```
import weblogic.management.Helper;
import weblogic.management.MBeanHome;

public class UseHelper {
    public static void main(String[] args) {
        String url = "t3://localhost:7001";
        String username = "weblogic";
        String password = "weblogic";
        String msName = "MS1";
        MBeanHome localHome = null;

        try {
            localHome = (MBeanHome)Helper.getMBeanHome(username, password, url,
                msName);
            System.out.println("Local MBeanHome for" + localHome +
                " found using the Helper class");
        } catch (IllegalArgumentException iae) {
            System.out.println("Illegal Argument Exception: " + iae);
        }
    }
}
```

```
    }  
  }  
}
```

Using JNDI to Retrieve an MBeanHome Interface

While the `Helper` APIs provide a simple way to obtain an `MBeanHome` interface, you might be more familiar with the standard approach of using JNDI to retrieve the `MBeanHome`. From the JNDI tree of a Managed Server, you can access the server's local `MBeanHome` interface. From the JNDI tree of the Administration Server, you can access the Administration `MBeanHome` as well as the local `MBeanHome` interface for any server instance in the domain.

To use JNDI to retrieve an `MBeanHome` interface:

1. Construct a `weblogic.jndi.Environment` object and use `Environment` methods to configure the object:
 - a. Use the `setSecurityPrincipal` and `setSecurityCredentials` methods to specify user credentials.

WebLogic Server verifies that the user credentials you supply have been granted permission to carry out requests through the `MBeanHome` interface. For more information, refer to “[Protecting System Administration Operations](#)” in *WebLogic Server Administration Guide*.
 - b. If your application and the `MBeanHome` interface are in different JVMs, use the `Environment.setProviderUrl` method to specify the server instance that hosts the `MBeanHome` interface. The URL must specify the listen address of the server and the port on which the server listens for administrative requests.

If you want to retrieve the Administration `MBeanHome`, `setProviderUrl` must specify the Administration Server.
 - c. Use the `getInitialContext` method to initialize a `javax.naming.Context` object.

For example, the following lines of code set the initial context to a server instance that runs on a host computer named `WLServerHost` and uses the default domain-wide administration port to receive administrative requests:

```
Environment env = new Environment();
env.setProviderUrl("t3://WLServerHost:9002");
env.setSecurityPrincipal("weblogic");
env.setSecurityCredentials("weblogic");
Context ctx = env.getInitialContext();
```

For more information about `weblogic.jndi.Environment`, refer to the [WebLogic Server Javadoc](#).

2. Use `javax.naming.Context` methods to look up and retrieve the `MBeanHome` interface for the current context.

Use one of the following APIs, depending on whether you are retrieving a local `MBeanHome` interface or the Administration `MBeanHome`:

- To retrieve the local `MBeanHome` for the current context, use the following API:

```
javax.naming.Context.lookup(MBeanHome.LOCAL_JNDI_NAME)
```

- If the current context is an Administration Server, use the following API to retrieve the local `MBeanHome` of any server instance in the domain:

```
javax.naming.Context.lookup("weblogic.management.home.relevantServerName")
```

where `relevantServerName` is the name of a server as defined in the domain configuration.

- If the current context is an Administration Server, use the following API to retrieve the Administration `MBeanHome`:

```
javax.naming.Context.lookup(MBeanHome.ADMIN_JNDI_NAME)
```

The Administration `MBeanHome` interface provides access to all Local Configuration, Administration, and Runtime MBeans in the domain.

For more information about

```
javax.naming.Context.lookup(String name), refer to the Sun Javadoc.
```

The following sections are examples of retrieving `MBeanHome` interfaces:

- [Example: Retrieving the Administration MBeanHome from an External Client](#)
- [Example: Retrieving a Local MBeanHome from an Internal Client](#)

Example: Retrieving the Administration MBeanHome from an External Client

The following example ([Listing 2-2](#)) shows how an application running in a separate JVM looks up the Administration MBeanHome interface. In the example, `weblogic` is a user who has permission to view and modify MBean attributes. For information about permissions to view and modify MBeans, refer to "[Protecting System Administration Operations](#)" in *WebLogic Server Administration Guide*.

Listing 2-2 Retrieving the Administration MBeanHome from an External Client

```
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.AuthenticationException;
import javax.naming.CommunicationException;
import javax.naming.NamingException;
import weblogic.jndi.Environment;
import weblogic.management.MBeanHome;

public class RetrieveMBeanHome{

    public static void main(String[] args) {
        MBeanHome home = null;
        //domain variables
        String url = "t3://localhost:7001";
        String username = "weblogic";
        String password = "weblogic";

        //Setting an initial context.
        try {
            Environment env = new Environment();
            env.setProviderUrl(url);
            env.setSecurityPrincipal(username);
            env.setSecurityCredentials(password);
            Context ctx = env.getInitialContext();

            //Retrieving the Administration MBeanHome interface
            home = (MBeanHome) ctx.lookup(MBeanHome.ADMIN_JNDI_NAME);
            System.out.println("Got the Admin MBeanHome: " + home + " from the
                               Admin server");

        } catch (Exception e) {
            System.out.println("Exception caught: " + e);
        }
    }
}
```

Example: Retrieving a Local MBeanHome from an Internal Client

If your client application resides in the same JVM as the Administration Server (or the WebLogic Server instance you want to manage), the JNDI lookup for the `MBeanHome` is simpler. Listing 2-3 shows how a JMX application running in the same JVM as a WebLogic Server instance would look up the local `MBeanHome` for a server instance that listens at `t3://localhost:7001`.

Listing 2-3 Retrieving a Local MBeanHome from an Internal Client

```
import javax.naming.Context;
import javax.management.ObjectName;
import weblogic.management.MBeanHome;
import weblogic.management.WebLogicMBean;
import weblogic.management.WebLogicObjectName;
import weblogic.jndi.Environment;

public class serverInfo {
    public static void main(String[] args) {

        MBeanHome home = null;
        //domain variables
        String url = "t3://localhost:7001";
        String username = "weblogic";
        String password = "weblogic";

        try {
            Environment env = new Environment();
            env.setProviderUrl(url);
            env.setSecurityPrincipal(username);
            env.setSecurityCredentials(password);

            //Setting the initial context
            Context ctx = env.getInitialContext();

            //Retrieving the server-specific MBeanHome interface
            home = (MBeanHome)ctx.lookup(MBeanHome.LOCAL_JNDI_NAME);
            System.out.println("Got the Server-specific MBeanHome: " + home);
```

Using the Type-Safe Interface to Access MBeans

After you retrieve the `MBeanHome` interface, the easiest approach for accessing MBeans is to use methods in the `MBeanHome` interface that retrieve a type-safe interface for MBeans.

You can use this type-safe interface only with the MBeans that WebLogic Server provides. You cannot use this type-safe interface for MBeans that are based on `MBean` types that you create.

Retrieving a List of All MBeans

You can use the `MBeanHome.getAllMBeans` method to look up the object names of MBeans that are within the scope of the `MBeanHome` interface that you retrieve. For example, if you retrieve the `AdministrationMBeanHome`, using `getAllMBeans()` returns a list of all MBeans in the domain. If you retrieve a `LocalMBeanHome` interface, using `getAllMBeans()` returns a list of the Runtime MBeans for the current server only and of all Local Configuration MBeans in the domain.

The example class in [Listing 2-4](#):

1. Uses `JNDI` APIs to retrieve the `AdministrationMBeanHome` interface.
2. Uses the `MBeanHome.getAllMBeans` method to retrieve all MBeans in a domain.
3. Assigns the list of MBeans to a `Set` object and uses methods of the `Set` and `Iterator` interfaces to iterate through the list.
4. Uses the `WebLogicMBean.getObjectNames` method to retrieve the `WebLogicObjectName` of each MBean.
5. Uses the `WebLogicObjectName.getName` and `WebLogicObjectName.getType` methods to retrieve the `Name` and `Type` values of the `WebLogicObjectName`.

In the example, `weblogic` is a user who has permission to view and modify MBean attributes. For information about permissions to view and modify MBeans, refer to "[Protecting System Administration Operations](#)" in *WebLogic Server Administration Guide*.

Listing 2-4 Retrieving All MBeans in a Domain

```
import javax.naming.Context;
import java.util.Set;
import java.util.Iterator;
import weblogic.jndi.Environment;
import weblogic.management.MBeanHome;
import weblogic.management.WebLogicMBean;
import weblogic.management.WebLogicObjectName;

public class ListAllMBeans{
    public static void main(String args[]) {
        String url = "t3://localhost:7001";
        String username = "weblogic";
        String password = "weblogic";

        try {
            //Obtaining an MBeanHome Using JNDI
            Environment env = new Environment();
            env.setProviderUrl(url);
            env.setSecurityPrincipal(username);
            env.setSecurityCredentials(password);
            Context ctx = env.getInitialContext();
            MBeanHome home = (MBeanHome)ctx.lookup(MBeanHome.ADMIN_JNDI_NAME);

            Set allMBeans = home.getAllMBeans();
            System.out.println("Size: " + allMBeans.size());
            for (Iterator itr = allMBeans.iterator(); itr.hasNext(); ) {
                WebLogicMBean mbean = (WebLogicMBean)itr.next();
                WebLogicObjectName objectName = mbean.getObject_name();
                System.out.println(objectName.getName() + " is a(n) " +
                    mbean.getType());
            }
        }catch(Exception e){
            System.out.println(e);
        }
    }
}
```

For more information about the `MBeanHome.getAllMBeans` method, refer to the [WebLogic Server Javadoc](#).

Retrieving MBeans By Type and Selecting From the List

Instead of retrieving a list of all MBeans in the scope of `MBeanHome`, you can retrieve a list of MBeans that match a specific type. `Type` indicates the type of resource that the MBean manages and whether the MBean is an Administration, Local Configuration, or Runtime MBean. For more information about types of MBeans, refer to the next section, “[Using WebLogicObjectNames for WebLogic Server MBeans](#)” on page 2-20.

The example class in [Listing 2-5](#):

1. Uses JNDI to retrieve the Administration `MBeanHome` interface.
2. Uses the `MBeanHome.getMBeansByType` method to retrieve a list of all `ServerRuntime` MBeans in a domain.
3. Assigns the list of MBeans to a `Set` object and uses methods of the `Set` and `Iterator` interfaces to iterate through the list.
4. Uses the `ServerRuntime.getName` method to retrieve the name of each `ServerRuntime` MBean. The name of a `ServerRuntime` MBean corresponds to the name of a server instance.
5. When it finds the `ServerRuntime` MBean for a server named `Server1`, it prints a message to standard out.

In the example, `weblogic` is a user who has permission to view and modify MBean attributes. For information about permissions to view and modify MBeans, refer to “[Protecting System Administration Operations](#)” in *WebLogic Server Administration Guide*.

Listing 2-5 Selecting by Type from a List of MBeans

```
import java.util.Set;
import java.util.Iterator;
import java.rmi.RemoteException;
import javax.naming.Context;
import javax.management.ObjectName;
```

```
import weblogic.management.MBeanHome;
import weblogic.management.WebLogicMBean;
import weblogic.management.WebLogicObjectName;
import weblogic.management.configuration.ServerMBean;
import weblogic.management.runtime.ServerRuntimeMBean;
import weblogic.jndi.Environment;

public class serverRuntimeInfo {

    public static void main(String[] args) {

        MBeanHome home = null;

        //domain variables
        String url = "t3://localhost:7001";
        String serverName = "Server1";
        String username = "weblogic";
        String password = "weblogic";

        ServerRuntimeMBean serverRuntime = null;
        Set mbeanSet = null;
        Iterator mbeanIterator = null;

        //Using JNDI to retrieve the Administration MBeanHome
        //Setting the initial context
        try {
            Environment env = new Environment();
            env.setProviderUrl(url);
            env.setSecurityPrincipal(username);
            env.setSecurityCredentials(password);
            Context ctx = env.getInitialContext();

            //Getting the Administration MBeanHome
            home = (MBeanHome) ctx.lookup(MBeanHome.ADMIN_JNDI_NAME);
            System.out.println("Got the Admin MBeanHome: " + home );
        } catch (Exception e) {
            System.out.println("Exception caught: " + e);
        }

        //Using the getMBeansByType method to get all ServerRuntime MBeans
        //in the domain.
        try {
            mbeanSet = home.getMBeansByType("ServerRuntime");

            //Iterating through the results and comparing the server names
            //find the one we want.
            mbeanIterator = mbeanSet.iterator();
            while(mbeanIterator.hasNext()) {
                serverRuntime = (ServerRuntimeMBean)mbeanIterator.next();
                //Using serverRuntime.getName to find the ServerRuntime
                //MBean for Server1.
            }
        }
    }
}
```

```
        if (serverRuntime.getName().equals(serverName)) {
            System.out.println("Got the serverRuntimeMBean: " +
                serverRuntime + " for: " + serverName);
        }
    }
} catch (Exception e) {
    System.out.println("Exception caught: " + e);
}
}
```

For more information about the `MBeanHome.getMBeansByType` method, refer to the [WebLogic Server Javadoc](#).

Walking the Hierarchy of Local Configuration and Runtime MBeans

WebLogic Server MBeans exist within a hierarchy that reflects the resources with which they are associated. For example, each server instance can contain multiple execute queues, and WebLogic Server represents this relationship by making each `ExecuteQueueMBean` a child of a `ServerMBean`.

Walking the hierarchy of MBeans is the easiest way to retrieve Local Configuration and Runtime MBeans. If you want to retrieve Administration MBeans, or if you want to use the Administration `MBeanHome` to retrieve MBeans, BEA recommends that you retrieve MBeans by type and then filter the list. See “[Retrieving MBeans By Type and Selecting From the List](#)” on page 2-12.

The root of the Configuration MBean hierarchy is `DomainMBean`. Below this root are MBeans such as:

- `ClusterMBean`
- `ServerMBean`
- `ApplicationMBean`
- `RealmMBean`
- JDBC and JMS configuration MBeans

The root of the Runtime hierarchy is `ServerRuntimeMBean`. Just below this root are MBeans such as:

- `ClusterRuntimeMBean`
- `ApplicationRuntimeMBean`
- JDBC and JMS runtime MBeans

Parent MBeans usually provide methods for retrieving their children. For example, `ServerMBean.getExecuteQueues` returns all `ExecuteQueueMBeans` that have been configured for the server.

To walk the hierarchy of Local Configuration MBeans or Runtime MBeans:

1. From your JMX application, retrieve the local `MBeanHome` interface.
2. From the local `MBeanHome` interface, retrieve one of the top-level MBeans by invoking one of the following methods:

- `getConfigurationMBean (java.lang.String name, java.lang.String type)`

See the Javadoc for [MBeanHome.getConfigurationMBean](#).

- `getRuntimeMBean (java.lang.String name, java.lang.String type)`

See the Javadoc for [MBeanHome.getRuntimeMBean](#).

Use these methods to retrieve only MBeans that are immediately below `DomainMBean` or `ServerRuntimeMBean`. These methods do not return MBeans that are below the first level of the MBean hierarchy.

3. From the MBean that you retrieved, invoke methods to retrieve the MBean's children.

If a parent MBean does not provide methods to retrieve child MBeans, use `getMBeanByType()` and iterate over the results to find the MBean that matches your criteria. If you want to retrieve Local Configuration MBeans, be sure to append `Config` to the MBean type value. See “[Retrieving MBeans By Type and Selecting From the List](#)” on page 2-12.

Note: BEA recommends that you retrieve Local Configuration MBeans only to read values; do not **change** attribute values in Local Configuration MBeans. When the Managed Server replicates the data of other Managed Servers, it uses the

2 Accessing WebLogic Server MBeans

values that are stored in Administration MBeans. Communication problems can occur if the values in Administration MBeans and Local Configuration MBeans differ.

[Listing 2-6](#) is an example of retrieving all Local Configuration `ExecuteQueueMBeans` on a server instance named `ManagedServer1`.

Listing 2-6 Retrieving Local Configuration `ExecuteQueueMBeans`

```
import javax.naming.Context;
import javax.management.ObjectName;
import weblogic.management.MBeanHome;
import weblogic.management.WebLogicMBean;
import weblogic.management.WebLogicObjectName;
import weblogic.management.configuration.ConfigurationMBean;
import weblogic.management.configuration.ServerMBean;
import weblogic.management.configuration.ExecuteQueueMBean;

import weblogic.jndi.Environment;

public class serverConfigInfo {
    public static void main(String[] args) {
        MBeanHome home = null;
        ServerMBean servercfg = null;
        ExecuteQueueMBean[] xqueues = null;
        ExecuteQueueMBean xqueue = null;

        //domain variables
        String url = "t3://localhost:7001";
        String serverName = "ManagedServer1";
        String username = "weblogic";
        String password = "weblogic";

        try {
            Environment env = new Environment();
            env.setProviderUrl(url);
            env.setSecurityPrincipal(username);
            env.setSecurityCredentials(password);

            //Setting the initial context
            Context ctx = env.getInitialContext();

            //Retrieving the server-specific MBeanHome interface
            home = (MBeanHome) ctx.lookup(MBeanHome.LOCAL_JNDI_NAME);
            System.out.println("Got the Server-specific MBeanHome: " + home);
        }
    }
}
```

```
//Retrieving the Local Configuration ServerMBean
servercfg = (ServerMBean)home.getConfigurationMBean(serverName,
                                                    "ServerConfig");
System.out.println("Got the Server Config MBean: " + servercfg);

//Retrieving all ExecuteQueue MBeans that have been
//configured for the server instance
xqueues = servercfg.getExecuteQueues();

//Iterating through the results
for (int i=0; i < xqueues.length; i++){
    xqueue = xqueues[i];
    System.out.println("Execute queue name: " +
                      xqueue.DEFAULT_QUEUE_NAME);
    System.out.println("Thread count:" + xqueue.getThreadCount());
}
} catch (Exception e) {
    System.out.println("Exception caught: " + e);
}
}
```

If you want to create generic JMX code that you can run on any server instance to retrieve its Server Configuration MBean:

1. From the local `MBeanHome` interface, use the `getMBeansByType` method to retrieve the server's `ServerRuntimeMBean`:

```
serverRuntime = MBeanHome.getMBeansByType(ServerRuntime)
```

The local `MBeanHome` interface can access only the runtime MBeans that are specific to the current server instance, so `getMBeansByType(ServerRuntime)` returns only the `ServerRuntimeMBean` for the current server.

