BEA WebLogic Server™

Programming WebLogic JMX Services

BEA WebLogic Server Version 6.1
Document Date: June 24, 2002

Copyright

Restricted Rights Legend

Trademarks or Service Marks

Programming WebLogic JMX Services

Part Number	Document Date	Software Version
N/A	June 24, 2002	BEA WebLogic Server Version 6.1, Service Pack 3

Contents

About This Document

1. Overview of WebLogic JMX Services

About This Document

This document describes how to use the BEA WebLogic Server™ managment APIs to enhance WebLogic Server to support your applications.

The document is organized as follows:

- Chapter 1, "Overview of WebLogic JMX Services," describes the WebLogic Server management interface, and provides overviews of MBeans, the administrative domain, and server configurations.
- Chapter 2, "Accessing WebLogic Server MBeans," describes how to access and use WebLogic Server MBeans from a client application.
- Chapter 3, "Using MBean Notifications," desribes how to listen and respond to MBean notifications in a client application.
- Chapter 4, "Monitoring WebLogic Server MBeans," describes how to monitor MBean attributes from a monitoring MBean.

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 core technologies. It is assumed that readers have a familiarity with the BEA WebLogic Server platform and the Java programming language.

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.

Related Information

The BEA corporate Web site provides all documentation for WebLogic Server. The following BEA WebLogic Server documentation contains information that is relevant to understanding how to extend WebLogic Server.

- BEA WebLogic Server Documentation (available online):
  - *Administration Guide*
  - *Programming Guides*
  - *WebLogic Server API*

- The Sun Microsystems, Inc. Java site at `http://java.sun.com/`

For more information about BEA WebLogic Server and Java, refer to the Bibliography at `http://edocs.bea.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 your 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.
italics	Emphasis and book titles.

`monospace text`	Code samples, commands and their options, Java classes, data types, directories, and filenames and their extensions. Monospace text also indicates text that you enter from the keyboard.
	Examples:
	`import java.util.Enumeration;`
	`chmod u+w *`
	`config/examples/applications`
	`.java`
	`config.xml`
	`float`

`monospace italic text`	Variables in code. *Example*: `String CustomerName;`
UPPERCASE TEXT	Device names, environment variables, and logical operators. *Examples*: LPT1 BEA_HOME OR
`{ }`	A set of choices in a syntax line.
`[ ]`	Optional items in a syntax line. *Example*: `java utils.MulticastTest -n name -a address [-p portnumber] [-t timeout] [-s send]`

Convention	Usage

|	Separates mutually exclusive choices in a syntax line. *Example*:
	`java weblogic.deploy [list|deploy|undeploy|update]`
	`password {application} {source}`

. . .	Indicates one of the following in a command line: ■ 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
. . .	Indicates the omission of items from a code example or from a syntax line.

1 Overview of WebLogic JMX Services

The following sections provide an introduction to the WebLogic Server JMX management framework:

Overview

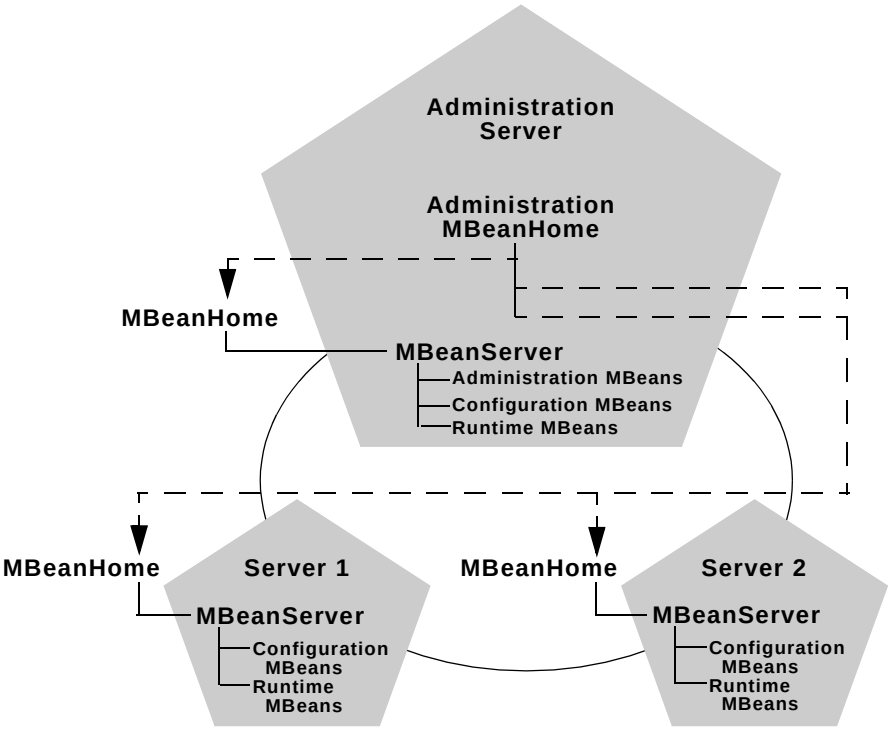
The WebLogic Server management architecture is based on the Sun Microsystems, Inc. Java Management Extensions (JMX) specification. BEA has instrumented many of the APIs and resources available in WebLogic Server, and provides the JMX-compliant Administration Console to monitor and manage those resources.

All WebLogic Server management functions are accessed using *management beans*, or *MBeans*, which retrieve their values from the WebLogic Server *domain configuration* or *runtime state*. MBeans provide developers with a means to access all configuration and monitoring information about WebLogic Server programmatically via the JMX standard API.

This guide provides an overview of the WebLogic Server JMX implementation, so that you can develop applications or management frameworks that monitor and manage WebLogic Server JMX-manageable resources.

The WebLogic Server Management System

The WebLogic Server management system implements all of the required components identified in the JMX 1.0 specification. Because a WebLogic Server installation can include multiple servers, JMX components are necessarily distributed throughout the installation. The figure below depicts JMX components in a typical WebLogic Server installation.



The sections that follow provide an overview of each JMX component.

Managed Resources

Managed resources comprise the APIs, services, and applications that are hosted on a WebLogic Server instance, and which BEA has instrumented for JMX management. Each managed resource provides one or more MBeans that can be used for monitoring or modifying the resource.

Instrumented APIs in WebLogic include EJB, JDBC, JMS, JTA, and XML. BEA has also instrumented WebLogic Server services such as startup classes, shutdown classes, and security realms. Finally, WebLogic Server provides JMX instrumentation for Web applications and their respective components, so that you can change an application's deployment parameters or monitor its deployment status using the JMX specification.

MBeans

MBeans (managed beans) are the JMX constructs that represent managed resources. Each managed resource in WebLogic Server (an API, service, or application component) uses one or more MBeans to provide an interface for modifying or monitoring the resource.

WebLogic Server MBeans provide all of the standard operations defined in the JMX specification such as:

- Constructors for instantiating MBeans
- Methods for listing, setting and getting the MBean's attributes
- Methods for performing additional MBean-specific operations
- Notifications for broadcasting MBean events

In the context of the JMX specification, all WebLogic MBeans are implemented as standard MBeans—their attributes and operations are specified directly in their associated interfaces. WebLogic Server defines several distinct types of MBean, to describe their function within the WebLogic Server management system. See WebLogic Server MBeans for more information on specific MBean types.

MBean Servers

As described in the JMX specification, an MBean server is the principle agent for accessing MBeans in the management framework. The MBean server acts as a registry for MBeans. Using the MBean Server, management applications can look up MBeans, determine MBean attributes and methods, and listen for MBean notifications.

Each server in a WebLogic administration domain contains its own MBean Server. With the exception of the administration server's MBean Server, each MBean Server registers only those MBeans that apply to the local WebLogic Server instance. For example, an application using the MBean Server for a managed WebLogic Server instance could monitor a Web application that was deployed on that particular server, but it could not monitor a Web application deployed on another server in the domain.

Because the administration server is itself a WebLogic Server instance, it also has an MBean server. The administration server's MBean server is unique in that it hosts domain-wide administration MBeans as well as the server's own configuration and runtime MBeans.

MBean Homes

The JMX 1.0 specification does not provide guidelines for making the MBean server interface available to management clients outside of the MBean server's JVM. WebLogic Server Version 6.1 makes the MBean server interface available to any client (local or external to the server's JVM) via the `MBeanHome` interface.