2. Use `ServerRuntimeMBean`'s `getName` method to retrieve the name of the server:

```
serverName = serverRuntime.getName()
```
3. Use the server name when invoking `MBeanHome.getConfigurationMBean`:

```
MBeanHome.getConfigurationMBean(serverName, "ServerConfig")
```

For more information, see [“Example: Determining the Active Domain and Servers” on page 4-2](#).

Using the MBeanServer Interface to Access MBeans

A standard JMX approach for interacting with MBeans is to use the `javax.management.MBeanServer` interface to look up the MBeans that are registered in the MBean Server. Then you use the `MBeanServer` interface to get or set MBean attributes or to invoke MBean operations. For the complete list of `MBeanServer` methods, refer to the JMX 1.0 API documentation, which you can download from <http://jcp.org/aboutJava/communityprocess/final/jsr003/index.html>. The archive that you download includes the API documentation.

In the WebLogic Server implementation of JMX, you use the `MBeanHome` interface to look up the `MBeanServer` interface.

The example class in [Listing 2-7](#):

1. Uses JNDI to retrieve the Administration `MBeanHome` interface. Because this example retrieves Administration MBeans, it must use the Administration `MBeanHome` interface.
2. Uses the Administration `MBeanHome` interface to retrieve the `MBeanServer` interface.
3. Uses the `MBeanServer.queryNames` method to look up all instances of `JDBCConnectionPoolMBean` in the domain. Note that the `queryNames` method signature requires the example to cast the string `"examples:type=JDBCConnectionPool,*"` as an `Object`.
4. Assigns the list of MBeans to a `Set` object and uses methods of the `Set` and `Iterator` interfaces to iterate through the list.

In the example, `weblogic` is a user who has permission to view and modify MBean attributes. For information about permissions to view and modify MBeans, refer to "[Protecting System Administration Operations](#)" in *WebLogic Server Administration Guide*.

Listing 2-7 Using the MBeanServer Interface

```
import java.util.Iterator;
import java.util.Set;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.AuthenticationException;
import javax.naming.CommunicationException;
import javax.naming.NamingException;
import javax.management.MBeanServer;
import javax.management.ObjectName;
import javax.management.QueryExp;
import weblogic.jndi.Environment;
import weblogic.management.MBeanHome;
import weblogic.management.RemoteMBeanServer;

public class ListJDBCInfo {

    public static void main(String[] args) {
        QueryExp query = null;
        MBeanHome home = null;
        RemoteMBeanServer homeServer = null;

        //domain variables
        String url = "t3://localhost:7001";
        String username = "weblogic";
        String password = "weblogic";

        //Setting an initial context.
        try {
            Environment env = new Environment();
            env.setProviderUrl(url);
            env.setSecurityPrincipal(username);
            env.setSecurityCredentials(password);
            Context ctx = env.getInitialContext();

            //Retrieving the Administration MBeanHome interface
            home = (MBeanHome) ctx.lookup(MBeanHome.ADMIN_JNDI_NAME);
            System.out.println("Got the Admin MBeanHome: " + home + " from the
                Admin server");

            //Retrieving the MBeanServer interface
            homeServer = home.getMBeanServer();

            //Retrieving a list of MBeans with object names that include
            //"JDBCConnectionPool"
            Set JDBCBeans = homeServer.queryNames(new
                ObjectName("mydomain:Type=JDBCConnectionPool,*"), query);
```

```
//where "query" could be any object that implements the JMX
//javax.management.QueryExp

for (Iterator itr = JDBCMBBeans.iterator(); itr.hasNext(); ) {
    ObjectName mbean = (ObjectName)itr.next();
    System.out.println("Matches to the MBean query:" + mbean);
}
} catch(Exception e){
    System.out.println(e);
}
}
}
```

Using WebLogicObjectName for WebLogic Server MBeans

When you instantiate a WebLogic Server MBean, the MBean Server registers the instance under a name that conforms to the `weblogic.management.WebLogicObjectName` conventions. If you know the `WebLogicObjectName` of an MBean, after you retrieve an `MBeanHome` interface, you can retrieve an MBean directly by name.

The MBean's `WebLogicObjectName` uses the following conventions to provide a unique identification for a given MBean across all domains:

```
domain:Name=name,Type=type[,Location=serverName]
[,TypeOfParentMBean=NameOfParentMBean][,TypeOfParentMBean1=NameOfParentMBean1]...
```

The order of the `attribute=value` pairs is not significant, but the name must begin with `domain:..` Note also that MBeans can express multiple parent MBeans.

For example, the following is the `WebLogicObjectName` for an `EJBComponentRuntime` MBean for an EJB in an application that is deployed on a server instance named `MyServer`. The first and fourth attribute/value pairs in this name, `ApplicationRuntime=MyServer_MyEAR` and `ServerRuntime=MyServer`, identify the parent MBeans of this EJB.

```
mydomain:ApplicationRuntime=MyServer_MyEAR,Location=MyServer,Name
=MyServer_MyEAR_SessionEJB,ServerRuntime=MyServer,Type=EJBCompon
entRuntime
```

The following table describes each name component.

Table 2-3 WebLogic Server MBean Naming Conventions

| This Component | Specifies |
|-------------------|--|
| <i>domain</i> | The name of the WebLogic Server administration domain. |
| Name= <i>name</i> | <p>The string that you provided when you created the associated resource. For example, when you create a JDBC connection pool, you must provide a name for that pool, such as <code>MyPool1</code>. The <code>JDBCConnectionPoolMBean</code> that represents <code>MyPool1</code> uses <code>Name=MyPool1</code> in its JMX object name.</p> <p>The <code>WebLogicObjectName.getName</code> method returns this value for any given MBean.</p> <p>If you create an MBean, you must specify a value for this Name component that is unique amongst all other MBeans in a domain.</p> |
| Type= <i>type</i> | <p>Refers to the interface class of which the MBean is an instance. All WebLogic Server MBeans are an instance of one of the interface classes defined in the <code>weblogic.management.configuration</code> or <code>weblogic.management.runtime</code> packages. For Configuration MBeans, Type also refers to whether an instance is an Administration MBean or a Local Configuration MBean. For a complete list of all WebLogic Server MBean interface classes, refer to the WebLogic Server Javadoc for the <code>weblogic.management.configuration</code> or <code>weblogic.management.runtime</code> packages.</p> <p>To determine the value that you provide for the Type component:</p> <ol style="list-style-type: none">1. Find the MBean's interface class and remove the MBean suffix from the class name. For example, for an MBean that is an instance of the <code>weblogic.management.runtime.JDBCConnectionPoolRuntime</code> MBean, use <code>JDBCConnectionPoolRuntime</code>.2. For a Local Configuration MBean, append <code>Config</code> to the name. For example, for a Local Configuration MBean that is an instance of the <code>weblogic.management.configuration.JDBCConnectionPool</code> MBean interface class, use <code>JDBCConnectionPoolConfig</code>. For the corresponding Administration MBean instance, use <code>JDBCConnectionPool</code>. |

2 Accessing WebLogic Server MBeans

Table 2-3 WebLogic Server MBean Naming Conventions

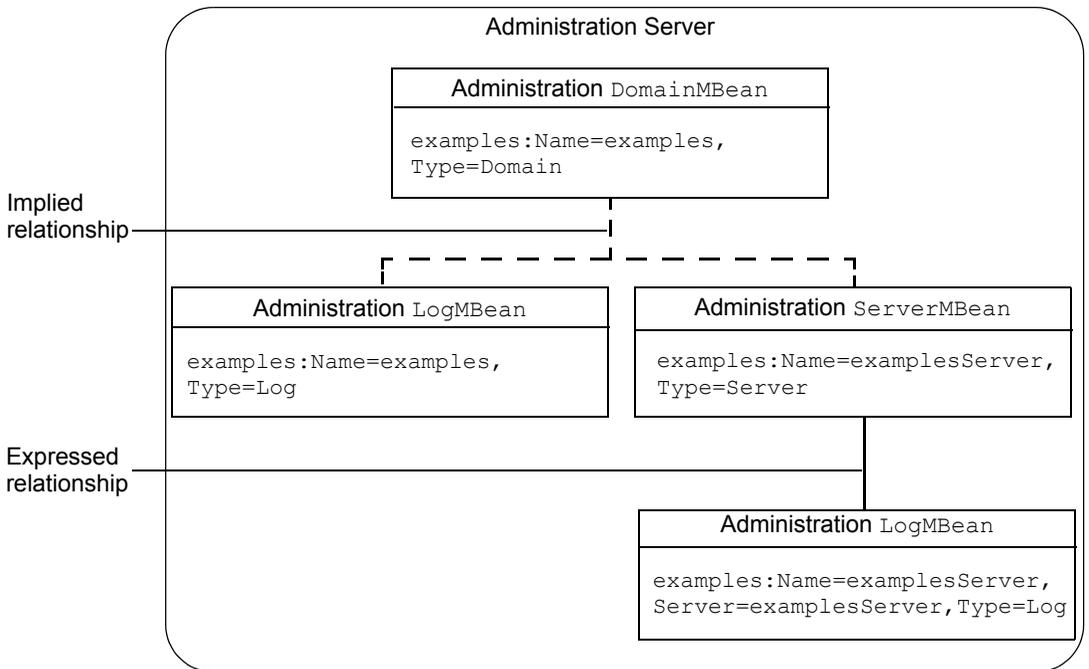
| This Component | Specifies |
|----------------------------------|--|
| <code>Location=servername</code> | <p>All Runtime and Local Configuration MBeans include a <code>Location</code> component that specifies the name of the server on which that MBean is located. Administration MBeans do not include this component.</p> <p>For example, for the <code>ServletRuntime</code> MBean that runs on a server named <code>myserver</code>, the <code>WebLogicObjectName</code> includes the following components:</p> <pre>mydomain:Name=myServlet,Type=ServletRuntime,Location= myserver</pre> <p>The <code>WebLogicObjectName.getLocation</code> method returns this value for any given MBean.</p> |

Table 2-3 WebLogic Server MBean Naming Conventions

| This Component | Specifies |
|--|--|
| <code>TypeOfParentMBean=NameOfParentMBean</code> | <p>Runtime, Local Configuration, or Administration MBeans that have a child relationship with a parent MBean use this extra attribute in their object names to identify the relationship.</p> <p>Note: With the exception of <code>DomainMBean</code>, all MBeans are direct or indirect children of the domain's <code>DomainMBean</code>. Because this parent-child relationship applies to all MBeans, it is not expressed in <code>WebLogicObjectName</code>.</p> <p>For example, an instance of <code>LogMBean</code> is used by a domain to configure the domain-wide log file. Each WebLogic Server instance also maintains its own instance of <code>LogMBean</code> to configure its server-specific log file. The <code>LogMBean</code> that a domain uses does not express a child relationship, while the <code>LogMBean</code> that a server instance uses expresses its child relationship with the server's <code>ServerMBean</code>. (See Figure 2-1.)</p> <p>To express the name of the Administration <code>LogMBean</code> that <code>examplesServer</code> uses to maintain its log file, use the following name:</p> <pre>examples:Name=examplesServer,Server=examplesServer,Type=Log</pre> <p>To express the name of the Local Configuration <code>LogMBean</code> that <code>examplesServer</code> uses to maintain its log file, use the following name:</p> <pre>examples:Location=examplesServer,Name=examplesServer,ServerConfig=examplesServer,Type=LogConfig</pre> <p>By convention, WebLogic Server child MBeans use the same value for the <code>Name</code> component as the parent MBean. For example, the <code>LogMBean</code> that is a child of the <code>examplesServer</code> <code>ServerMBean</code> uses <code>Name=examplesServer</code> in its <code>WebLogicObjectName</code>. WebLogic Server cannot follow this convention when a parent MBean has multiple children of the same type.</p> <p>To determine whether the <code>WebLogicObjectName</code> of an MBean expresses a parent-child relation, use the <code>WebLogicObjectName.getParent</code> method or the <code>weblogic.Admin GET</code> command.</p> |

[Figure 2-1](#) illustrates that one instance of `LogMBean` is a child of `DomainMBean` and is used to manage the domain-wide log file. Another instance of `LogMBean` is a child of a server instance's `ServerMBean` and is used to manage the server-specific log file. The `TypeOfParentMBean=NameOfParentMBean` component of the `WebLogicObjectName` removes any ambiguity in the application of an MBean instance.

Figure 2-1 Parent-Child Relation of LogMBean Instances



Using `weblogic.Admin` to Find the `WebLogicObjectName`

If you are unsure which values to supply for an MBean's `WebLogicObjectName`, you can use the `weblogic.Admin` utility to find the `WebLogicObjectName`. The utility can return information only for WebLogic Server MBeans that are on an active server instance.

For example, to find the `WebLogicObjectName` for the Administration instance of the `LogMBean` in the `examples` domain, enter the following command on the `examplesServer` Administration Server, where the Administration Server is listening on port 8001 and `weblogic` is the name and password of a user who has permission to view MBean attributes:

```
java weblogic.Admin -url localhost:8001 -username weblogic
  -password weblogic GET -pretty -type Log
```

The command returns the output in [Listing 2-8](#). Notice that the command returns two MBeans of type `Log` on the Administration Server. The first MBean, `examples:Name=examplesServer, Server=examplesServer, Type=Log`, has a child relationship with the `ServerMBean` of `examplesServer`; this relationship indicates that the MBean is the `LogMBean` that configures the server-specific log file. The second MBean, `examples:Name=examples, Type=Log`, has no child relationship, which indicates that it configures the domain-wide log file.

The `-pretty` causes the `weblogic.Admin` utility to place each MBean attribute and value on a separate line. Without this argument, the utility surrounds each attribute/value pair with curly braces `{}`, but all output is on a single line.

Listing 2-8 Output from `weblogic.Admin`

```
-----
MBeanName:
"examples:Name=examplesServer, Server=examplesServer, Type=Log"
  CachingDisabled: true
  FileCount: 7
  FileMinSize: 500
  FileName: examplesServer\examplesServer.log
  FileTimeSpan: 24
  Name: examplesServer
  Notes:
  NumberOfFilesLimited: false
  ObjectName: examplesServer
  Parent: examplesServer
  Registered: false
  RotationTime: 00:00
  RotationType: none
  Type: Log
-----
MBeanName: "examples:Name=examples, Type=Log"
  CachingDisabled: true
```

2 Accessing WebLogic Server MBeans

```
FileCount: 7
FileMinSize: 500
FileName: ./logs/wl-domain.log
FileTimeSpan: 24
Name: examples
Notes:
NumberOfFilesLimited: false
ObjectName: examples
Parent: examples
Registered: false
RotationTime: 00:00
RotationType: none
Type: Log
```

To view the Local Configuration MBean instances of `LogMBean`, append `Config` to the value of the `type` argument:

```
java weblogic.Admin -url localhost:8001 -username weblogic
-password weblogic GET -pretty -type LogConfig
```

The command returns output in [Listing 2-9](#). Notice that the `WebLogicObjectName` of the Local Configuration MBeans includes a `Location` component.

Listing 2-9 Local Configuration MBeans

```
-----
MBeanName:
"examples:Location=examplesServer,Name=examplesServer,ServerConfig=examplesServer,Type=LogConfig"
  CachingDisabled: true
  FileCount: 7
  FileMinSize: 500
  FileName: examplesServer\examplesServer.log
  FileTimeSpan: 24
  Name: examplesServer
  Notes:
  NumberOfFilesLimited: false
  ObjectName: examplesServer
  Registered: false
  RotationTime: 00:00
  RotationType: none
  Type: LogConfig
```

```
-----  
MBeanName:  
"examples:Location=examplesServer,Name=examples,Type=LogConfig"  
  CachingDisabled: true  
  FileCount: 7  
  FileMinSize: 500  
  FileName: ./logs/wl-domain.log  
  FileTimeSpan: 24  
  Name: examples  
  Notes:  
  NumberOfFilesLimited: false  
  ObjectName: examples  
  Registered: false  
  RotationTime: 00:00  
  RotationType: none  
  Type: LogConfig
```

3 Accessing and Changing Configuration Information

Configuration MBeans on the Administration Server (Administration MBeans) configure the managed resources on all WebLogic Server instances in a domain. To enhance performance, each server instance creates and uses local replicas of the Administration MBeans. These local replicas are called Local Configuration MBeans.

Note: While you can view the values of Local Configuration MBeans, BEA recommends that you do not change attribute values in Local Configuration MBeans. Instead, change only the values of Administration MBean attributes. When the Managed Server replicates the domain's configuration data, it uses the values that are stored in Administration MBeans. Communication problems can occur if the values in Administration MBeans and Local Configuration MBeans differ.

The following sections provide examples for programmatically viewing and modifying the configuration of WebLogic Server resources using the `weblogic.Admin` utility, the `JMX MBeanServer` APIs, and the WebLogic Server type-safe interface:

- [“Example: Using `weblogic.Admin` to View the Message Level for Standard Out” on page 3-2](#)
- [“Example: Configuring the Message Level for Standard Out” on page 3-3](#)

Example: Using `weblogic.Admin` to View the Message Level for Standard Out

This example uses the `weblogic.Admin` utility to connect directly to a Managed Server and look up the value of its `StdoutSeverityLevel` attribute. This attribute, which belongs to the server's `ServerMBean`, specifies a threshold for determining which severity-level of messages a server prints to its standard out.

While BEA recommends that you use only Administration MBeans to change values, there might be situations in which it is preferable to look up the values that are in Local Configuration MBeans. For example, the Administration Server might be down, making it impossible for you to access Administration MBeans.

The example command:

1. Uses the `-url` argument to connect to a Managed Server that runs on a host named `myHost` and that listens on port 8001.
2. Uses the `-username` and `-password` arguments to specify the credentials of a user who has permission to view MBean attributes. For information about permissions to view and modify MBeans, refer to “[Protecting System Administration Operations](#)” in *WebLogic Server Administration Guide*.
3. Uses the `GET` command to retrieve a Local Configuration MBean.