An `MBeanHome` is simply a wrapper around an MBean server interface that can be used for accessing WebLogic Server MBeans. In most cases, applications can use `MBeanHome` in place of the MBean server for managing server resources. Any client can access management functions by using a simple JNDI lookup to obtain the `MBeanHome` of an MBean server.

The `MBeanHome` interface provides a strongly-typed interface for accessing WebLogic Server MBean attributes, which generally makes `MBeanHome` easier to use than MBean server. For example, once an application has obtained a `serverMBean` from `MBeanHome`, it can call `serverMBean.getListenPort()` to return an `int` value of the server's listen port. To perform a similar operation using MBean server, the application would need to first obtain the JMX object name of the `serverMBean` and

request its `ListenPort` attribute. The attribute itself would be returned as a generic object, and the application would need to know that this particular attribute should be cast into an `int` value.

`MBeanHome` can be used only for accessing WebLogic Server MBeans—applications cannot obtain user-defined MBeans using `MBeanHome`. Pure JMX applications, or applications that need to access registered user MBeans, can obtain and use the MBean server interface by first looking up `MBeanHome` and invoking `getMBeanServer()`.

As shown in the previous figure, each WebLogic Server instance in an administration domain has an MBean server and a corresponding `MBeanHome`. Using these `MBeanHome` interfaces, an application can work with configuration and runtime MBeans for an individual WebLogic Server.

In addition to providing a standard `MBeanHome` interface (for a server instance's local configuration and runtime MBeans), the administration server provides an additional, domain-wide `MBeanHome` interface, as described below.

Administration MBeanHome

The WebLogic Server management system utilizes a domain-wide `MBeanHome` interface that can access all WebLogic MBeans for all server instances in a management domain. This includes the administration MBeans for the domain as a whole, as well as the configuration and runtime MBeans for the administration server and all managed servers.

Although the domain-wide `MBeanHome` interface does not have an associated MBean server, it operates in the same manner as a server-specific `MBeanHome`. Only the list of available MBeans differs. The administration `MBeanHome` accesses another server's MBeans using the respective server's `MBeanHome` interface, as shown by the dotted lines in the previous figure.

Applications obtain the domain-wide `MBeanHome` interface via the WebLogic administration server. After obtaining the domain-wide `MBeanHome`, an application can work with domain-wide administration MBeans or any individual server's MBeans by filtering the list of available MBeans in the domain. Accessing WebLogic Server MBeans explains how to obtain an `MBeanHome` interface programmatically, and also provides information about which `MBeanHome` interface an application should obtain.

WebLogic Server MBeans

WebLogic Server defines three distinct types of MBeans:

- Administration MBeans, which represent domain-wide configuration parameters read from `config.xml`.
- Configuration MBeans, which are the per-server copies of administration MBeans that a server uses to configure itself.
- Runtime MBeans, which represent the run-time state of various WebLogic Server components and subsystems.

The following sections describe each MBean type.

Administration MBeans

Administration MBeans represent the configured properties of an entire WebLogic Server administration domain. When you start up the administration server for a domain, the server creates administration MBeans using the elements and attributes specified in the domain's `config.xml` file.

All administration MBeans are registered automatically when the administration server starts up. This includes administration MBeans for managed servers that are not yet running, or that have not yet attached to the administration server.

JMX management applications can modify the administration domain's `config.xml` file indirectly by changing attributes of administration MBeans. Every 5 minutes, the administration MBean server checks to determine if administration MBeans have been changed, and writes the changes back to `config.xml` as necessary.

Changes to administration MBeans are also written back to the `config.xml` file when the administration server shuts down, or when MBean attributes are modified by a WebLogic Server utility such as the Administration Console, `weblogic.Admin`, or `weblogic.Deploy`.

Configuration MBeans

Whereas administration MBeans represent the persistent value of `config.xml` elements, configuration MBeans represent the "active" value of those same elements. It is the active value of the configuration attributes (the configuration MBeans) that WebLogic Server subsystems use for operation during the life span of the server.

When you start up a WebLogic Server, most of its configuration MBean attributes are derived from the server's administration MBeans as registered in the administration server. For example, a managed server might connect to the administration server and derive all of its configuration MBeans from the associated administration MBeans.

However, when you start up a WebLogic Server, it is also possible to override `config.xml` properties using command-line options. In this case, a server's configuration MBean attributes are populated from the override values, rather than the administration MBean values. Attributes that do not have override values are then derived via administration MBeans registered in the administration server.

JMX applications can modify configuration MBeans to temporarily affect the configuration of an active WebLogic Server instance. However, changes to configuration MBeans are lost when the server reboots or shuts down. To make permanent changes to a server's configuration, the application should instead modify the corresponding resource's administration MBean, which is automatically persisted back to the `config.xml` file. Changes made to an administration MBean are also propagated to the corresponding configuration MBean, so that the WebLogic Server subsystems use the newly-configured attribute value.

Runtime MBeans

A runtime MBean represents the run-time transient state of the underlying resource or subsystem that it represents. Runtime MBeans differ from administration and configuration MBeans in that their attribute values are not derived or overridden—rather, they represent the current state of a server resource at a given point in time.

For example, runtime MBeans are used to represent the current number of sockets a WebLogic Server has opened, or the current state of the server (whether it is running, suspended, or is about to be shut down).

Applications can use runtime MBeans to monitor the resource usage of managed resources, such as Web applications, and to potentially diagnose performance bottlenecks.

MBean Naming Conventions

All WebLogic Server MBeans have a name, a type and a domain. These attributes are reflected in the MBean's JMX Object Name. The Object Name is the unique identifier for a given MBean across all domains, and has the following structure:

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

`Name` is a unique identifier for a given domain and type of MBean.

`Type` indicates type of managed resource the MBean exposes. Examples of resource types include `Server`, `WebComponent` or `JDBCConnectionPoolRuntime`. `Type` is also used to distinguish between administration, configuration, and runtime MBeans by appending the following standard suffixes:

- <no suffix> indicates an Administration MBean
- `Config` indicates a configuration MBean
- `Runtime` indicates a runtime MBean

For example, the value of `Type` for a `JDBCConnectionPool` MBean is:

- `JDBCConnectionPool` for an Administration MBean
- `JDBCConnectionPoolConfig` for a Configuration MBean
- `JDBCConnectionPoolRuntime` for a runtime MBean

Note that the "MBean" suffix is removed from the MBean interface name to get the base type of an MBean. In the case of the `JDBCConnectionPool` MBean, the actual MBean interface name is `JDBCConnectionPoolMBean`.

Specific kinds of MBeans have additional attributes in the JMX object name. All runtime and configuration MBeans have a `Location` component that uses the name of the server on which that MBean is located as its value. For example:

```
mydomain:Name=myServlet,Type=ServletRuntime,Location=myserver
```

Any MBean that has a child relationship with a parent MBean has an extra attribute in its object name to identify the relationship. The format of the attribute is:

TypeOfParentMBean=NameOfParentMBean

In the following example, `Server` is the type of Parent MBean, and `myserver` is the name of the Parent MBean:

```
mydomain:Name=mylog,Type=Log,Server=myserver
```

Package Naming Conventions

All interface types for administration and configuration MBeans are located in the `weblogic.management.configuration` API.

All interfaces types for runtime MBeans are located in the `weblogic.management.runtime` API.

Agent-level interfaces (for example, the `MBeanHome` and `RemoteMBeanServer` interfaces) are located in the `weblogic.management` API.

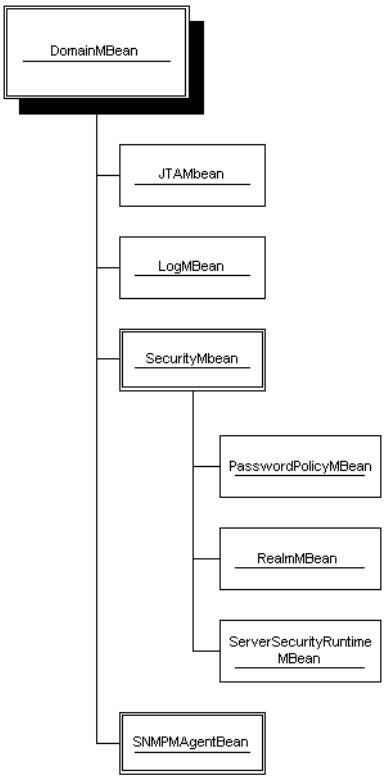
Quick Reference to WebLogic Server MBeans

WebLogic Server provides a large number of MBeans used for configuring the server system as well as applications. In many cases, related MBeans are easily accessed via a “parent” mbean that provide getter to obtain one or more related MBeans. This section provides an overview and quick reference to the major categories of WebLogic Server MBeans to help you better focus your JMX programming efforts.