To specify a Local Configuration MBean, it removes `MBean` and appends `Config` to the `ServerMBean` interface name. Note that the `-type` value for a Local Configuration instance of the `ServerMBean` is `ServerConfig` while the `-type` value for the corresponding Administration MBean instance is `Server`. For more information, refer to the description of `Type` in [Table 2-3, “WebLogic Server MBean Naming Conventions,”](#) on page 2-21.

Listing 3-1 Configuring the Message Level

```
java weblogic.Admin -url myHost:8001 -username weblogic -password weblogic
GET -pretty -type ServerConfig
```

```
-----
MBeanName:
```

```
"examples:Location=examplesServer,Name=examplesServer,Type=ServerConfig"  
  AcceptBacklog: 50  
  AdministrationPort: 0  
...  
  
  StdoutDebugEnabled: false  
  StdoutEnabled: true  
  StdoutFormat: standard  
  StdoutLogStack: true  
  StdoutSeverityLevel: 16
```

Example: Configuring the Message Level for Standard Out

The class in this example changes the value of the `StdoutSeverityLevel` attribute in the `weblogic.management.configuration.ServerMBean` to change the level of messages that a server instance named `examplesServer` sends to standard out.

Because the example is changing configuration values, it changes the value in the Administration MBean and relies on the WebLogic management services to propagate the change to the Managed Server.

The example class:

1. Uses JNDI to look up the `AdministrationMBeanHome` interface on the Administration Server.
2. Uses the `MBeanHome.getMBean(String name, String type)` API to retrieve the type-safe interface of the `ServerMBean` Administration MBean for a server instance named `Server1`.
3. Uses the type-safe interface to invoke the `ServerMBean.setStdoutSeverityLevel` method and set the severity level to 64.

3 Accessing and Changing Configuration Information

In the example, `weblogic` is a user who has permission to view and modify MBean attributes. For information about permissions to view and modify MBeans, refer to “[Protecting System Administration Operations](#)” in *WebLogic Server Administration Guide*.

Listing 3-2 Configuring Standard Out Severity Level

```
import java.util.Set;
import java.util.Iterator;
import java.rmi.RemoteException;
import javax.naming.Context;
import javax.management.MBeanServer;
import javax.management.Attribute;
import java.lang.Object;

import weblogic.jndi.Environment;
import weblogic.management.MBeanHome;
import weblogic.management.configuration.ServerMBean;

public class ChangeStandardOut1 {

    public static void main(String[] args) {
        MBeanHome home = null;
        ServerMBean server = null;
        //domain variables
        String url = "t3://localhost:7001";
        String username = "weblogic";
        String password = "weblogic";
        String serverName = "Server1";

        //setting the initial context
        try {
            Environment env = new Environment();
            env.setProviderUrl(url);
            env.setSecurityPrincipal(username);
            env.setSecurityCredentials(password);
            Context ctx = env.getInitialContext();

            //getting the Administration MBeanHome
            home = (MBeanHome) ctx.lookup(MBeanHome.ADMIN_JNDI_NAME);

            // Using MBeanHome.getMBean(name, type) to retrieve a type-safe
            // interface for a ServerMBean
            server = (ServerMBean) home.getMBean(serverName, "Server");

            // Using ServerMBean.setStdoutSeverityLevel
            server.setStdoutSeverityLevel(64);
        }
    }
}
```

Example: Configuring the Message Level for Standard Out

```
// Providing feedback that operation succeeded.
System.out.println("Changed standard out severity level to: " +
    server.getStdoutSeverityLevel());
} catch (Exception e) {
    System.out.println("Caught exception: " + e);
}
}
```

3 *Accessing and Changing Configuration Information*

4 Accessing Runtime Information

WebLogic Server includes a large number of MBeans that provide information about the runtime state of managed resources. If you want to create applications that view and modify this runtime data, you must first use the `MBeanServer` interface or the WebLogic Server type-safe interface to retrieve Runtime MBeans. Then you use APIs in the `weblogic.management.runtime` package to view or change the runtime data. For information about viewing the API documentation, refer to [“Documentation for Runtime MBean APIs” on page 1-11](#).

The following sections provide examples for retrieving and modifying runtime information about WebLogic Server domains and server instances:

- [“Example: Determining the Active Domain and Servers” on page 4-2](#)
- [“Example: Viewing and Changing the Runtime State of a WebLogic Server Instance” on page 4-5](#)
- [“Example: Viewing Runtime Information About Clusters” on page 4-14](#)
- [“Viewing Runtime Information for EJBs” on page 4-16](#)

Example: Determining the Active Domain and Servers

The `MBeanHome` interface includes APIs that you can use to determine the name of the currently active domain and the name of a server instance. You can use this information in other APIs that take the domain name or a server name as arguments.

The example class in [Listing 4-1](#) does the following:

1. Retrieves the Administration `MBeanHome` interface.

While this example uses the Administration `MBeanHome`, you can also use a Local `MBeanHome` interface. A Local `MBeanHome` interface can retrieve only the name of the current domain and server instance. The Administration `MBeanHome` interface can retrieve the names of **all** currently active servers in the domain.

2. Uses `MBeanHome.getActiveDomain().getName()` to retrieve the name of the domain.
3. Uses the `getMBeansByType` method to retrieve the set of all `ServerRuntimeMBeans` in the domain.
4. Iterates through the set and compares the names of the `ServerRuntimeMBean` instances with the name of the WebLogic Server instance. If the instance is active, it prints the name of the server.
5. If the instance is active, it prints the name of the server.

In the following example, `weblogic` is the username and password for a user who has permission to view and modify MBean attributes. For information about permissions to modify MBeans, refer to “[Protecting System Administration Operations](#)” in *WebLogic Server Administration Guide*.

Listing 4-1 Determining the Active Domain and Servers

```
import java.util.Set;
import java.util.Iterator;
import javax.naming.Context;
```

Example: Determining the Active Domain and Servers

```
import weblogic.jndi.Environment;
import weblogic.management.MBeanHome;
import weblogic.management.runtime.ServerRuntimeMBean;

public class getActiveDomainAndServers {
    public static void main(String[] args) {
        MBeanHome home = null;

        //url of the Administration Server
        String url = "t3://localhost:7001";
        String username = "weblogic";
        String password = "weblogic";

        //setting the initial context
        try {
            Environment env = new Environment();
            env.setProviderUrl(url);
            env.setSecurityPrincipal(username);
            env.setSecurityCredentials(password);
            Context ctx = env.getInitialContext();

            //getting the Administration MBeanHome
            home = (MBeanHome) ctx.lookup(MBeanHome.ADMIN_JNDI_NAME);
        } catch (Exception e) {
            System.out.println("Exception caught: " + e);
        }

        //getting the name of the active domain
        try {
            System.out.println("Active Domain: " +
                home.getActiveDomain().getName() );
        } catch (Exception e) {
            System.out.println("Exception: " + e);
        }

        //getting the names of servers in the domain
        System.out.println("Active Servers: ");
        Set mbeanSet = home.getMBeansByType("ServerRuntime");
        Iterator mbeanIterator = mbeanSet.iterator();
        while(mbeanIterator.hasNext()) {
            ServerRuntimeMBean serverRuntime =
                (ServerRuntimeMBean)mbeanIterator.next();
            //printing the names of active servers
            if(serverRuntime.getState().equalsIgnoreCase("RUNNING")) {
                System.out.println("Name: " + serverRuntime.getName());
                System.out.println("ListenAddress: " +
                    serverRuntime.getListenAddress());
                System.out.println("ListenPort: " +
                    serverRuntime.getListenPort());
                //count++;
            }
        }
    }
}
```

4 Accessing Runtime Information

```
        }  
    }  
  
    System.out.println("Number of servers active in the domain: " +  
                       mbeanSet.size());  
}  
}
```

Using `weblogic.Admin` to Determine Active Domains and Servers

While you can compile and run the example code in [Listing 4-1](#) to determine active domains and servers, you can use the `weblogic.Admin` utility to accomplish a similar task without having to compile Java classes.

The following command returns the name of the currently active domain, where `AdminServer` is the domain's Administration Server, `MyHost` is the Administration Server's host computer, and `weblogic` is the name and password of a user who has permission to view MBean attributes:

```
java weblogic.Admin -url MyHost:8001 -username weblogic -password  
weblogic GET -type DomainRuntime -property Name
```

The command output includes the `WebLogicObjectName` of the `DomainRuntimeMBean` and the value of its `Name` attribute:

```
{MBeanName="myDomain:Location=AdminServer,Name=myDomain,ServerRun  
time=AdminServer,Type=DomainRuntime"{Name=myDomain}}
```

Example: Viewing and Changing the Runtime State of a WebLogic Server Instance

The `weblogic.management.runtime.ServerRuntimeMBean` interface provides runtime information about a WebLogic Server instance. For example, it indicates which listen ports and addresses a server is using. It also includes operations that can gracefully or forcefully shut down a server.

This section provides examples of finding `ServerRuntimeMBean` and using it to gracefully shut down a server instance. When you initiate a graceful shutdown, the server notifies subsystems to complete all in-work requests. After the subsystems complete their work, the server stops.

Each example illustrates a different way of retrieving `ServerRuntimeMBean`:

- [“Using a Local MBeanHome and getRuntimeMBean\(\)” on page 4-5](#)
- [“Using the Administration MBeanHome and getMBean\(\)” on page 4-10](#)
- [“Using the Administration MBeanHome and getMBeansByType\(\)” on page 4-8](#)
- [“Using the MBeanServer Interface” on page 4-12](#)

You cannot use the `weblogic.Admin` utility to change the value of Runtime MBean attributes.

Using a Local MBeanHome and getRuntimeMBean()

Each WebLogic Server instance hosts its own `MBeanHome` interface, which provides access to the Local Configuration and Runtime MBeans on the server instance. As opposed to using the Administration `MBeanHome` interface, using the local `MBeanHome` saves you the trouble of filtering Runtime MBeans to find those that apply to the current server. It also uses fewer network hops to access MBeans, because you are connecting directly to the server (instead of routing requests through the Administration Server).

4 Accessing Runtime Information

The `MBeanHome` interface includes the `getRuntimeMBean()` method, which returns only the top-level Runtime MBeans that reside on the current WebLogic Server (see [“Walking the Hierarchy of Local Configuration and Runtime MBeans”](#) on page 2-14). If you invoke `MBeanHome.getRuntimeMBean()` on the Administration Server, it returns only the Runtime MBeans that manage and monitor the Administration Server.

The example class in [Listing 4-2](#) does the following:

1. Configures a `javax.naming.Context` object with information for connecting to a server instance that listens for requests at `t3://ServerHost:7001`.
2. Uses the `Context.lookup` method to retrieve the local `MBeanHome` interface for the server instance.

The `MBeanHome.LOCAL_JNDI_NAME` field returns the JNDI name of the current server’s local `MBeanHome`.

3. Uses the `MBeanHome.getRuntimeMBean(String name, String type)` method to retrieve the `ServerRuntimeMBean` for the current server instance.
4. Invokes `ServerRuntimeMBean` methods to retrieve and modify the server state.

In the following example, `weblogic` is the username and password for a user who has permission to view and modify MBean attributes and `Server1` is the name of the WebLogic Server instance for which you want to view and change status. For information about permissions to modify MBeans, refer to [“Protecting System Administration Operations”](#) in *WebLogic Server Administration Guide*.

Listing 4-2 Using a Local MBeanHome and getRuntimeMBean()

```
import javax.naming.Context;

import weblogic.jndi.Environment;
import weblogic.management.MBeanHome;
import weblogic.management.runtime.ServerRuntimeMBean;

public class serverRuntime1 {

    public static void main(String[] args) {
        MBeanHome home = null;

        //domain variables
        String url = "t3://ServerHost:7001";
        String serverName = "Server1";
        String username = "weblogic";
```

Example: Viewing and Changing the Runtime State of a WebLogic Server Instance

```
String password = "weblogic";
ServerRuntimeMBean serverRuntime = null;

//setting the initial context
try {
    Environment env = new Environment();
    env.setProviderUrl(url);
    env.setSecurityPrincipal(username);
    env.setSecurityCredentials(password);
    Context ctx = env.getInitialContext();

    //getting the local MBeanHome
    home = (MBeanHome) ctx.lookup(MBeanHome.LOCAL_JNDI_NAME);
    System.out.println("Got the MBeanHome: " + home + " for server: " +
        serverName);
} catch (Exception e) {
    System.out.println("Caught exception: " + e);
}

// Here we use the getRuntimeMBean method to access the
//ServerRuntimeMbean of the server instance.

try {
    serverRuntime =
        (ServerRuntimeMBean)home.getRuntimeMBean
            (serverName,"ServerRuntime");
    System.out.println("Got serverRuntimeMBean: " + serverRuntime);

    //Using ServerRuntimeMBean to retrieve and change state.
    System.out.println("Current state: " + serverRuntime.getState() );
} catch (javax.management.InstanceNotFoundException e) {
    System.out.println("Caught exception: " + e);
}

try{
    serverRuntime.shutdown();
    System.out.println("Current state: " + serverRuntime.getState() );
} catch (weblogic.server.ServerLifecycleException e) {
    System.out.println("Caught exception: " + e);
}
}
}
```

Using the Administration MBeanHome and getMBeansByType()

Like the example in [Listing 4-1, “Determining the Active Domain and Servers,”](#) on [page 4-2](#), the example class in this section uses the Administration MBeanHome interface to retrieve a ServerRuntime MBean. The Administration MBeanHome provides a single access point for all MBeans in the domain, but it requires you to either construct the WebLogicObjectName of the MBean you want to retrieve or to filter MBeans to find those that apply to a specific current server.

The example class in [Listing 4-3](#) does the following:

1. Retrieves the Administration MBeanHome interface.
2. Uses the MBeanHome.getMBeansByType method to retrieve the set of all ServerRuntime MBeans in the domain.
3. Assigns the list of MBeans to a Set object and uses methods of the Set and Iterator interfaces to iterate through the list.
4. Uses the ServerRuntimeMBean.getName method to retrieve the Name component of the MBean’s WebLogicObjectName. It then compares the Name value with another value.
5. When it finds the ServerRuntimeMBean for a specific server instance, it uses the ServerRuntimeMBean.getState method to return the current server state.
6. Then it invokes the ServerRuntimeMBean.shutdown() method, which initiates a graceful shutdown.

In the following example, `weblogic` is the username and password for a user who has permission to change the state of servers, and `Server1` is the name of the WebLogic Server instance for which you want to view and change status. For information about permissions to modify MBeans, refer to “[Protecting System Administration Operations](#)” in *WebLogic Server Administration Guide*.

Listing 4-3 Using the Administration MBeanHome and getMBeansByType()

```
import java.util.Set;
import java.util.Iterator;
import javax.naming.Context;
```

Example: Viewing and Changing the Runtime State of a WebLogic Server Instance

```
import weblogic.jndi.Environment;
import weblogic.management.MBeanHome;
import weblogic.management.runtime.ServerRuntimeMBean;

public class serverRuntimeInfo {
    public static void main(String[] args) {
        MBeanHome home = null;

        //domain variables
        String url = "t3://localhost:7001";
        String serverName = "Server1";
        String username = "weblogic";
        String password = "weblogic";
        ServerRuntimeMBean serverRuntime = null;
        Set mbeanSet = null;
        Iterator mbeanIterator = null;

        //Setting the initial context
        try {
            Environment env = new Environment();
            env.setProviderUrl(url);
            env.setSecurityPrincipal(username);
            env.setSecurityCredentials(password);
            Context ctx = env.getInitialContext();

            // Getting the Administration MBeanHome.
            home = (MBeanHome) ctx.lookup(MBeanHome.ADMIN_JNDI_NAME);
            System.out.println("Got the Admin MBeanHome: " + home );
        } catch (Exception e) {
            System.out.println("Exception caught: " + e);
        }
    }

    // Using the getMBeansByType method to get the set of
    //ServerRuntime mbeans.
    try {
        mbeanSet = home.getMBeansByType("ServerRuntime");
        mbeanIterator = mbeanSet.iterator();
        // Comparing the name of the server in each ServerRuntime
        // MBean to the value specified by serverName
        while(mbeanIterator.hasNext()) {
            serverRuntime = (ServerRuntimeMBean)mbeanIterator.next();
            if(serverRuntime.getName().equals(serverName)) {
                System.out.println("Found the serverRuntimeMBean: " +
                    serverRuntime + " for: " + serverName);
                System.out.println("Current state: " +
                    serverRuntime.getState() );
                System.out.println("Stopping the server ...");
                serverRuntime.shutdown();
                System.out.println("Current state: " +
                    serverRuntime.getState() );
            }
        }
    }
}
```

```
        }
    }
} catch (Exception e) {
    System.out.println("Caught exception: " + e);
}
}
```

Using the Administration MBeanHome and getMBean()

Instead of retrieving a list of all MBeans and then filtering the list to find the `ServerRuntimeMBean` for a specific server, this example uses the MBean naming conventions to construct the `WebLogicObjectName` for the `ServerRuntimeMBean` on a server instance named `Server1`. For information about constructing a `WebLogicObjectName`, refer to [“Using WebLogicObjectNames for WebLogic Server MBeans”](#) on page 2-20.

To make sure that you supply the correct object name, use the `weblogic.Admin GET` command. For example, the following command returns the object name and list of attributes of the `ServerRuntimeMBean` for a server instance that runs on a host computer named `MyHost`:

```
java weblogic.Admin -url http://MyHost:7001 -username weblogic
    -password weblogic GET -pretty -type ServerRuntime
```

For more information about using the `weblogic.Admin` utility to find information about MBeans, refer to [“MBean Management Command Reference”](#) in the *WebLogic Server Administration Guide*.

In [Listing 4-4](#), `weblogic` is the username and password for a user who has permission to view and modify MBean attributes, `Server1` is the name of the WebLogic Server instance for which you want to view and change status, and `mihirDomain` is the name of the WebLogic Server administration domain. For information about permissions to modify MBeans, refer to [“Protecting System Administration Operations”](#) in *WebLogic Server Administration Guide*.