Note: The WebLogic Server Management API is fully documented online in the Javadoc. The WebLogic Server Programming Guides provide additional information about the programming APIs and services modeled by WebLogic Server MBeans.

Domain MBean

`DomainMBean` is a high-level WebLogic Server MBean that represents an entire management domain. Once you have obtained a `DomainMBean`, you can use its getter methods to obtain MBeans representing the domain's log, security, SNMP, and JTA configuration, as shown in the figure below. Similarly, MBeans such as `SecurityMBean` and `SNMPAgentMBean` provide getters to access MBeans that control portions of their configuration.



Target MBeans

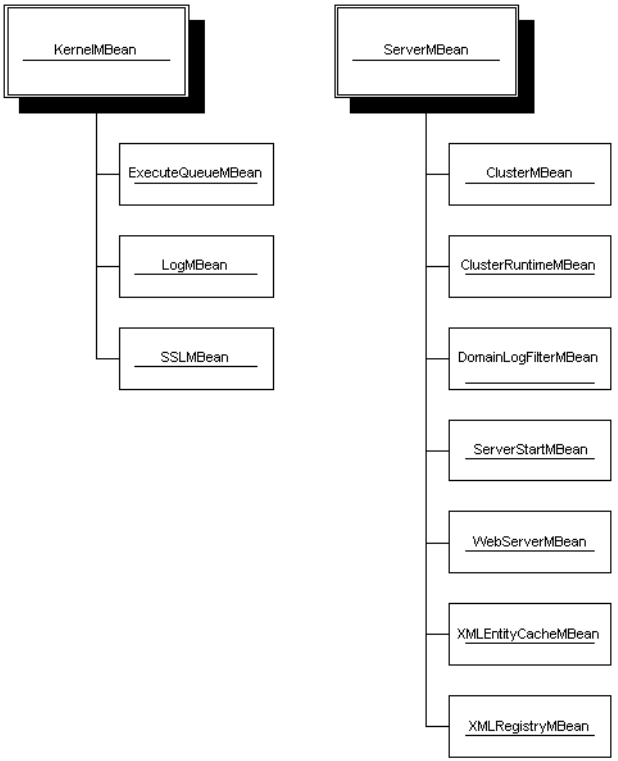
Target MBeans represent objects that you can select when deploying applications and resources in a management domain. These include MBeans that represent WebLogic Server instances and WebLogic Server clusters.

All Target MBeans implement the `weblogic.management.configuration.TargetMBean` interface. This means that both clusters and servers can be selected as a target for deploying application components, or as targets for deploying resources such as connection pools.

Server and Kernel MBeans

The `ServerMBean`, which extends the `KernelMBean` interface, represents a particular WebLogic Server instance in a management domain. Applications that obtain the `ServerMBean` can conveniently access child MBeans that control the configuration of the associated server and WebLogic Server Kernel.

The figure below depicts the child MBeans that can be obtained via getter methods in the `KernelMBean` and `ServerMBean` interfaces.



Cluster MBeans

Applications can also select a configured WebLogic Server cluster as a target when deploying Web applications or WebLogic Server resources. `ClusterMBean` primarily uses getter and setter methods to configure cluster properties, such as load balancing algorithms and multicast message properties. `ClusterMBean` also has a getter method to return all `ServerMBeans` that are members of the cluster.

Deployable Unit MBeans

A large number of WebLogic Server MBeans implement the `DeploymentMBean` interface. `DeploymentMBean` represents any type of web application, web application component, or WebLogic Server resources that can be deployed to a server or cluster in the domain.

If you are interested in working with deployable units, first familiarize yourself with weblogic.management.configuration.DeploymentMBean, as this interface provides the basic methods used to obtain or add targets, as well as set the deployment order.