Example: Viewing and Changing the Runtime State of a WebLogic Server Instance

Listing 4-4 Using the Administration MBeanHome and getMBean()

```
import java.util.Set;
import java.util.Iterator;
import javax.naming.Context;

import weblogic.jndi.Environment;
import weblogic.management.MBeanHome;
import weblogic.management.runtime.ServerRuntimeMBean;
import weblogic.management.WebLogicObjectName;

public class serverRuntimeInfo2 {
    public static void main(String[] args) {
        MBeanHome home = null;
        //domain variables
        String url = "t3://localhost:7001";
        String serverName = "Server1";
        String username = "weblogic";
        String password = "weblogic";
        String domain = "examples";
        ServerRuntimeMBean serverRuntime = null;

        //setting the initial context
        try {
            Environment env = new Environment();
            env.setProviderUrl(url);
            env.setSecurityPrincipal(username);
            env.setSecurityCredentials(password);
            Context ctx = env.getInitialContext();

            home = (MBeanHome) ctx.lookup(MBeanHome.ADMIN_JNDI_NAME);
            System.out.println("Got Admin MBeanHome from the Admin server: "
                + home);
        } catch (Exception e) {
            System.out.println("Exception caught: " + e);
        }

        try {
            WebLogicObjectName objName = new WebLogicObjectName(serverName,
                "ServerRuntime",home.getDomainName(),serverName);
            System.out.println("Created WebLogicObjectName: " + objName);
            serverRuntime = (ServerRuntimeMBean)home.getMBean(objName);
            System.out.println("Got the serverRuntime using the adminHome: " +
                serverRuntime );
            System.out.println("Current state: " + serverRuntime.getState() );
            System.out.println("Stopping the server ...");

            //changing the state to SHUTDOWN
            serverRuntime.shutdown();
        }
    }
}
```

4 Accessing Runtime Information

```
        System.out.println("Current state: " + serverRuntime.getState() );
    } catch(Exception e) {
        System.out.println("Exception: " + e);
    }
}
}
```

Using the MBeanServer Interface

The example in this section uses a standard JMX approach for interacting with MBeans. It uses the `Administration MBeanHome` interface to retrieve the `javax.management.MBeanServer` interface and then uses `MBeanServer` to retrieve the value of the `ListenPort` attribute of the `ServerRuntimeMBean` for a server instance named `Server1`.

In the following example, `weblogic` is the username and password for a user who has permission to view and modify MBean attributes and `mihirDomain` is the name of the WebLogic Server administration domain. For information about permissions to modify MBeans, refer to “[Protecting System Administration Operations](#)” in *WebLogic Server Administration Guide*.

Listing 4-5 Using the Administration MBeanHome and getMBean()

```
import java.util.Set;
import java.util.Iterator;
import javax.naming.Context;
import javax.management.MBeanServer;

import weblogic.jndi.Environment;
import weblogic.management.MBeanHome;
import weblogic.management.WebLogicObjectName;

public class serverRuntimeInfo3 {
    public static void main(String[] args) {
        MBeanHome home = null;

        //domain variables
        String url = "t3://localhost:7001";
        String serverName = "Server1";
        String username = "weblogic";
        String password = "weblogic";
```

Example: Viewing and Changing the Runtime State of a WebLogic Server Instance

```
Object attributeValue = null;
MBeanServer homeServer = null;

//setting the initial context
try {
    Environment env = new Environment();
    env.setProviderUrl(url);
    env.setSecurityPrincipal(username);
    env.setSecurityCredentials(password);
    Context ctx = env.getInitialContext();

    // Getting the Administration MBeanHome.
    home = (MBeanHome) ctx.lookup(MBeanHome.ADMIN_JNDI_NAME);
    System.out.println("Got Admin MBeanHome from the Admin server: " +
                                                                home);
} catch (Exception e) {
    System.out.println("Exception caught: " + e);
}
try {
    // Creating the mbean object name.
    WebLogicObjectName objName = new WebLogicObjectName(serverName,
                                                         "ServerRuntime",home.getDomainName(),serverName);
    System.out.println("Created WebLogicObjectName: " + objName);

    //Retrieving the MBeanServer interface
    homeServer = home.getMBeanServer();

    //Retrieving the ListenPort attribute of ServerRuntimeMBean
    attributeValue = homeServer.getAttribute(objName, "ListenPort");
    System.out.println("ListenPort for " + serverName + " is:" +
                                                                attributeValue);
} catch(Exception e) {
    System.out.println("Exception: " + e);
}
}
```

Example: Viewing Runtime Information About Clusters

The example in this section retrieves the number and names of WebLogic Server instances currently running in a cluster. It uses `weblogic.management.runtime.ClusterRuntimeMBean`, which provides information about a single Managed Server's view of the members of a WebLogic cluster.

Only Managed Servers host instances of `ClusterRuntimeMBean`, and you must retrieve the `ClusterRuntimeMBean` instance from a Managed Server that is actively participating in a cluster.

To make sure that it retrieves a `ClusterRuntimeMBean` from an active Managed Server that is in a cluster, this example does the following:

1. Retrieves the `AdministrationMBeanHome`, which runs on the Administration Server and can provide access to all `ClusterRuntimeMBeans` in the domain.
2. Retrieves all `ClusterRuntimeMBeans` and determines whether they belong to a specific cluster.
3. Finds one `ClusterRuntimeMBean` for a Managed Server in the cluster of interest.
4. Uses the `ClusterRuntimeMBean` APIs on the Managed Server to determine the number and name of active servers in the cluster.

In the example, `weblogic` is the username and password for a user who has permission to view and modify MBean attributes. For information about permissions to modify MBeans, refer to “[Protecting System Administration Operations](#)” in *WebLogic Server Administration Guide*.

Listing 4-6 Retrieving a List of Servers Running in a Cluster

```
import java.util.Set;
import java.util.Iterator;
import java.rmi.RemoteException;
import javax.naming.Context;
```

Example: Viewing Runtime Information About Clusters

```
import weblogic.jndi.Environment;
import weblogic.management.MBeanHome;
import javax.management.ObjectName;
import weblogic.management.WebLogicMBean;
import weblogic.management.runtime.ClusterRuntimeMBean;
import weblogic.management.WebLogicObjectName;
import weblogic.management.MBeanHome;

public class getRunningServersInCluster {
    public static void main(String[] args) {
        MBeanHome home = null;

        //domain variables
        String url = "t3://localhost:7001"; //url of the Administration Server
        /* If you have more than one cluster in your domain, define a list of
         * all the servers in the cluster. You compare the servers in the domain
         * with this list to determine which servers are in a specific cluster.
         */
        String server1 = "cs1"; // name of server in the cluster
        String server2 = "cs2"; // name of server in the cluster
        String username = "weblogic";
        String password = "weblogic";
        ClusterRuntimeMBean clusterRuntime = null;
        Set mbeanSet = null;
        Iterator mbeanIterator = null;
        String name = "";
        String[] aliveServerArray = null;

        //Setting the initial context
        try {
            Environment env = new Environment();
            env.setProviderUrl(url);
            env.setSecurityPrincipal(username);
            env.setSecurityCredentials(password);
            Context ctx = env.getInitialContext();

            // Getting the Administration MBeanHome.
            home = (MBeanHome) ctx.lookup(MBeanHome.ADMIN_JNDI_NAME);

            // Retrieving a list of ClusterRuntime MBeans in the domain.
            mbeanSet = home.getMBeansByType("ClusterRuntime");
            mbeanIterator = mbeanSet.iterator();
            while(mbeanIterator.hasNext()) {

                // Retrieving one ClusterRuntime MBean from the list.
                clusterRuntime = (ClusterRuntimeMBean)mbeanIterator.next();
                // Getting the name of the ClusterRuntime MBean.
                name = clusterRuntime.getName();
                // Determining if the current ClusterRuntimeMBean belongs to a
                // server in the cluster of interest.
            }
        }
    }
}
```

4 Accessing Runtime Information

```
        if(name.equals(server1) || name.equals(server2) ) {
            // Using the current ClusterRuntimeMBean to retrieve the
            // number of servers in the cluster.
            System.out.println("\nNumber of active servers in the
                cluster: " + clusterRuntime.getAliveServerCount());
            // Retrieving the names of servers in the cluster.
            aliveServerArray = clusterRuntime.getServerNames();
            break;
        }
    }
} catch (Exception e) {
    System.out.println("Caught exception: " + e);
}
if(aliveServerArray == null) {
    System.out.println("\nThere are no running servers in the cluster");
    System.exit(1);
}

System.out.println("\nThe running servers in the cluster are: ");
for (int i=0; i < aliveServerArray.length; i++) {
    System.out.println("server " + i + " : " + aliveServerArray[i]);
}
}
```

Viewing Runtime Information for EJBs

For each EJB that you deploy on a server instance, WebLogic Server instantiates MBean types from the `weblogic.management.runtime` package (see [Table 4-1](#)). For more information about the MBeans in the `weblogic.management.runtime` package, refer to the [WebLogic Server Javadoc](#).

Table 4-1 MBeans that Provide Runtime Information for EJBs

| MBean Type | Description |
|---------------------------------------|--|
| <code>EJBComponentRuntimeMBean</code> | The top level interface for all runtime information collected for an EJB module. |

Table 4-1 MBeans that Provide Runtime Information for EJBs

| MBean Type | Description |
|---|--|
| <code>StatefulEJBRuntimeMBean</code> | Instantiated for stateful session beans only. Contains methods for accessing EJB runtime information collected for a Stateful Session Bean. |
| <code>StatelessEJBRuntimeMBean</code> | Instantiated for stateless session beans only. Contains methods for accessing EJB runtime information collected for a Stateless Session Bean. |
| <code>MessageDrivenEJBRuntimeMBean</code> | Instantiated for message driven bean only. Contains methods for accessing EJB runtime information collected for a Message Driven Bean. |
| <code>EntityEJBRuntimeMBean</code> | Contains methods for accessing EJB runtime information collected for an Entity Bean. |
| <code>EJBCacheRuntimeMBean</code> | Contains methods for accessing cache runtime information collected for an EJB. |
| <code>EJBLockingRuntimeMBean</code> | Contains methods for accessing lock manager runtime information collected for an EJB. |
| <code>EJBTransactionRuntimeMBean</code> | Contains methods for accessing transaction runtime information collected for an EJB. |
| <code>EJBPoolRuntimeMBean</code> | Instantiated for stateless session beans only. Contains methods for accessing free pool runtime information collected for a stateless session EJB. WebLogic Server uses a free pool to improve performance and throughput for stateless session EJBs. The free pool stores unbound stateless session EJBs. Unbound EJB instances are instances of a stateless session EJB class that are not processing a method call. |

WebLogic Server provides an additional, abstract interface, `EJBRuntimeMBean`, which contains methods that the other EJB runtime MBeans use.

EJB runtime MBeans are instantiated within a hierarchy. For example:

- Each EJB jar file exposes its runtime data through an instance of `EJBComponentRuntimeMBean`.

4 Accessing Runtime Information

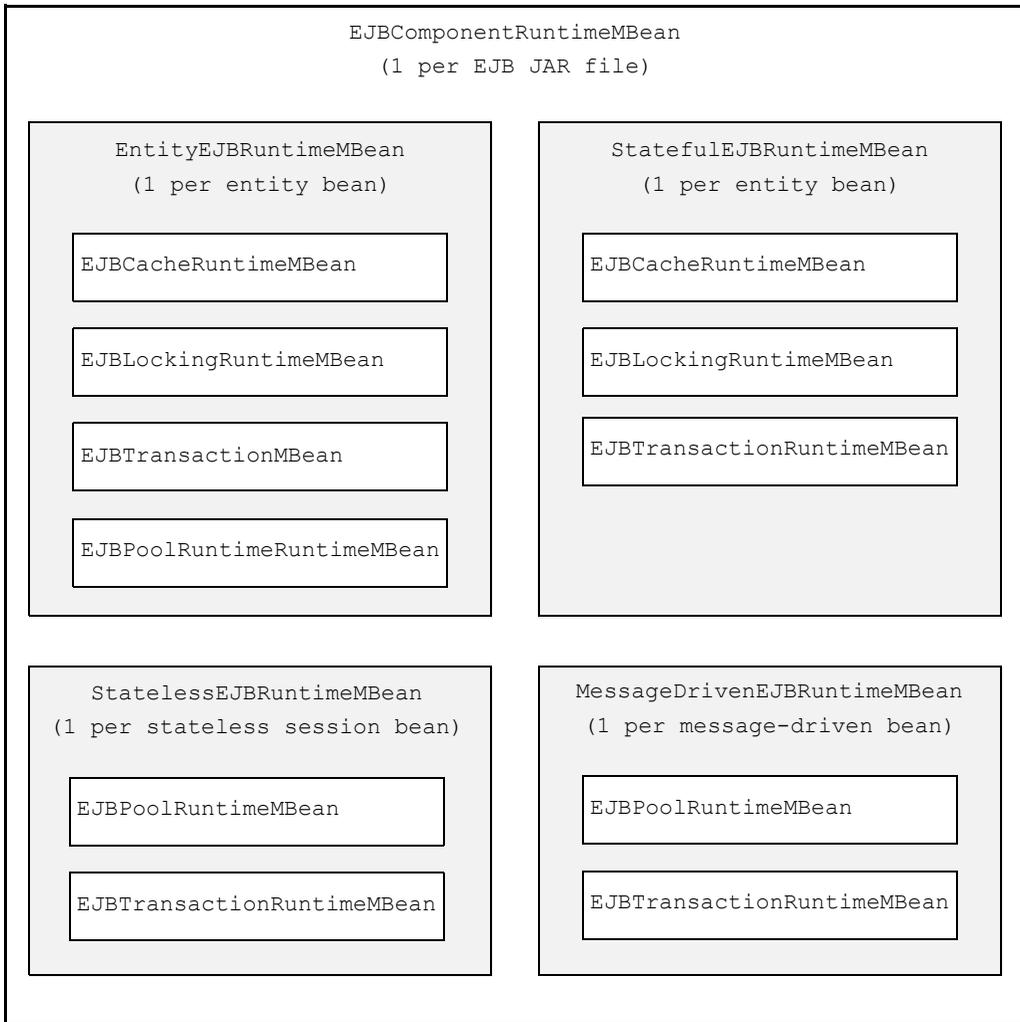
- Each entity bean within the jar exposes its runtime data through an instance of `EntityEJBRuntimeMBean`.
- Each `EntityEJBRuntimeMBean` is the parent of up to four additional MBeans. `EntityEJBRuntimeMBean` is always the parent of the `EJBCacheRuntimeMBean`, `EJBTransactionRuntimeMBean`, and `EJBPoolRuntimeMBean` MBeans. The fourth child MBean, `EJBLockingRuntimeMBean`, is only created if the entity bean uses an exclusive concurrency strategy (which is configured in the `weblogic-ejb-jar.xml` deployment descriptor).

Depending on the type of runtime data that you retrieve, you typically also need to retrieve the name of any parent MBeans to provide context for the data. For example, if you retrieve the value of

`EJBTransactionRuntimeMBean.TransactionsRolledBackTotalCount`, you also retrieve the name of the parent `EJBEntityRuntimeMBean` to determine which entity bean the value comes from.

[Figure 4-1](#) illustrates the hierarchical relationships.

Figure 4-1 Hierarchy of EJB Runtime MBeans



Example: Retrieving Runtime Information for All Stateful and Stateless EJBs

To retrieve runtime information for all EJBs deployed in a domain, the example in [Listing 4-7](#) does the following:

1. Connects to the Administration Server and retrieves the Administration `MBeanHome` interface.

If you want to retrieve runtime information only for the EJBs that are deployed on a specific server instance, you can connect to the specific server instance and retrieve the local `MBeanHome` interface. For more information, refer to [“Example: Retrieving a Local MBeanHome from an Internal Client” on page 2-9](#).

2. To display the number of idle bean instances in the free pool., the example:
 - a. Invokes the `MBeanHome.getMBeansByType` to retrieve all `StatelessEJBRuntimeMBeans`.
 - b. For each stateless EJB, it invokes the `displayEJBInfo` method (which is defined later in this class). This method:
 - Invokes the `StatelessEJBRuntimeMBean.getEJBName` method (which all EJB runtime MBeans inherit from `EJBRuntimeMBean`) to retrieve the name of the MBean.
 - Walks up the MBean hierarchy to retrieve the names of the parent EJB component and application.

All EJBs are packaged within an EJB component, which functions as a J2EE module. EJB components can be packaged with an enterprise application.
 - c. Invokes the `StatelessEJBRuntimeMBean.getPoolRuntime` method to retrieve the `EJBPoolRuntimeMBean` that is associated with the stateless EJB.
 - d. Invokes the `EJBPoolRuntimeMBean.getIdleBeansCount` method.
3. To determine percentage of transactions that have been rolled back for each stateful EJB in the domain, the example:
 - a. Invokes the `MBeanHome.getMBeansByType` to retrieve all `StatefulEJBRuntimeMBeans`.
 - b. Invokes the `displayEJBInfo` method (which is defined later in this class).

- c. Invokes the `EJBRuntime.getTransactionRuntime` method to retrieve the `EJBTransactionRuntimeMBean` that is associated with the stateful EJB.
- d. Invokes the `EJBTransactionRuntimeMBean.getTransactionRollbackTotalCount` and `getTransactionsCommittedTotalCount` methods.
- e. Divides the number of committed transactions by the number rolled back transactions to determine the percentage of rolled back transactions.