Once you are familiar with the basic operations `DeploymentMBean`, refer to the individual deployable unit MBeans that implement the interface. MBeans that represent deployable application components include:

- ComponentMBean
- ConnectorComponentMBean
- EJBComponentMBean
- ShutdownClassMBean
- StartupClassMBean
- VirtualHostMBean
- WebAppComponentMBean
- WebDeploymentMBean
- WebServerMBean

MBeans that represent deployable WebLogic Server resources include:

- JDBCConnectionPoolMBean
- JDBCDataSourceMBean
- JDBCMultiPoolMBean
- JDBCTxDataSourceMBean
- JMSConnectionFactoryMBean
- JMSServerMBean

- MessagingBridgeMBean
- RMCFactoryMBean
- WLECConnectionPoolMBean

2 Accessing WebLogic Server MBeans

The following sections describe how to access WebLogic Server MBeans from a client application or management framework:

Overview

As described in The WebLogic Server Management System, two primary agent-level interfaces provide client access to MBeans: `MBeanServer` and `MBeanHome`.

BEA provides `MBeanHome` as a simple, strongly-typed interface for accessing MBeans from clients that are either internal or external to the WebLogic Server JVM. If your application requires pure JMX-compliant access to MBeans, you can also obtain the `MBeanServer` interface via `MBeanHome`.

This section describes the basic procedure for obtaining an `MBeanHome` and accessing WebLogic Server MBeans via the `MBeanHome` interface. See the JMX specification for more information about accessing MBeans via the MBean server interface.

Selecting the Client Interface to WebLogic Server MBeans

Each server in a domain contains an MBeanHome (and a corresponding MBeanServer), which hosts configuration and runtime MBeans on that server. In addition, the administration server has an administration `MBeanHome` that provides access to all MBeans in the entire domain. The particular interface that you choose to use in your application depends on:

- Whether your application needs to be purely JMX-compliant
- Whether your application needs to access user-defined MBeans
- Whether or not your application works with administration MBeans
- Whether or not your application manages a single WebLogic Server, or multiple WebLogic Servers

MBeanHome Versus MBeanServer

The `MBeanHome` interface provides easy access to WebLogic Server MBeans. However, it does not provide access to user-defined MBeans that may be registered in the `MBeanServer` interface. If your application must access user-defined MBeans, it must do so using the `MBeanServer` interface.

You may also choose to use the `javax.management.MBeanServer` interface if your application must fully comply with the JMX specification. Note, however, that the `MBeanHome` interface provides a strongly-typed interface for accessing WebLogic MBeans, and is generally easier to use than `MBeanServer`.

Note: All examples in this chapter use `MBeanHome` as the primary method for accessing MBeans. For information on using `MBeanServer`, see the JMX specification.

Server MBeanHome Versus Administration MBeanHome

Applications can use individual server `MBeanHome` interfaces and/or the administration `MBeanHome` interface, depending which MBean(s) the application accesses.

If your application needs to manage administration MBeans, you must use the domain-wide `MBeanHome` interface on the administration server, because administration MBeans are not available via the `MBeanHome` interfaces of managed servers.

If your application will manage multiple WebLogic Server instances in a domain, it may be preferable to use the domain-wide `MBeanHome` interface. The domain-wide interface allows you to access MBeans from any WebLogic Server in the management domain, by filtering the JMX object names.

If your application manages only a single WebLogic Server instance in a domain, then you may want to obtain the local `MBeanHome` interface for that server, rather than the domain-wide `MBeanHome`. Using the local interface saves you the trouble of filtering MBeans to find those that apply to the single server. Using the local interface also uses fewer network hops to access MBeans, because you are connecting directly to the managed server itself.

Obtaining an MBeanHome Using JNDI

The `MBeanHome` of any server can be obtained from the relevant server's JNDI tree by using the `MbeanHome.LOCAL_JNDI_NAME` constant.

The domain-wide administration `MBeanHome` is published in the administration server's JNDI tree at `MBeanHome.ADMIN_JNDI_NAME`.

The administration server also publishes an `MBeanHome` for each server in the domain in its JNDI tree. This `MBeanHome` is available from the JNDI tree of the administration server, and can be accessed using the `MbeanHome.JNDI_NAME+"."+`*`relevantServerName`* constant.