Listing 4-7 Viewing Runtime Information for EJBs

```
import java.util.Iterator;
import java.util.Set;
import javax.management.InstanceNotFoundException;
import javax.naming.Context;
import javax.naming.InitialContext;

import weblogic.management.MBeanHome;
import weblogic.management.WebLogicObjectName;
import weblogic.management.configuration.ApplicationMBean;
import weblogic.management.configuration.EJBComponentMBean;
import weblogic.management.configuration.ServerMBean;
import weblogic.management.runtime.EJBComponentRuntimeMBean;
import weblogic.management.runtime.EJBPoolRuntimeMBean;
import weblogic.management.runtime.EJBRuntimeMBean;
import weblogic.management.runtime.EJBTransactionRuntimeMBean;
import weblogic.management.runtime.StatelessEJBRuntimeMBean;
import weblogic.jndi.Environment;

public final class EJBMonitor {
    private String url = "t3://localhost:7001";
    private String user = "weblogic";
    private String password = "weblogic";
    private MBeanHome mBeanHome; // admin

    public EJBMonitor() throws Exception {
        Environment env = new Environment();
        env.setProviderUrl(url);
        env.setSecurityPrincipal(user);
        env.setSecurityCredentials(password);
        Context ctx = env.getInitialContext();
        mBeanHome = (MBeanHome) ctx.lookup(MBeanHome.ADMIN_JNDI_NAME);
    }

    public void displayStatelessEJBPoolIdleCount()
        throws Exception
```

4 Accessing Runtime Information

```
{
    int idleCount = 0;
    String type = "StatelessEJBRuntime";
    Set beans = mBeanHome.getMBeansByType(type);
    System.out.println("Printing Stateless Session pool idle count:");
    for(Iterator it=beans.iterator();it.hasNext();) {
        StatelessEJBRuntimeMBean rt = (StatelessEJBRuntimeMBean)it.next();
        displayEJBInfo(rt);
        EJBPooolRuntimeMBean pool = rt.getPoolRuntime();
        idleCount = pool.getIdleBeansCount();
    }
    System.out.println("Pool Idle Bean Count: "+ idleCount +"\n");
}

public void displayStatefulEJBTransactionRollbackPercentages()
throws Exception
{
    String type = "StatefulEJBRuntime";
    Set beans = mBeanHome.getMBeansByType(type);
    System.out.println("Printing Stateful transaction rollback
        percentages:");
    for(Iterator it=beans.iterator();it.hasNext();) {
        EJBRuntimeMBean rt = (EJBRuntimeMBean)it.next();
        displayEJBInfo(rt);
        EJBTransactionRuntimeMBean trans = rt.getTransactionRuntime();
        String rollbackPercentage = "0";
        long rollbackCount = trans.getTransactionsRolledBackTotalCount();
        if(rollbackCount > 0) {
            long totalTransactions = rollbackCount +
                trans.getTransactionsCommittedTotalCount();
            rollbackPercentage =
                ""+(float)rollbackCount/totalTransactions*100;
        }

        System.out.println("Transaction rollback percentage: "+
            rollbackPercentage +"\n");
    }
}

private void displayEJBInfo(EJBRuntimeMBean rt) throws Exception {
    System.out.println("EJB Name: "+rt.getEJBName());
    EJBComponentRuntimeMBean compRTMBean =
        EJBComponentMBean compMBean = compRTMBean.getEJBComponent();
    ApplicationMBean appMBean = (ApplicationMBean)compMBean.getParent();
    System.out.println("Application Name: "+appMBean.getName());
    System.out.println("Component Name: "+compMBean.getName());
    WebLogicObjectName objName = rt.getObjectName();
    System.out.println("Server Name: "+objName.getLocation());
}
```

```
public static void main(String[] argv) throws Exception {
    EJBMonitor m = new EJBMonitor();
    m.displayStatelessEJBPoolIdleCount();
    m.displayStatefulEJBTransactionRollbackPercentages();
}
}
```

5 Using WebLogic Server MBean Notifications and Monitors

To report changes in configuration and runtime information, all WebLogic Server MBeans emit JMX notifications. A **notification** is a JMX object that describes a state change or some other specific condition that has occurred in an underlying resource.

You can create Java classes called **listeners** that listen for these notifications. For example, your application can include a listener that receives notifications when applications are deployed, undeployed, or redeployed.

The following sections describe working with notifications and listeners:

- [“How Notifications are Broadcast and Received” on page 5-2](#)
- [“Monitoring Changes in MBeans” on page 5-3](#)
- [“Best Practices: Listening Directly Compared to Monitoring” on page 5-6](#)
- [“Listening for Notifications from WebLogic Server MBeans: Main Steps” on page 5-9](#)
- [“Using Monitor MBeans to Observe Changes: Main Steps” on page 5-24](#)

How Notifications are Broadcast and Received

All WebLogic Server MBeans implement the `javax.management.NotificationBroadcaster` interface, which enable them to emit different types of notification objects depending on the type of event that occurs. For example, MBeans emit notifications when the values of their attributes change.

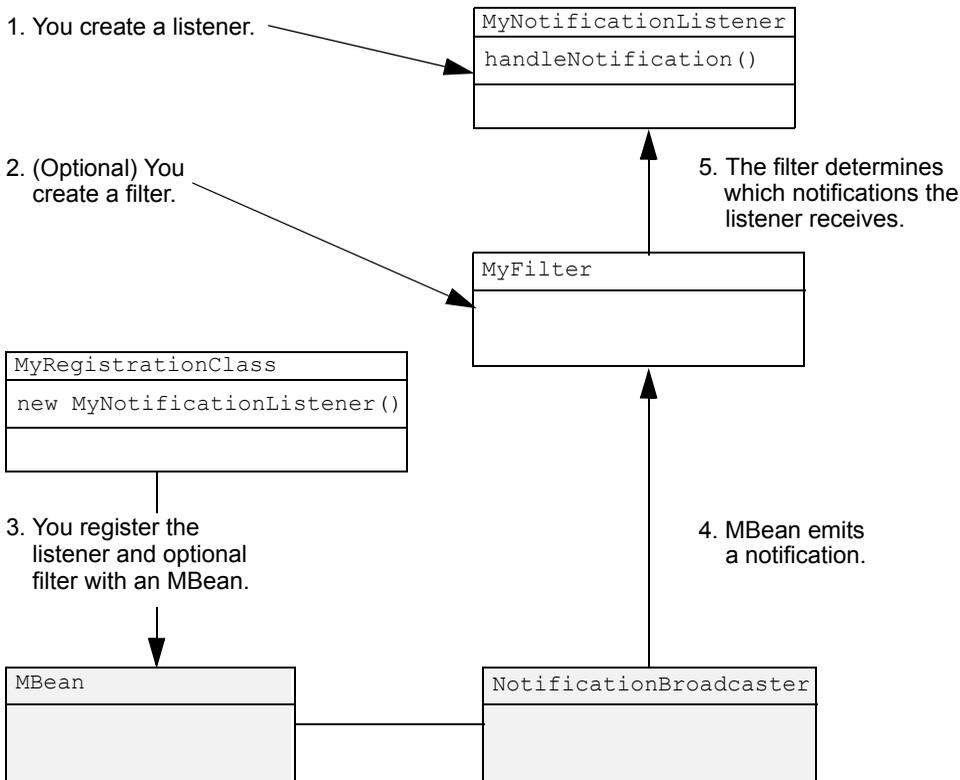
To listen for these notifications, you create a listener class that implements `javax.management.NotificationListener`.

By default, your listener receives all notifications that the MBean emits. However, typically, you want your listener to retrieve only specific notifications. For example, the `LogBroadcasterRuntime` MBean emits a notification each time a WebLogic Server instance generates a log message. Usually you listen for only specific log messages, such as messages of specific severity level. To limit the notifications that your listener receives, you can create a notification filter.

After creating your listener and optional filter, you register the classes with the MBeans from which you want to receive notifications.

[Figure 5-1](#) shows a basic system in which a `NotificationListener` receives only a subset of the notifications that an MBean broadcasts.

Figure 5-1 Receiving Notifications from an MBean



For a complete explanation of JMX notifications and how they work, download the JMX 1.0 specification from <http://jcp.org/aboutJava/communityprocess/final/jsr003/index.html>.

Monitoring Changes in MBeans

WebLogic Server includes a set of **monitor MBeans** that can be configured to periodically observe MBeans and emit JMX notifications only if a specific MBean attribute has changed beyond a specific threshold. A monitor MBean can observe the

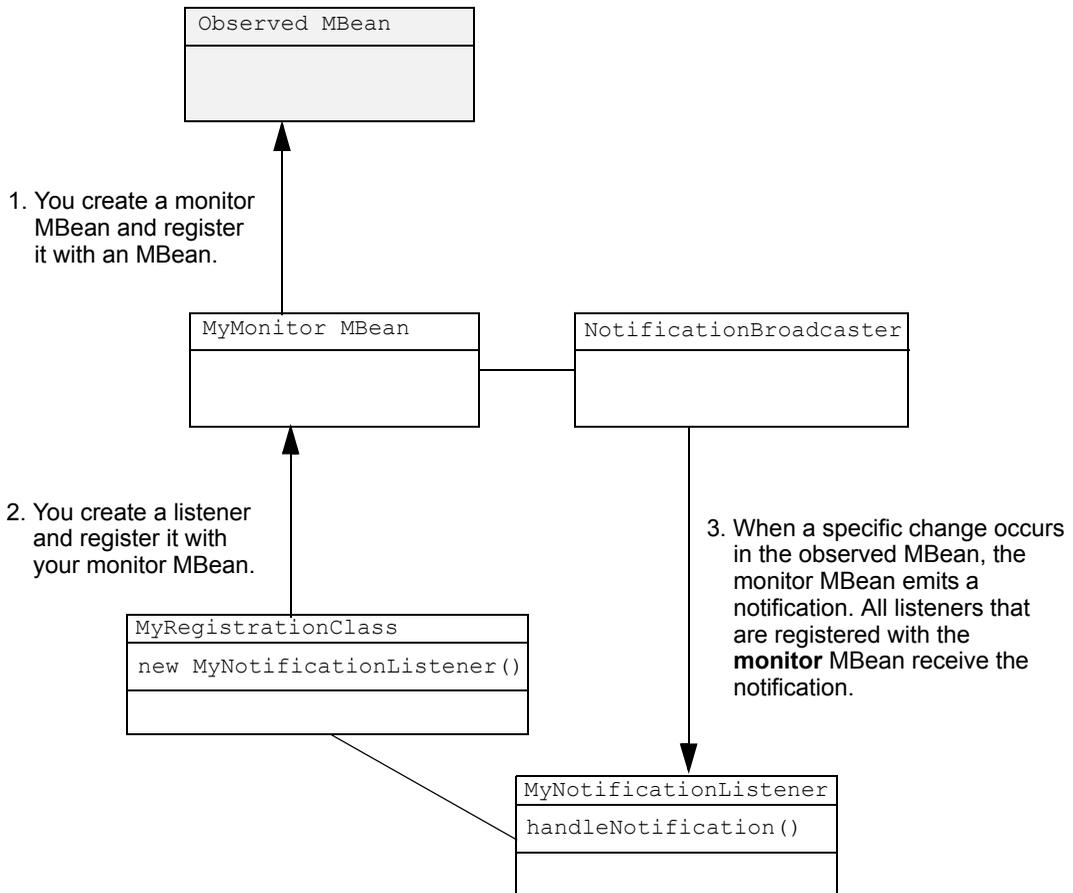
exact value of an attribute in an MBean, or optionally, the difference between two consecutive values of a numeric attribute. The value that a monitor MBean observes is called the **derived gauge**.

When the value of the derived gauge satisfies a set of conditions, the monitor MBean emits a specific notification type. Monitors can also send notifications when certain error cases are encountered while monitoring an attribute value.

To use monitor MBeans, you configure and register a monitor with a WebLogic Server MBean. Then you create a listener class and register the class with the **monitor MBean**. Because monitor MBeans emit only very specific types of notification, you usually do not use filters when listening for notifications from monitor MBeans.

[Figure 5-2](#) shows a basic system in which a monitor MBean is registered with a WebLogic Server MBean. A `NotificationListener` is registered with the monitor MBean, and it receives notifications when the conditions within the monitor MBean are satisfied.

Figure 5-2 Monitor MBeans



Best Practices: Listening Directly Compared to Monitoring

WebLogic Server provides two ways to be notified about changes in an MBean: you can create a listener and register it directly with an MBean (see [Figure 5-1](#)), or you can configure a monitor MBean that periodically observes an MBean and sends notifications when an attribute value satisfies criteria that you specify (see [Figure 5-2](#)). The method that you choose depends mostly on the complexity of the situations in which you want to receive notifications.

If your requirements are simple, registering a listener directly with an MBean is the preferred technique. The `NotificationListener` and `NotificationFilter` interfaces, which are classes that you implement in your listener and filter, provide few facilities for comparing values with thresholds and other values. You must create your own code to evaluate the data within notifications and respond accordingly. However, the advantage of registering a listener directly with an MBean is that the MBean pushes its notifications to your listener and you are notified of a change almost immediately.

If your notification requirements are sufficiently complex, or if you want to monitor some set of changes that are not directly associated with a single change in the value of an MBean attribute, use a monitor MBean. The monitor MBeans provide a rich set of tools for comparing data and sending notifications only under highly specific circumstances. However, the monitor periodically polls the observed MBean for changes in attribute value and you are notified of a change only as frequently as the polling interval that you specify.

Best Practices: Commonly Monitored Attributes

The attributes in [Table 5-3](#) provide a general overview of the performance of WebLogic Server. You can monitor these attributes either by creating a listener and registering it directly with the MBeans that contain the attributes or by configuring monitor MBeans.

To create and register a listener or to configure monitor MBeans, you must provide the `WebLogicObjectName` of the MBean that contains the attributes you want to monitor. (See [“Registering a Notification Listener and Filter”](#) on page 5-16 and [“Instantiating the Monitor and Listener”](#) on page 5-28.)

Use the information in [Table 5-3](#) to construct the `WebLogicObjectName` for each MBean. In the table, *domain* refers to the name of the WebLogic Server domain, and *server* refers to the name of the WebLogic Server instance that hosts the MBean you want to monitor.

Table 5-3 Commonly Monitored WebLogic Server Attributes

| MBean and Attribute Names | Description |
|--|---|
| <p>MBean Type: <code>ServerRuntime</code></p> <p>Attribute Name: <code>State</code></p> <p>WebLogicObjectName for the MBean: <code>domain:Location=server,Name=server,Type=ServerRuntime</code></p> <p>For example: <code>examples:Location=ExamplesServer,Name=ExamplesServer,Type=ServerRuntime</code></p> | <p>Indicates whether the server is in an <code>Initializing</code>, <code>Suspended</code>, <code>Running</code>, or <code>ShuttingDown</code> state.</p> |
| <p>MBean Type: <code>ServerRuntime</code></p> <p>Attribute Name: <code>OpenSocketsCurrentCount</code></p> <p>WebLogicObjectName for the MBean: See the previous row in this table.</p> | <p>Use these two attributes together to compare the current activity on the server's listen ports to the total number of requests that can be backlogged on the ports.</p> |
| <p>MBean Type: <code>Server</code></p> <p>Attribute Name: <code>AcceptBacklog</code></p> <p>WebLogicObjectName for the MBean: <code>domain:Name=server,Type=Server</code></p> <p>For example: <code>examples:Name=ExamplesServer,Type=Server</code></p> | <p>Note that the attributes are located in two separate MBeans:</p> <ul style="list-style-type: none"> ■ <code>OpenSocketsCurrentCount</code> is in the <code>ServerRuntime</code> MBean. ■ <code>AcceptBacklog</code> is in the <code>Server</code> configuration MBean. |

Table 5-3 Commonly Monitored WebLogic Server Attributes

| MBean and Attribute Names | Description |
|--|---|
| <p>MBean Type: <code>ExecuteQueueRuntime</code></p> <p>Attribute Name: <code>ExecuteThreadCurrentIdleCount</code></p> <p>WebLogicObjectName for the MBean: <code>domain:Location=server,Name=default,ServerRuntime=server,Type=ExecuteQueueRuntime</code></p> <p>For example: <code>examples:Location=ExamplesServer,Name=default,ServerRuntime=ExamplesServer,Type=ExecuteQueueRuntime</code></p> | <p>Displays the number of threads in a server's default execute queue that are taking up memory space but are not being used to process data.</p> <p>You can create multiple execute queues on a server instance to optimize the performance of critical applications, but the <code>default</code> execute queue is available by default. For more information, refer to "Using Execute Queues to Control Thread Usage."</p> |
| <p>MBean Type: <code>ExecuteQueueRuntime</code></p> <p>Attribute Name: <code>PendingRequestCurrentCount</code></p> <p>WebLogicObjectName for the MBean: See the previous row in this table.</p> | <p>Displays the number of requests waiting in a server's default execute queue.</p> |
| <p>MBean Type: <code>JVMRuntime</code></p> <p>Attribute Name: <code>HeapSizeCurrent</code></p> <p>WebLogicObjectName for the MBean: <code>domain:Location=server,Name=server,ServerRuntime=server,Type=JVMRuntime</code></p> <p>For example: <code>examples:Location=ExamplesServer,Name=ExamplesServer,ServerRuntime=ExamplesServer,Type=JVMRuntime</code></p> | <p>Displays the amount of memory (in bytes) that is currently available in the server's JVM heap.</p> <p>For more information, refer to "Tuning Java Virtual Machines (JVMs)."</p> |

Table 5-3 Commonly Monitored WebLogic Server Attributes

| MBean and Attribute Names | Description |
|--|---|
| <p>MBean Type: <code>JDBCConnectonPoolRuntime</code></p> <p>Attribute Name: <code>ActiveConnectionsCurrentCount</code></p> <p>WebLogicObjectName for the MBean: <code>domain:Location=server,Name=poolName,ServerRuntime=server,Type=JDBCConnectionPoolRuntime</code></p> <p>where <code>poolName</code> is the name that you gave to the connection pool when you created it.</p> <p>For example: <code>examples:Location=ExamplesServer,Name=MyPool-PointBase,ServerRuntime=ExamplesServer,Type=JDBCConnectionPoolRuntime</code></p> | <p>Displays the current number of active connections in a JDBC connection pool.</p> <p>For more information, refer to "Tuning WebLogic Server."</p> |
| <p>MBean Type: <code>JDBCConnectonPoolRuntime</code></p> <p>Attribute Name: <code>ConnectionsHighCount</code></p> <p>WebLogicObjectName for the MBean: See the previous row in this table.</p> | <p>The high water mark of active connections in a JDBC connection pool. The count starts at zero each time the connection pool is instantiated.</p> |

Listening for Notifications from WebLogic Server MBeans: Main Steps

To listen for the notifications that WebLogic Server MBeans emit directly:

1. Determine which notification type you want to listen for. See "[WebLogic Server Notification Types](#)" on page 5-10.
2. Create a listener class in your application. See "[Creating a Notification Listener](#)" on page 5-11.

3. Optionally create a filter class, which specifies the types of notifications that the listener receives from the MBeans. See [“Creating a Notification Filter” on page 5-14](#).
4. Create an additional class that registers your listener and filter with the MBeans whose notifications you want to receive. See [“Registering a Notification Listener and Filter” on page 5-16](#).

WebLogic Server Notification Types

WebLogic Server MBeans implement the `javax.management.NotificationBroadcaster` interface, which enable them to emit different types of notification objects depending on the type of event that occurs:

- When an MBean’s attribute value changes, it emits a `javax.management.AttributeChangeNotification` object.
- When a WebLogic Server resource generates a log message, the server’s `LogBroadcasterRuntimeMBean` emits a notification of type `weblogic.management.WebLogicLogNotification`. For more information about `WebLogicLogNotification`, refer to the [WebLogic Server Javadoc](#).
- When MBeans are registered or unregistered, the WebLogic Server JMX services emit notifications of type `javax.management.MBeanServerNotification`.
- If an MBean attribute is an array, when you invoke the MBean’s `addAttributeName` method to add an element to the array, the MBean emits a `weblogic.management.AttributeAddNotification` object. One example of an MBean that exposes `addAttributeName` methods is `weblogic.management.configuration.XMLRegistryMBean`. For more information, refer to the [WebLogic Server Javadoc](#).
- If an MBean attribute is an array, when you invoke the MBean’s `removeAttributeName` method to remove an element from the array, the MBean emits a `weblogic.management.AttributeRemoveNotification` object.

For more information about the `javax.management` notification types, refer to the JMX 1.0 API documentation, which you can download from <http://jcp.org/aboutJava/communityprocess/final/jsr003/index.html>. The archive that you download includes the API documentation.

For more information about the `weblogic.management` notification types, refer to the Javadoc for [AttributeAddNotification](#) and [AttributeRemoveNotification](#).

Creating a Notification Listener

To create a notification listener:

1. Create a class that implements **one** of the following:
 - For a client that runs within the same JVM as WebLogic Server, implement `javax.management.NotificationListener`.
 - For a client that runs in a remote JVM, implement `weblogic.management.RemoteNotificationListener`.
`RemoteNotificationListener` extends `javax.management.NotificationListener` and `java.rmi.Remote`, making MBean notifications available to external clients via RMI.
2. Within the class, add **one** of the following:
 - For a client that runs within the same JVM as WebLogic Server, add a `NotificationListener.handleNotification(Notification notification, java.lang.Object handback)` **method**.
 - For a client that runs within the same JVM as WebLogic Server, add a `RemoteNotificationListener.handleNotification(Notification notification, java.lang.Object handback)` **method**.

Note: Your implementation of this method should return as soon as possible to avoid blocking its notification broadcaster.
3. To retrieve data from the notification objects that the listener receives, within your `handleNotification` method, invoke `javax.management.Notification` methods on the notification objects.
For example, to retrieve the time stamp associated with the notification, invoke `notification.getTimeStamp()`.

Because all notification types extend `javax.management.Notification`, the following `Notification` methods are available for all notifications:

- `getMessage()`
- `getSequenceNumber()`
- `getTimeStamp()`
- `getType()`
- `getUserData()`

For more information on `Notification` methods, refer to the `javax.management.Notification` Javadoc in the JMX 1.0 API documentation, which you can download from <http://jcp.org/aboutJava/communityprocess/final/jsr003/index.html>. The archive that you download includes the API documentation.

4. Most notification types provide additional methods for retrieving data that is specific to the notification. For example, `WebLogicLogNotification` provides methods for retrieving specific attributes of WebLogic Server log messages, such as `getSeverity()`, which retrieves the severity level that the log message specifies.

If you want to retrieve data that is specific to a notification type (and therefore not retrievable through the standard `javax.management.Notification` methods):

- a. Add logic within the `handleNotification` method to filter through the notifications and select only notifications of a specific type.
- b. Invoke methods that the notification type provides to extract data from the notification object.

For example:

```
if(notification instanceof MonitorNotification) {
    MonitorNotification monitorNotification =
(MonitorNotification)
                                                                    notific
ation;
    System.out.println("This notification is a
MonitorNotification");
    System.out.println("Observed Attribute: " +
                                                                    monitorNotification.getObservedAttrib
```

```
    ute() );  
}
```

In addition to the previous steps, consider the following while creating your `NotificationListener` class:

- Unless you create and use a notification filter, your listener receives all notifications (of all notification types) from the MBeans with which it is registered.

Instead of using one listener for all possible notifications that an MBean emits, the best practice is to use a combination of filters and listeners. While having multiple listeners adds to the amount of time for initializing the JVM, the trade-off is ease of code maintenance.

- If your WebLogic Server environment contains multiple instances of MBean types that you want to monitor, you can create one notification listener and then create as many registration classes as MBean instances that you want to monitor.

For example, if your WebLogic Server domain contains three JDBC connection pools, you can create one listener class that listens for `AttributeChangeNotifications`. Then, you create three registration classes. Each registration class registers the listener with a specific instance of a `JDBCConnectionPoolRuntime` MBean.

- While the `handleNotification` method signature includes an argument for a handback object, your listener does not need to retrieve data from or otherwise manipulate the handback object. It is an opaque object that helps the listener to associate information regarding the MBean emitter.

The following example creates a remote listener. Then the listener receives a `AttributeChangeNotification` object, it uses `AttributeChangeNotification` methods to retrieve the name of the attribute with a changed value, and the old and new values.

Listing 5-1 Notification Listener

```
import javax.management.Notification;  
import javax.management.NotificationFilter;  
import javax.management.NotificationListener;  
import weblogic.management.RemoteNotificationListener;  
import javax.management.AttributeChangeNotification;  
  
public class MyListener implements RemoteNotificationListener {
```

```
public void handleNotification(Notification notification, Object obj) {
    if(notification instanceof AttributeChangeNotification) {
        AttributeChangeNotification attributeChange =
            (AttributeChangeNotification) notification;
        System.out.println("This notification is an
            AttributeChangeNotification");
        System.out.println("Observed Attribute: " +
            attributeChange.getAttributeName() );
        System.out.println("Old Value: " + attributeChange.getOldValue() );
        System.out.println("New Value: " + attributeChange.getNewValue() );
    }
}
```

Creating a Notification Filter

To create and register a filter:

1. Create a serializable class that implements `javax.management.NotificationFilter`.

Optionally import the `javax.management.NotificationFilterSupport` class, which provides utility methods for filtering notifications. For more information about using these methods, refer to the JMX 1.0 API documentation, which you can download from <http://jcp.org/aboutJava/communityprocess/final/jsr003/index.html>. The archive that you download includes the API documentation.

The filter needs to be serializable only if it is used in a remote notification listener. A class that is used with RMI must be serializable so it can be deconstructed and reconstructed in remote JVMs.

2. Use the `isNotificationEnabled(Notification notification)` method to indicate whether the serializable object returns a true value when a set of conditions are satisfied.

If the boolean returns true, then the filter forwards the notification to the listener with which the filter is registered.

3. (Optional) You can include code that retrieves data from notifications and carries out actions based on the data in the notification. For example, your filter can use `javax.management.AttributeChangeNotification` methods to view the new value of a specific attribute. If the value is over a threshold that you specify, you can use JavaMail API to send e-mail to an administrator.

[Listing 5-2](#) provides an example `NotificationFilter` that forwards only notifications of type `AttributeChangeNotification`.

Listing 5-2 Example Notification Filter

```
import javax.management.Notification;
import javax.management.NotificationFilter;
import javax.management.AttributeChangeNotification;

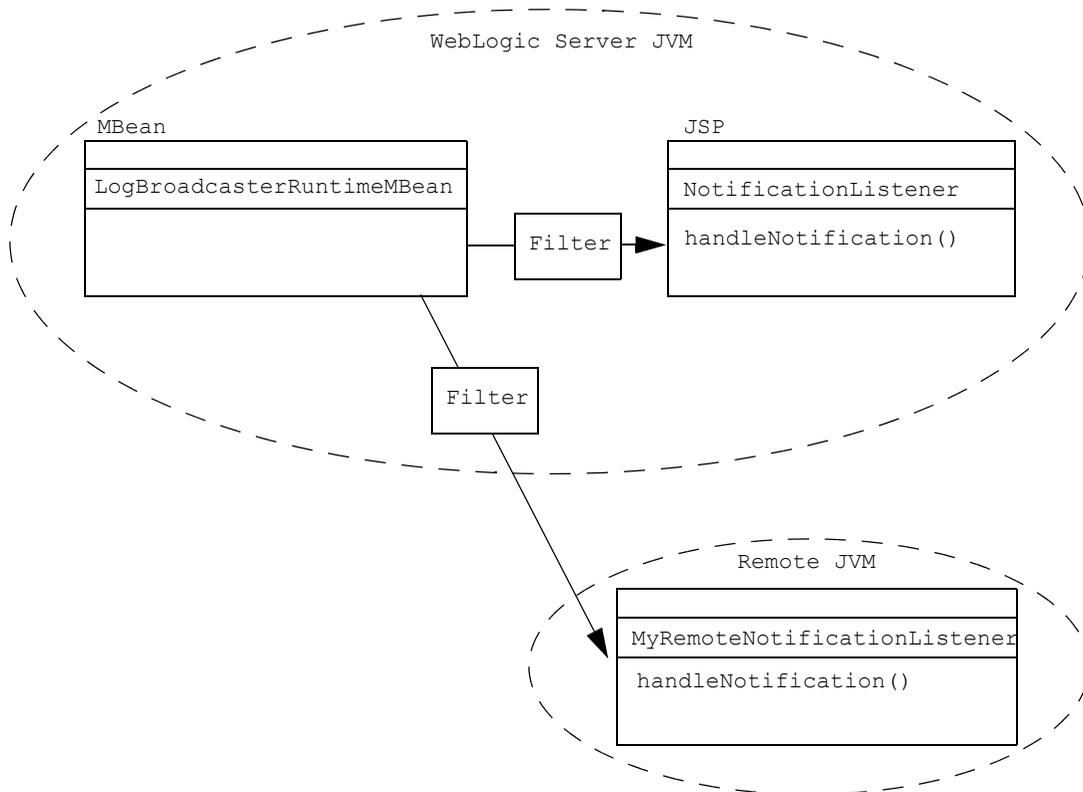
public class MyHiCountFilter implements NotificationFilter,
    java.io.Serializable {

    public boolean isNotificationEnabled(Notification notification) {
        if (!(notification instanceof AttributeChangeNotification)) {
            return false;
        }
        AttributeChangeNotification acn =
            (AttributeChangeNotification)notification;
        acn.getAttributeName().equals("ActiveConnectionsHighCount"); {
            return true;
        }
    }
}
```

Adding Filter Classes to the Server Classpath

If you create a filter for a listener that runs in a remote JVM, you can add the filter's classes to the classpath of the server instance from which you are listening for notifications. Although the listener runs in the remote JVM, adding the filter's classes to the server's classpath minimizes the transportation of serialized data between the filter and the listener. (See [Figure 5-4](#).)

Figure 5-4 Filters Can Run on WebLogic Server



Registering a Notification Listener and Filter

After you implement a notification listener class and optional filter class, you create an additional class that registers your listener and filter with an MBean instance. You must create one registration class for each MBean instance that you want to monitor.

To register a notification listener and filter:

1. Create a class that retrieves the `MBeanHome` interface and then uses `MBeanHome` to retrieve the `MBeanServer` interface.

If you want to register a listener and filter with an Administration MBean, you must retrieve the `Administration MBeanHome`, which resides only on the Administration Server. If you want to register with a Local Configuration MBean or a Runtime MBean, you must retrieve the `Local MBeanHome` for the server instance that hosts the MBean.

2. Instantiates the listener class and filter class that you created.
3. Constructs the `WebLogicObjectName` of the MBean with which you want to register.
4. Registers the listener and filter by passing the `WebLogicObjectName`, listener class, and filter class to the `addNotificationListener()` method of the `MBeanServer` interface.

While [Figure 5-1](#) illustrates registering a listener and filter directly with an MBean (which you can do by calling the MBean's `addNotificationListener()` method), in practice it is preferable to use the `addNotificationListener()` method of the `MBeanServer` interface, which saves the trouble of looking up a particular MBean simply for registration purposes.

The following example is a registration class that runs in a remote JVM. If the class ran within the same JVM as a WebLogic Server instance, the code for retrieving the `MBeanHome` interface would be simpler. For more information, refer to [“Accessing an MBeanHome Interface” on page 2-4](#).

The example class registers the listener from [Listing 5-1](#) and filter from [Listing 5-2](#) with the `Server Administration` MBean for a server instance named `Server1`. In the example, `weblogic` is a user who has permission to view and modify MBean attributes. For information about permissions to view and modify MBeans, refer to [“Protecting System Administration Operations”](#) in *WebLogic Server Administration Guide*.

The example class also includes some code to keep the class active until it receives a notification. Usually this code is not necessary because a listener class runs in the context of some larger application that is responsible for invoking the class and keeping it active. It is included here so you can easily compile and see the example working.

Listing 5-3 Registering a Listener for an Administration MBean

```
import java.util.Set;
import java.util.Iterator;
import java.rmi.RemoteException;
import javax.naming.Context;
import javax.management.ObjectName;
import javax.management.Notification;

import weblogic.jndi.Environment;
import weblogic.management.MBeanHome;
import weblogic.management.WebLogicMBean;
import weblogic.management.WebLogicObjectName;
import weblogic.management.RemoteMBeanServer;
import weblogic.management.configuration.ServerMBean;

public class listener {

    public static void main(String[] args) {

        MBeanHome home = null;
        RemoteMBeanServer rmbs = null;

        //domain variables
        String url = "t3://localhost:7001";
        String serverName = "Server1";
        String username = "weblogic";
        String password = "weblogic";

        //Using MBeanHome to get MBeanServer.
        try {
            Environment env = new Environment();
            env.setProviderUrl(url);
            env.setSecurityPrincipal(username);
            env.setSecurityCredentials(password);
            Context ctx = env.getInitialContext();

            //Getting the Administration MBeanHome.
            home = (MBeanHome) ctx.lookup(MBeanHome.ADMIN_JNDI_NAME);
            System.out.println("Got the Admin MBeanHome: " + home );
            rmbs = home.getMBeanServer();
        } catch (Exception e) {
            System.out.println("Caught exception: " + e);
        }

        try {
            //Instantiating your listener class.
            MyListener listener = new MyListener();
            MyFilter filter = new MyFilter();
        }
    }
}
```

```
//Constructing the WebLogicObjectName of the MBean that you want
//to listen to.
WebLogicObjectName mbeanName = new WebLogicObjectName(serverName,
    "Server",home.getDomainName());
System.out.println("Created WebLogicObjectName: " + mbeanName);

//Passing the name of the MBean and your listener class to the
//addNotificationListener method of MBeanServer.
rmbs.addNotificationListener(mbeanName, listener, filter, null);
System.out.println("\n[myListener]: Listener registered ...");

//Keeping the remote client active.
System.out.println("pausing.....");
System.in.read();
} catch(Exception e) {
    System.out.println("Exception: " + e);
}
}
```

Listening for Configuration Auditing Messages: Main Steps

By default, the Administration Server emits a log message when a user changes the configuration or invokes management operations on any resource within a domain. For example, if a user disables SSL on a Managed Server in a domain, the Administration Server emits a log message. These messages provide an audit trail of changes within a domain's configuration (configuration auditing). See "[Configuration Auditing](#)" in the *Administration Guide*.

To create and use a JMX listener and filter that respond to configuration auditing messages:

1. Create and compile a notification listener that extracts information from WebLogic Server log messages.
See "[Notification Listener for Configuration Auditing Messages](#)" on page 5-20.
2. Create and compile a notification filter that selects only configuration auditing messages.
See "[Notification Filter for Configuration Auditing Messages](#)" on page 5-21.

3. Create and compile a class that registers the listener and filter with the Administration Server's `LogBroadcasterRuntime` MBean. This is the MBean that a WebLogic Server instance uses to broadcast its log messages as JMX notifications.

See [“Registration Class for Configuration Auditing Messages”](#) on page 5-22.

4. Add the notification filter to the classpath for the Administration Server.

If the notification listener runs within the Administration Server's JVM (for example, if it runs as a startup class), add the notification listener and registration class to the Administration Server's classpath as well.

5. Invoke the registration class or configure it as a startup class for the Administration Server.

See [“Configuring Startup and Shutdown Classes”](#) in the Administration Guide.

Notification Listener for Configuration Auditing Messages

Like the notification listener in [Listing 5-1](#), the listener in [Listing 5-1](#) implements `RemoteNotificationListener` and its `handleNotification` method.

Because all configuration auditing messages are of type `WebLogicLogNotification`, the listener in [Listing 5-1](#) imports the `WebLogicLogNotification` interface and uses its methods to retrieve information within each configuration auditing message.

Table 5-1 Notification Listener for Configuration Auditing Messages

```
import javax.management.Notification;
import javax.management.NotificationListener;
import weblogic.management.RemoteNotificationListener;
import weblogic.management.logging.WebLogicLogNotification;

public class ConfigAuditListener implements RemoteNotificationListener {
    public void handleNotification(Notification notification, Object obj) {
        WebLogicLogNotification changeNotification =
            (WebLogicLogNotification) notification;
        System.out.println("A user has attempted to change the configuration
            of a WebLogic Server domain.");
        System.out.println("Admin Server Name: " +
            changeNotification.getServername() );
        System.out.println("Time of attempted change:" +
            changeNotification.getTimeStamp() );
        System.out.println("Message details:" +
```

```
        changeNotification.getMessage() );
System.out.println("Message ID string:" +
        changeNotification.getMessageId() );
    }
}
```

Notification Filter for Configuration Auditing Messages

Without a notification filter, the listener in [Listing 5-1](#) would print the Server Name, Timestamp, and Message Text for all messages that the Administration Server broadcast.

To forward only the configuration auditing message that indicates a resource has been modified, the filter in [Listing 5-4](#) uses the `WebLogicLogNotification.getMessageId` method to retrieve the message ID of all incoming log notifications.

The resource-change configuration auditing message is identified by the message ID 159904 (see "[Configuration Auditing](#)" in the *Administration Guide*). If the message ID value in an incoming log notification matches the configuration auditing message ID, the filter evaluates as `true` and forwards the message to its registered listener.