The `javax.management.MBeanServer` for an individual server's `MBeanHome` can be obtained by invoking the `getMBeanServer()` method on that `MBeanHome`. Note that the domain-wide `MBeanHome` does not have a corresponding `javax.management.MBeanServer`; calling `getMBeanServer()` on the administration `MbeanHome` returns the `MBeanServer` of the admninistration server.

Example: Looking Up MBeanHome from an External Client

The following example shows how an application running in a separate JVM would look up an `MBeanHome` interface on the administration server.

```
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 void findExternal(String host,
                         int    port,
                         String password) {
        String url = "t3://" + host +
                     ":" + port;
        String username = "system";
        try {
            Environment env = new Environment();
            env.setProviderUrl(url);
            env.setSecurityPrincipal(username);
```

```
            env.setSecurityCredentials(password);
            ctx = env.getInitialContext();
            home = (MBeanHome)ctx.lookup(MBeanHome.JNDI_NAME + "." +
                                         SERVER_NAME);
            System.out.println(SERVER_NAME +
                               " MBeanHome found externally");
            ctx.close();
        } catch (AuthenticationException ae) {
            System.out.println("Authentication Exception: " + ae);
        } catch (CommunicationException ce) {
            System.out.println("Communication Exception: " + ce);
        } catch (NamingException ne) {
            System.out.println("Naming Exception: " + ne);
        }
    }
```

Example: Looking Up 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 monitor), the JNDI lookup for the `MBeanHome` is simpler. The following example shows how an JSP running in the same JVM as the administration server would look up an `MBeanHome`.

```
...
public void findInternal() {
        Environment env = new Environment();
```

```
        try {
            ctx = env.getInitialContext();
            home = (MBeanHome)ctx.lookup(MBeanHome.JNDI_NAME + "." +
                                         SERVER_NAME);
            System.out.println(SERVER_NAME +
                               " MBeanHome found internally");
            ctx.close();
        } catch (NamingException ne) {
            System.out.println("Naming Exception: " + ne);
        }
    }
```

Example: Obtaining MBeanServer from MBeanHome

For applications that need to interact directly with the `MBeanServer` interface, `MBeanHome` provides a simple method to obtain its associated `MBeanServer`.

```
...
home = (MBeanHome)ctx.lookup(MBeanHome.ADMIN_JNDI_NAME);
RemoteMBeanServer homeServer = (RemoteMBeanServer)home.getMBeanServer();
...
```

Using the Helper Class to Obtain MBeanHome Interfaces

WebLogic Server version 6.1 provides the `weblogic.management.Helper` class to further simplify the process of obtaining `MBeanHome` interfaces in an internal client. The `Helper` class provides a methods to obtain the server or administration `MBeanHome`.

For example, to obtain both the administration server and local server `MBeanHome` using the `Helper` class:

```
public void find(String host,
                     int port,
                     String password) {
        String url = "t3://" + host +
                     ":" + port;
        try {
            localHome = (MBeanHome)Helper.getMBeanHome("system",
                                                       password,
                                                       url,
                                                       SERVER_NAME);
            adminHome = (MBeanHome)Helper.getAdminMBeanHome("system",
                                                            password,
                                                            url);
            System.out.println("Local and Admin Homes " +
                               "found using the Helper class");
        } catch (IllegalArgumentException iae) {
            System.out.println("Illegal Argument Exception: " + iae);
        }
```

```
    }
```

Accessing MBeans from MBeanHome

After obtaining the `MBeanHome`, you can look up individual MBeans using the methods described in `javax.management.MBeanHome`. For example, to look up all MBeans in the administration `MBeanHome` and print their JMX object names:

```
public void displayMBeans() {
        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.getObjectName();
            System.out.println(objectName.getName() +
                               " is a(n) " +
                               mbean.getType());
        }
    }
```

You can access individual MBeans by using the `MBeanHome.getMBean()` methods. `getMBean()` has several different method signatures, the simplest of which returns a `WebLogicMBean` with the given name and type in the default domain.