Listing 5-4 Notification Filter for Configuration Auditing Messages

```
import javax.management.Notification;
import javax.management.NotificationFilter;
import weblogic.management.logging.WebLogicLogNotification;

public class ConfigAuditFilter implements NotificationFilter ,
        java.io.Serializable{
    int configChangedId = 159904;

    public boolean isNotificationEnabled(Notification notification) {
        if (!(notification instanceof WebLogicLogNotification)) {
            return false;
        }

        WebLogicLogNotification wln =
            (WebLogicLogNotification)notification;
        int messageId = wln.getMessageId();
        if (configChangedId == messageId) {
            return true;
        }
    }
}
```

```
        } else {  
            return false;  
        }  
    }  
}
```

Registration Class for Configuration Auditing Messages

The class in [Listing 5-5](#) registers the notification listener and filter with the `LogBroadcasterRuntime` MBean of the Administration Server. This MBean is a singleton in each instance of WebLogic Server and is always named `TheLogBroadcaster`.

Listing 5-5 Registration Class for Configuration Auditing Messages

```
import java.util.Set;  
import java.util.Iterator;  
import java.rmi.RemoteException;  
import javax.naming.Context;  
import javax.management.ObjectName;  
import javax.management.Notification;  
import weblogic.jndi.Environment;  
import weblogic.management.MBeanHome;  
import weblogic.management.WebLogicMBean;  
import weblogic.management.WebLogicObjectName;  
import weblogic.management.RemoteMBeanServer;  
import weblogic.management.configuration.ServerMBean;  
  
public class ListenRegistration {  
    public static void main(String[] args) {  
        MBeanHome home = null;  
        RemoteMBeanServer rmbs = null;  
  
        //domain variables  
        String url = "t3://localhost:7001";  
        String serverName = "examplesServer";  
        String username = "weblogic";  
        String password = "weblogic";  
  
        //Using MBeanHome to get MBeanServer.  
        try {  
            Environment env = new Environment();  
            env.setProviderUrl(url);  
            env.setSecurityPrincipal(username);
```

Listening for Notifications from WebLogic Server MBeans: Main Steps

```
env.setSecurityCredentials(password);
Context ctx = env.getInitialContext();

//Getting the Administration MBeanHome.
home = (MBeanHome) ctx.lookup(MBeanHome.ADMIN_JNDI_NAME);
System.out.println("Got the Admin MBeanHome: " + home );
rmbs = home.getMBeanServer();

} catch (Exception e) {
    System.out.println("Caught exception: " + e);
}

try {
    //Instantiating your listener class.
    ConfigAuditListener listener = new ConfigAuditListener();
    ConfigAuditFilter filter = new ConfigAuditFilter();

    //Constructing the WebLogicObjectName of the MBean that you want
    //to listen to.

    WebLogicObjectName mbeanName = new WebLogicObjectName(
        "TheLogBroadcaster",
        "LogBroadcasterRuntime",
        home.getDomainName(),
        serverName );
    System.out.println("Created WebLogicObjectName: " + mbeanName);

    //Passing the name of the MBean and your listener class to the
    //addNotificationListener method of MBeanServer.
    rmbs.addNotificationListener(mbeanName, listener, filter, null);
    System.out.println("\n[myListener]: Listener registered ...");

    //Keeping the remote client active.
    System.out.println("pausing.....");
    System.in.read();
} catch (Exception e) {
    System.out.println("Exception: " + e);
}

}

}
```

Using Monitor MBeans to Observe Changes: Main Steps

To configure and use monitor MBeans:

1. Choose a monitor MBean type that matches the type of data you want to observe. [“Choosing a Monitor MBean Type” on page 5-24](#)
2. Create a listener class that can listen for notifications from monitor MBeans. See [“Creating a Notification Listener for a Monitor MBean” on page 5-27](#).
3. Create a class that configures a monitor MBean, registers your listener class with the monitor MBean, and then registers the monitor MBean with an observed MBean. [“Instantiating the Monitor and Listener” on page 5-28](#)

Choosing a Monitor MBean Type

WebLogic Server provides monitor MBeans that are specialized to observe changes in specific data types. You must configure and instantiate the type of monitor MBean that matches the type of the object that an MBean returns for an attribute value. For example, a monitor MBean based on the `StringMonitor` type can observe an attribute that is declared as an `Object` as long as actual values of the attributes are `String` instances, as determined by the `instanceof` operator.

To choose a monitor type:

1. Determine the type of object that is returned by the MBean attribute that you want to observe by doing any of the following:
 - Refer to the WebLogic Server Javadoc.
 - Use the `weblogic.Admin GET` command, which provides information about the MBean that you specify. For more information, refer to ["MBean Management Command Reference"](#) in the *WebLogic Server Administration Guide*.
 - Use the `javap` command on the MBean you are monitoring. The `javap` command is a standard Java utility that disassembles a class file.

2. Choose a monitor type from the following table.

Table 5-2 Monitor MBeans and Observed Object Types

| A Monitor MBean of This Type | Observes This Object Type |
|------------------------------|---|
| CounterMonitor | Integer |
| GaugeMonitor | Integer or floating-point (Byte, Integer, Short, Long, Float, Double) |
| StringMonitor | String |

For more information about monitor types, refer to the JMX 1.0 specification, which you can download from <http://jcp.org/aboutJava/communityprocess/final/jsr003/index.html>. The archive that you download includes the API documentation.

Monitor Notification Types

Each type of monitor MBean emits specific types of `javax.management.monitor.MonitorNotification` notifications. For any given notification, you can use the `MonitorNotification.getType()` method to determine its type.

The following table describes the type of notifications that monitor MBeans emit.

Table 5-3 Monitor MBeans and MonitorNotification Types

| A Monitor MBean of This Type | Emits This MonitorNotification Type |
|------------------------------|--|
| CounterMonitor | A counter monitor emits a <code>jmx.monitor.counter.threshold</code> when the value of the counter reaches or exceeds a threshold known as the comparison level. |

Table 5-3 Monitor MBeans and MonitorNotification Types

| A Monitor MBean of This Type | Emits This MonitorNotification Type |
|------------------------------|---|
| GaugeMonitor | <ul style="list-style-type: none">■ If the observed attribute value is increasing and becomes equal to or greater than the high threshold value, the monitor emits a notification type of <code>jmx.monitor.gauge.high</code>. Subsequent crossings of the high threshold value do not cause further notifications unless the attribute value becomes equal to or less than the low threshold value.■ If the observed attribute value is decreasing and becomes equal to or less than the low threshold value, the monitor emits a notification type of <code>jmx.monitor.gauge.low</code>. Subsequent crossings of the low threshold value do not cause further notifications unless the attribute value becomes equal to or greater than the high threshold value. |
| StringMonitor | <ul style="list-style-type: none">■ If the observed attribute value matches the string to compare value, the monitor emits a notification type of <code>jmx.monitor.string.matches</code>. Subsequent matches of the string to compare values do not cause further notifications unless the attribute value differs from the string to compare value.■ If the attribute value differs from the string to compare value, the monitor emits a notification type of <code>jmx.monitor.string.differs</code>. Subsequent differences from the string to compare value do not cause further notifications unless the attribute value matches the string to compare value. |

Error Notification Types

All monitors can emit the following notification types to indicate error cases:

- `jmx.monitor.error.mbean`, which indicates that the observed MBean is not registered in the MBean Server. The observed object name is provided in the notification.
- `jmx.monitor.error.attribute`, which indicates that the observed attribute does not exist in the observed object. The observed object name and observed attribute name are provided in the notification.
- `jmx.monitor.error.type`, which indicates that the object instance of the observed attribute value is `null` or not of the appropriate type for the given monitor. The observed object name and observed attribute name are provided in the notification.

- `jmx.monitor.error.runtime`, which contains exceptions that are thrown while trying to get the value of the observed attribute (for reasons other than the cases described above).

The counter and the gauge monitors can also emit the following `jmx.monitor.error.threshold` notification type under the following circumstances:

- For a counter monitor, when the threshold, the offset, or the modulus is not of the same type as the observed counter attribute.
- For a gauge monitor, when the low threshold or high threshold is not of the same type as the observed gauge attribute.

Creating a Notification Listener for a Monitor MBean

As any other MBean, monitor MBeans emit notifications by implementing `javax.management.NotificationBroadcaster`. To create a listener for notifications from a monitor MBean, create a class that does the following:

1. Implements `NotificationBroadcaster` or `weblogic.management.RemoteNotificationListener`.
2. Includes the `NotificationListener.handleNotification()` or the `RemoteNotificationListener.handleNotification()` method.

You can register the same notification listener with instances of `LogBroadcasterMBean`, monitor MBeans, or any other MBean.

The example below creates a listener object for an application that runs in a JVM outside the WebLogic Server JVM. It includes logic that outputs additional messages when it receives notifications from monitor MBeans. You could further refine the logic so that listener responds differently to the different types of monitor notifications described in [“Monitor Notification Types” on page 5-25](#).

Listing 5-6 Listener for Monitor Notifications

```
import java.rmi.Remote;
import javax.management.Notification;
```

5 Using WebLogic Server MBean Notifications and Monitors

```
import javax.management.NotificationListener;
import javax.management.monitor.MonitorNotification;

import weblogic.management.RemoteNotificationListener;
import weblogic.management.MBeanHome;

public class CounterListener implements RemoteNotificationListener {

    public void handleNotification(Notification notification ,Object obj) {
        System.out.println("\n\n Notification Received ...");
        System.out.println("Type=" + notification.getType() );
        System.out.println("SequenceNumber=" +
            notification.getSequenceNumber());
        System.out.println("Source=" + notification.getSource());
        System.out.println("Timestamp=" + notification.getTimeStamp() + "\n" );
        if(notification instanceof MonitorNotification) {
            MonitorNotification monitorNotification = (MonitorNotification)
                notification;
            System.out.println("This notification is a MonitorNotification");
            System.out.println("Observed Attribute: " +
                monitorNotification.getObservedAttribute() );
            System.out.println("Observed Object: " +
                monitorNotification.getObservedObject() );
            System.out.println("Trigger value: " +
                monitorNotification.getTrigger() );
        }
    }
}
```

Instantiating the Monitor and Listener

The steps you take to register a monitor MBean with an observed MBean differ depending on whether you are registering the monitor MBean on a single server instance or on multiple server instances in a domain.

To register a monitor MBean on a single server instance:

1. Instantiate and configure the monitor MBean.
2. Retrieve the `MBeanHome` interface of the server instance that hosts the observed MBean.
3. Register the monitor MBean with the observed MBean.

To register a monitor MBean on multiple server instances:

1. Instantiate and configure the monitor MBean.
2. Retrieve the `MBeanHome` interface of **each server instance** that hosts instances of the observed MBean.
3. For each server instance, register the monitor MBean with the observed MBean instance.

The following sections provide examples for both tasks:

- [“Example: Monitoring an MBean on a Single Server” on page 5-29](#)
- [“Example: Monitoring Instances of an MBean on Multiple Servers” on page 5-32](#)

Example: Monitoring an MBean on a Single Server

The following example creates a monitor for the `ServiceRequestTotalCount` attribute of the `ExecuteQueueRuntimeMBean`, which returns the number of requests that have been processed by the corresponding execution queue. WebLogic Server uses execute queues to optimize the performance of critical applications. For more information, refer to [“Using Execute Queues to Control Thread Usage.”](#)

To create a counter monitor for an `ExecuteQueueRuntimeMBean` on a single server instance, the example class in [Listing 5-7](#):

1. Instantiates a `javax.management.monitor.CounterMonitor` object.
2. Configures the monitor object by doing the following:
 - a. Assign the JMX object name for the **monitor object** to a variable.

[Listing 5-7](#) uses `WebLogicObjectName()`, but you can use `javax.management.ObjectName` for the monitor object. The object name must be unique throughout the entire WebLogic Server domain, and it must follow the JMX conventions:

```
domain name:Name=name,Type=type[,attr=value]...
```

- b. Uses `WebLogicObjectName()` to assigns the JMX object name for the **observed MBean** to a variable.

If the observed MBean is a WebLogic Server MBean, you must use `WebLogicObjectName()` instead of `javax.management.ObjectName`. You can also use `MBeanHome.getMBeansByType()` or other WebLogic Server APIs to get the name of the observed MBean object. For examples of different methods of retrieving MBeans, refer to [“Accessing WebLogic Server MBeans” on page 2-1](#).

- c. Sets values for the monitor’s threshold parameters. The set of available parameters varies, depending on whether you are instantiating a `CounterMonitor`, `GaugeMonitor`, or `StringMonitor`.
- d. Configures the monitor object using the monitor’s APIs.

For more information about the parameters that you pass to configure monitors, refer to:

- [“Configuring CounterMonitor Objects” on page 5-36](#)
 - [“Configuring GaugeMonitor Objects” on page 5-38](#)
 - [“Configuring StringMonitor Objects” on page 5-39](#).
3. Instantiates the listener object that you created in [“Creating a Notification Listener for a Monitor MBean” on page 5-27](#).
 4. Registers the listener object using the monitor’s `addNotificationListener()` method.
 5. (This step is needed only if your monitor class runs in a JVM that is outside the WebLogic Server JVM.) Pre-registers a reference to the MBean Server in the remote JVM by doing the following:
 - a. Retrieve the `MBeanServer` interface using the `AdministrationMBeanHome` interface.
 - b. Use the monitor’s `preRegister()` method.
 6. Starts the monitor using the monitor’s `start()` method.

In the example, `weblogic` is a user who has permission to view and modify MBean attributes. For information about permissions to view and modify MBeans, refer to [“Protecting System Administration Operations”](#) in *WebLogic Server Administration Guide*.

Listing 5-7 Instantiating the Monitor and Listener

```
import javax.management.monitor.CounterMonitor;
import javax.management.ObjectName;
import javax.naming.Context;

import weblogic.jndi.Environment;
import weblogic.management.MBeanHome;
import weblogic.management.WebLogicMBean;
import weblogic.management.WebLogicObjectName;
import weblogic.management.RemoteMBeanServer;
import weblogic.management.configuration.ServerMBean;

public class clientMonitor {

    // The name of the WebLogic domain, please change this to match the //
    // name of your installation specific domain name //
    private static String weblogicDomain = "mydomain";

    // The name of the WebLogic server, please change this to match the //
    // name of your installation specific server name //
    private static String weblogicServer = "myserver";

    public static void main (String Args[]) {

        try {
            //Instantiate a CounterMonitor
            CounterMonitor monitor = new CounterMonitor();

            // construct the objectName for your CounterMonitor object
            WebLogicObjectName monitorObjectName = new
                WebLogicObjectName("MyCounter",
                    "CounterMonitor",weblogicDomain);

            // Construct the objectName for the parent MBean
            WebLogicObjectName pObjectName = new
                WebLogicObjectName(weblogicServer,
                    "ServerRuntime",weblogicDomain);

            // Construct the objectName for the observed MBean
            WebLogicObjectName qObjectName = new
                WebLogicObjectName("default",
                    "ExecuteQueueRuntime",weblogicDomain,
                    weblogicServer, pObjectName);

            // Define variables to be used when configuring your CounterMonitor
            // object.
            Integer threshold = new Integer(10);
            Integer offset = new Integer(1);
```

5 Using WebLogic Server MBean Notifications and Monitors

```
//Configure your monitor object using the CounterMonitor APIs
monitor.setThreshold(threshold);
monitor.setNotify(true);
monitor.setOffset(offset);
monitor.setObservedObject(qObjectName);
monitor.setObservedAttribute("ServicedRequestTotalCount");

//Instantiate and register your listener with the monitor
CounterListener listener = new CounterListener();
monitor.addNotificationListener(listener, null, null);

//Use the Administration MBeanHome API to get the MBeanServer
//interface this is needed when you are registering a monitor from
// the client side.
String url = "t3://localhost:7001"; //URL of the Admin Server
String username = "weblogic";
String password = "weblogic";
MBeanHome home = null;
Environment env = new Environment();
env.setProviderUrl(url);
env.setSecurityPrincipal(username);
env.setSecurityCredentials(password);
Context ctx = env.getInitialContext();
home = (MBeanHome) ctx.lookup(MBeanHome.ADMIN_JNDI_NAME);
RemoteMBeanServer rmbs = home.getMBeanServer();
monitor.preRegister(rmbs, monitorObjectName);

//start the monitor
monitor.start();
} catch (Exception e) { e.printStackTrace(); }
}
```

Example: Monitoring Instances of an MBean on Multiple Servers

A WebLogic Server domain maintains a set of MBean instances for each server instance. For example, each server instance hosts its own `ServerRuntimeMBean`, `LogMBean`, and `ExecuteQueueRuntimeMBean`.

Some MBeans are instantiated only if a server instance hosts a specific service. For example, if you use the Java Messaging Service (JMS), each server instance that is defined as a JMS destination hosts its own `JMSDestinationRuntimeMBean`. For information about JMS destinations, refer to “[Using Distributed Destinations](#)” in *Programming WebLogic JMS*.

To monitor instances of a `JMSDestinationRuntimeMBean` on each server instance in a domain, the example in [Listing 5-8](#):

1. Retrieves the domain's Administration `MBeanHome`.
2. Invokes `MBeanHome.getMBeansByType` to retrieve all instances of `JMSDestinationRuntimeMBean` in the domain.
3. For each `JMSDestinationRuntimeMBean`, the class instantiates and configures a `GaugeMonitor` object by doing the following:
 - a. Instantiates a `GaugeMonitor` object using the default constructor of `javax.management.monitor.GaugeMonitor`.
 - b. Configures the `GaugeMonitor` object.

To provide the value for the `GaugeMonitor.setObservedObject` method, the class casts the `JMSDestinationRuntimeMBean` as a `WebLogicMBean`. Then it invokes `WebLogicMBean.getObjectName`.
4. For each `GaugeMonitor` object, the class instantiates a notification listener and registers the listener with the monitor by invoking the `GaugeMonitor.addNotificationListener` method.

To see an example notification listener, refer to [“Creating a Notification Listener for a Monitor MBean”](#) on page 5-27.
5. For each `GaugeMonitor` object, the class registers a reference to the MBean Server in the host server's JVM by doing the following:
 - a. Invokes the `Context.lookup(MBeanHome.JNDI_NAME, serverName)` method to retrieve the `MBeanServer` interface using the Administration `MBeanHome` interface.

To provide the `serverName` value in the `lookup` method, the class refers to the `WebLogicMBean` cast of the `JMSDestinationRuntimeMBean` and invokes its `WebLogicMBean.getObjectName().getLocation()` method.
 - b. Invokes the monitor's `preRegister()` method.
6. Starts the monitor using the monitor's `start()` method.
7. Includes code that keep the class active. Usually this code is not necessary because a monitor runs in the context of some larger application that is responsible for invoking the class and keeping it active. It is included here so you can easily compile and see the example working.

8. Includes code that stops the monitor (and thereby ends the thread that the JVM assigned to the monitor).

In the example, `weblogic` is a user who has permission to view and modify MBean attributes. For information about permissions to view and modify MBeans, refer to "[Security Roles](#)" in the *Securing WebLogic Resources* guide.

Listing 5-8 Instantiating a Gauge Monitor on Multiple Server Instances

```
import java.util.Set;
import java.util.Iterator;
import java.util.List;
import java.util.ArrayList;
import java.util.Collections;
import javax.naming.Context;

import javax.management.monitor.GaugeMonitor;
import javax.management.ObjectName;
import weblogic.jndi.Environment;
import weblogic.management.MBeanHome;
import weblogic.management.WebLogicMBean;
import weblogic.management.RemoteMBeanServer;
import weblogic.management.runtime.JMSDestinationRuntimeMBean;
import weblogic.management.WebLogicObjectName;

public class GaugeMonitorClient {

    public static void main (String Args[]) throws Exception {
        //url of the Administration Server
        String url = "t3://localhost:7001";
        String username = "weblogic";
        String password = "weblogic";
        String domain = "examples";

        try {
            //retrieve the Administration MBeanHome
            Environment env = new Environment();
            env.setProviderUrl(url);
            env.setSecurityPrincipal(username);
            env.setSecurityCredentials(password);
            Context ctx = env.getInitialContext();
            MBeanHome home = (MBeanHome)
                ctx.lookup(MBeanHome.ADMIN_JNDI_NAME);

            //retrieve all JMSDestinationRuntimeMBean instances in
            //the domain
```

Using Monitor MBeans to Observe Changes: Main Steps

```
Set mbeanSet =
    home.getMBeansByType("JMSDestinationRuntime");

System.out.println("Retrieved the following mbeans");
Iterator iter = mbeanSet.iterator();
while (iter.hasNext()){
    WebLogicMBean bean = (WebLogicMBean) iter.next();
    System.out.println("Name = "+bean.getName());
    System.out.println("WebLogicObjectName =
        "+bean.getObjectName()+"\n");
}

List list = Collections.synchronizedList(new ArrayList());
Iterator it = mbeanSet.iterator();
int i = 0;
while (it.hasNext()) {
    //instantiate a Gauge monitor
    GaugeMonitor monitor = new GaugeMonitor();
    //configure the Gauge monitor
    monitor.setThresholds(new Long("30"), new Long("4"));
    monitor.setNotifyHigh(true);
    monitor.setNotifyLow(true);
    WebLogicMBean bean = (WebLogicMBean) it.next();
    ObjectName myON = bean.getObjectName();
    monitor.setObservedObject(myON);
    monitor.setObservedAttribute("MessagesCurrentCount");

    //instantiate and register a notification listener
    MyNotificationListener listener = new MyNotificationListener();
    monitor.addNotificationListener(listener,null,null);
    //pre-registering and starting the monitor
    MBeanHome localhome = (MBeanHome)
        ctx.lookup(MBeanHome.JNDI_NAME
            +"."+bean.getObjectName().getLocation());
    RemoteMBeanServer rmbs = localhome.getMBeanServer();
    WebLogicObjectName monitorObjectName = new WebLogicObjectName
        ("myGaugeMonitor" + (++i), "GaugeMonitor", domain,
            bean.getObjectName().getLocation());
    monitor.preRegister(rmbs, monitorObjectName);
    monitor.start();
    System.out.println("Monitor waiting on event notification.");
    list.add(monitor);
    myON = null;
}

//Keeping the monitor active.
System.out.println("pausing.....");
System.in.read();
```

```
//stopping each monitor.
Iterator deregisterList = list.iterator();
while (deregisterList.hasNext()) {
    GaugeMonitor gauge = (GaugeMonitor) deregisterList.next();
    System.out.println("deregistering...");
    gauge.preDeregister();
}
return;
}
catch (Exception e){
    e.printStackTrace();
}
}
}
```

Configuring CounterMonitor Objects

`CounterMonitor` objects observe changes in MBean attributes that are expressed as integers. The following list describes groups of `CounterMonitor` operations that you use to achieve typical configurations of a `CounterMonitor` instance:

- Sends a notification when the observed attribute exceeds the threshold.

```
setThreshold(int threshold);
setNotify(true);
setObservedObject(ObjectName);
setObservedAttribute("AttributeName");
```

- Sends a notification when the observed attribute exceeds the threshold. Then it increases the threshold by the offset value. Each time the observed attribute exceeds the new threshold, the threshold is increased by the offset value. For example, if you set `Threshold` to 1000 and `Offset` to 2000, when the observed attribute exceeds 1000, the `CounterMonitor` object sends a notification and increases the threshold to 3000. When the observed attribute exceeds 3000, the `CounterMonitor` object sends a notification and increases the threshold again to 5000.

```
setThreshold(int threshold);
setNotify(true);
setOffset(int offset);
setObservedObject(ObjectName);
setObservedAttribute("AttributeName");
```

- Sends a notification when the observed attribute exceeds the threshold, and increases the threshold by the offset value. When the threshold reaches the value specified by the `setModulus` method, the threshold is returned to the value that was specified through the latest call to the monitor's `setThreshold` method, before any offsets were applied. For example, if the original `Threshold` is set to 1000 and the `Modulus` is set to 5000, when the `Threshold` exceeds 5000, the monitor sends a notification and resets the `Threshold` to 1000.

```
setThreshold(int threshold);
setNotify(true);
setOffset(int offset);
setModulus(int modulus);
setObservedObject(ObjectName);
setObservedAttribute("AttributeName");
```

- Sends a notification when the difference between two consecutive observations exceeds the threshold. For example, the `Threshold` is 20 and the monitor observes an attribute value of 2. If the next observation is greater than 22, then the monitor sends a notification. However, if the value is 10 at the next observation, and 25 at the following observation, then the monitor does not send a notification because the value has not changed by more than 20 for any two consecutive observations.

```
setThreshold(int threshold);
setNotify(true);
setDifferenceMode(true);
setObservedObject(ObjectName);
setObservedAttribute("AttributeName");
```

- Sends a notification when the difference between two consecutive observations exceeds the threshold, and increases the threshold by the offset value. When the threshold reaches the value specified by the `setModulus` method, the threshold is returned to the value that was specified through the latest call to the monitor's `setThreshold` method, before any offsets were applied.

```
setThreshold(int threshold);
setNotify(true);
setOffset(int offset);
setModulus(int modulus);
setDifferenceMode(true);
setObservedObject(ObjectName);
setObservedAttribute("AttributeName");
```

To see all possible configurations of a `CounterMonitor` instance, refer to the JMX 1.0 API documentation, which you can download from <http://jcp.org/aboutJava/communityprocess/final/jsr003/index.html>. The archive that you download includes the API documentation.

Configuring GaugeMonitor Objects

`GaugeMonitor` objects observe changes in MBean attributes that are expressed as integers or floating-point. The following list describes groups of `GaugeMonitor` operations that you use to achieve typical configurations of a `GaugeMonitor` instance:

- Sends a notification when the observed attribute is beyond the high threshold.

```
setHighThreshold(int Highthreshold);
setNotifyHigh(true);
setObservedObject(ObjectName);
setObservedAttribute("AttributeName");
```

- Sends a notification when the observed attribute is outside the range of the high or low threshold.

```
setThresholds(int Highthreshold, Lowthreshold);
setNotifyHigh(true);
setNotifyLow(true);
setObservedObject(ObjectName);
setObservedAttribute("AttributeName");
```

- Sends a notification when the difference between two consecutive observations is outside the range of the high or low threshold.

```
setThresholds(int Highthreshold, Lowthreshold);
setNotifyHigh(true);
setNotifyLow(true);
setDifferenceMode(true);
setObservedObject(ObjectName);
setObservedAttribute("AttributeName");
```

`GaugeMonitor` does not support an offset or modulus.

To see all possible configurations of a `GaugeMonitor` instance, refer to the JMX 1.0 API documentation, which you can download from <http://jcp.org/aboutJava/communityprocess/final/jsr003/index.html>. The archive that you download includes the API documentation.

Configuring StringMonitor Objects

`StringMonitor` objects observe changes in MBean attributes that are expressed as strings. The following list describes groups of `StringMonitor` operations that you use to achieve typical configurations of a `StringMonitor` instance:

- Sends a notification when the observed attribute **matches** the string specified in `StringToCompare`.

```
setStringToCompare (String);  
setNotifyMatch (true);  
setObservedObject (ObjectName);  
setObservedAttribute ("AttributeName");
```

- Sends a notification when the observed attribute **differs from** the string specified in `StringToCompare`.

```
setStringToCompare (String);  
setNotifyDiffer (true);  
setObservedObject (ObjectName);  
setObservedAttribute ("AttributeName");
```

To see all possible configurations of a `StringMonitor` instance, refer to the JMX 1.0 API documentation, which you can download from <http://jcp.org/aboutJava/communityprocess/final/jsr003/index.html>. The archive that you download includes the API documentation.

Sample Monitoring Scenarios

This section outlines some typical MBean attributes that you might consider monitoring to observe performance and/or resource usage. For more details on individual MBean attributes or methods, see the [WebLogic Server Javadoc](#) for the MBean.

JDBC Monitoring

The `JDBCConnectionPoolRuntime` MBean maintains several attributes that describe the connections to a deployed JDBC connection pool. Applications may monitor these attributes to observe the connection delays and connection failures, as well as connection leaks. The following table outlines those MBean attributes typically used for JDBC monitoring.

Table 5-4 JDBC Monitoring Attributes

| JDBCConnectionPoolRuntime MBean Attribute | Typical Monitoring Application |
|--|---|
| <code>LeakedConnectionCount</code> | Notify a listener when the total number of leaked connections reaches a predefined threshold. Leaked connections are connections that have been checked out but never returned to the connection pool via a <code>close()</code> call; it is important to monitor the total number of leaked connections, as a leaked connection cannot be used to fulfill later connection requests. |
| <code>ActiveConnectionsCurrentCount</code> | Notify a listener when the current number of active connections to a specified JDBC connection pool reaches a predefined threshold. |
| <code>ConnectionDelayTime</code> | Notify a listener when the average time to connect to a connection pool exceeds a predefined threshold. |
| <code>FailuresToReconnect</code> | Notify a listener when the connection pool fails to reconnect to its datastore. Applications may notify a listener when this attribute increments, or when the attribute reaches a threshold, depending on the level of acceptable downtime. |

Index

- A**
- ADMIN_JNDI_NAME JNDI variable 2-7
 - Administration Console
 - defined 1-18
 - administration domain. *See* domain 1-3
 - Administration MBeanHome interface
 - defined 1-15
 - retrieving ClusterRuntimeMBean 4-14
 - retrieving from an external client 2-8
 - retrieving ServerRuntimeMBean 4-8, 4-10
 - retrieving through JNDI 2-7
 - retrieving with the Helper API 2-5
 - when to use 2-3
 - Administration MBeans
 - accessing from Administration Console 1-18
 - accessing from `weblogic.Admin` 1-18
 - API documentation 1-8
 - defined 1-4
 - interfaces for accessing 2-3
 - lifecycle 1-5–1-8
 - Managed Server Independence 1-8
 - retrieving a list of ??–2-14
 - `WebLogicObjectName` 2-21
 - Administration Servers 1-4–1-8
 - accessing MBeans 1-15
 - defined 1-3
 - JNDI tree 2-6
 - `LogMBeans` 2-25
 - registered MBeans 1-13
 - AttributeAddNotification object 5-10
 - AttributeChangeNotification object 5-10
 - AttributeRemoveNotification object 5-10
- C**
- child relationship with MBeans 2-23
 - clusters 4-14
 - `config.xml` file ??–1-8
 - editing from Administration Console 1-18
 - no runtime data 1-9
 - configurable MBean attributes. *See* dynamic changes to MBeans
 - Configuration MBeans
 - defined 1-2
 - See also* Local Configuration MBeans *and* Administration MBeans
 - CounterMonitor objects
 - configuring 5-36
 - type of data monitored 5-25
 - type of notifications emitted 5-25
 - custom MBeans 1-13
- D**
- derived gauge, defined 5-4
 - destroying MBeans 1-5
 - DifferenceMode attribute
 - for CounterMonitor objects 5-37

- for GaugeMonitor objects 5-38
- domains
 - defined 1-3
 - retrieving all MBeans 2-10
 - specified in WebLogicObjectName 2-21
- dynamic attributes in the Administration Console 1-18
- dynamic changes to MBeans 1-7

E

- e-mail 5-15
- error notification types 5-26
- examples
 - notification filter 5-15

G

- GaugeMonitor objects
 - configuring 5-38
 - type of data monitored 5-25
 - type of notifications emitted 5-26
- getAllMBeans method 2-10
- getMBeansByType method 2-14

H

- handleNotification method 5-11
 - for local applications 5-27
 - for remote applications 5-11, 5-27
- Helper API 2-4
- hierarchical relationship of MBeans 2-23

I

- instantiating MBeans 1-5
- Integer data type, monitoring 5-25

J

- Javadoc
 - for Configuration MBeans 1-8

I-2 Programming WebLogic Management Services with JMX

- for Runtime MBeans 1-11
- JDBC monitoring 5-40
- JMX object names 2-20
- JMX specification 1-1
- JNDI tree
 - Administration Servers 2-6
 - Managed Servers 2-6

L

- lifecycle of MBeans 1-5
- listen ports, setting 1-6
- listeners
 - creating 5-9, 5-27
 - defined 5-1
 - types of notification objects 5-25
- Local Configuration MBeans
 - accessing from weblogic.Admin 1-18
 - API documentation 1-8
 - defined 1-4
 - interfaces for accessing 2-3
 - lifecycle 1-5–1-8
 - no access from Administration Console 1-18
 - on Administration Server 1-13
 - retrieving a list of ??–2-14
 - WebLogicObjectName 2-21
 - WebLogicObjectName, examples 2-26
- Local MBeanHome interface
 - defined 1-15
 - retrieving from an internal client 2-9
 - retrieving ServerRuntimeMBean 4-5
 - retrieving through JNDI 2-7
 - retrieving with the Helper API 2-5
 - when to use 2-3
- LOCAL_JNDI_NAME JNDI variable 2-7
- log messages 5-10
- LogMBean on Administration Servers 2-25

M

managed resources, defined 1-2
Managed Server Independence (MSI) 1-8
Managed Servers
 defined 1-3
 JNDI tree 2-6
 local interface, performance of 1-15, 2-3
 MBean replicas 1-4, 1-5
 MBeans accessible from 1-13, 1-15
 propagating changes to Local
 Configuration MBeans 1-7
 runtime information about clusters 4-14
 See also Local `MBeanHome` interface
MBean types, defined 2-21
`MBeanHome` interface 1-14
 See also Local `MBeanHome` interface,
 Administration `MBeanHome`
 interface, *and* type-safe
 interface
`MBeanHome` methods. *See* type-safe interface
MBeans
 accessing, main steps 2-1
 creating custom 1-13
 defined 1-2
 notifications generated 5-10
 See also Local Configuration MBeans,
 Administration MBeans, *and*
 Runtime MBeans
`MBeanServer` interface
 accessing MBeans 2-18
 defined 1-14
 registering listeners 5-17
 retrieving and changing runtime data
 4-12
 when to use 2-4
message level for standard out 3-2
metrics for runtime data 1-9
modulus for `CounterMonitor` objects 5-37
monitor MBeans
 defined 5-3

 types 5-24
monitoring attributes of MBeans
 comparing changes to MBean attributes
 5-39
 JDBC example 5-40
 main steps 5-24
 notification types 5-25
MSI 1-8

N

names of MBeans 2-21
notification filters
 creating and registering 5-14
 example 5-15
notification listeners. *See* listeners
notifications
 defined 5-1
 types 5-25

O

object names for MBeans 2-10, 2-20
overriding values
 in `config.xml` 1-6

P

parent relationship with MBeans 2-23
performance metrics 1-9
persistence
 of runtime data 1-9
propagating changes to Local Configuration
 MBeans 1-7

R

registering MBeans 1-13
`RemoteMBeanServer` interface
 defined 1-14
`RemoteNotificationListener` object
 5-11, 5-27

replicas of Administration MBeans 1-5

RMI 1-15

runtime changes to MBeans 1-7, 1-18

Runtime MBeans

- API documentation 1-11

- defined 1-2

- distribution 1-9

- interfaces for accessing 2-3

- on Administration Server 1-13

- persistence 1-9

- retrieving a list of ??–2-14

- retrieving with Administration

 - `MBeanHome.getMBeansByType`
e 4-8

- `WebLogicObjectName` 2-21

Runtime MBeans, accessing

- from Administration Console 1-18

- from Administration `MBeanHome` 2-12,
4-8

- from Local `MBeanHome` 4-6

- from `MBeanServer` 4-12

- from `weblogic.Admin` 1-18

S

security MBeans 1-12

`ServerRuntimeMBean` interface

- accessing from Administration

 - `MBeanHome` 4-8

- changing with `MBeanServer` 4-12

- defined 4-5

standard out

- configuring message level with

 - `MBeanServer` 3-3

String data type, monitoring 5-25

StringMonitor objects

- configuring 5-39

- type of data monitored 5-25

- type of notifications emitted 5-26

T

thresholds

- for `CounterMonitor` objects 5-36

- for `GaugeMonitor` objects 5-38

type, MBean 2-21

type-safe interface

- accessing MBeans 2-10–2-14

- defined 1-14

- when to use 2-3

W

`weblogic.Admin` utility

- changing configuration data 3-2

- defined 1-18

- determining active domain and servers
4-4

- finding `WebLogicObjectName` 2-24

`weblogic.Server` startup command 1-5

`WebLogicObjectName`

- defined 2-20

- examples 2-26

- finding with `weblogic.Admin` 2-24

- retrieving with

 - `WebLogicMBean.getName`
2-10

- using to retrieve `ServerRuntimeMBean`
4-10