`MBeanHome` provides additional getter methods to obtain specific WebLogic Server MBean types. For example, to obtain the `Server` configuration MBean from the current domain, you can use the `getConfigurationMBean()` method:

```
String myBeanType = "ServerConfig";
ConfigurationMBean myServerMBean =
        home.getConfigurationMBean(SERVER_NAME, myBeanType);
```

To obtain a runtime MBean, use the `getRuntimeMBean()` method. A **runtime MBean** is a local MBean that gives runtime information about WebLogic Server and application components. Unlike other `MBeanHome` methods, `getRuntimeMBean()` returns only runtime MBeans that reside on the current WebLogic Server. If you call `MBeanHome.getRuntimeMBean()`on the Administration Server, it does not return runtime MBeans from Managed Servers. For example, the following code fragment returns the `JDBCConnectionPoolRuntime` MBean from the current WebLogic Server:

```
String poolName = "requestConnectionPool";
JDBCConnectionPoolRuntimeMBean runtimeMBean =
        (JDBCConnectionPoolRuntimeMBean)home.getRuntimeMBean(poolName,
        "JDBCConnectionPoolRuntime");
```

See the Javadocs for `weblogic.management.MBeanHome` for information about each of the getter methods available in `MBeanHome`.

Registering Custom MBeans with MBeanServer

Because WebLogic Server management services are implemented using JMX, you can also create your own MBeans and register them with an MBean Server in a WebLogic Server installation. This allows you to leverage the WebLogic Server MBean Server implementation to host your own MBeans and make them available to internal and external clients.

Note that all custom MBeans must be registered and accessed using the JMX-compliant `MBeanServer` interface. You cannot use the `MBeanHome` interface for custom MBeans, as `MBeanHome` only makes WebLogic Server MBeans available to clients. Furthermore, you cannot use BEA utilities such as `weblogic.Admin` to access custom MBeans.

The example that follows shows a very basic MBean implementation and a client application that registers the MBean with the MBean Server on an administration server. Note, however, that this example does not show all of the requirements (for

example, MBean exception handling) outlined in the JMX specification. For full details on implementing your own custom MBeans, please refer to the JMX specification.

Example Custom MBean

For the purposes of this example, the custom MBean consists of a skeleton interface requiring only a single method implementation:

```
public interface MyCustomMBean {

     int getMyAttribute();

}
```

Example Client Application

The client application performs the following actions:

- Obtains the `MBeanHome` of the administration server using the WebLogic Server Helper class
- Obtains the associated MBean Server interface using MBeanHome
- Registers the custom MBean with the MBean Server
- Obtains the value of a custom MBean attribute
- Unregisters the MBean from the MBean Server

Note that many of the above actions, such as obtaining `MBeanHome` and `MBeanServer`, are discussed earlier in this section. Only the registration and attribute calls differ for MBeans, because these calls operate directly against the MBean Server interface (and are full JMX-compliant). Additional information appears within the Java comments.

```
import weblogic.management.MBeanHome;

import weblogic.management.Helper;

import weblogic.management.RemoteMBeanServer;

import javax.management.*;
```

```
import MyCustomMBean;

// The client class implements MyCustomMBean, and the main function obtains
// MBeanHome and MBeanServer; registers the MBean; accesses an attribute value;
// and unregisters itself.

public class MyCustom implements MyCustomMBean, java.io.Serializable {

  public static void main(String[] args)
  // An actual JMX client would handle appropriate exceptions at different points
  // in the application. For clarity, this example client only throws exceptions.
  throws Exception {

    // The client obtains the MBeanHome of the administration server using the
    // Helper class.
    MBeanHome mbh = Helper.getMBeanHome("system",
    "system_password","t3://localhost:7001","examplesserver");

    // The client obtains the MBeanServer interface via MBeanHome.
    RemoteMBeanServer mbs = mbh.getMBeanServer();

    // To use JMX calls against the MBeanServer interface, the client must use
    // ObjectNames.
    ObjectName mbo = new ObjectName("user_Domain:Name=x");

    // The client attempts to register the custom MBean with the MBeanServer.
    try {
      mbs.registerMBean((Object)new MyCustom(),mbo);
```

```
    } catch(InstanceAlreadyExistsException i) {
      System.out.println("MBean ("+mbo+") allready exists");
    }

    // The client obtains and prints the value of MyAttribute.
    System.out.println("Value of MyAttribute of ("+mbo+")from MBeanServer = "+
    mbs.getAttribute(mbo,"MyAttribute"));

    // The custom MBean is unregistered.
      mbs.unregisterMBean(mbo);
  }

  // The example client implements the MyCustomMBean interface's single method
  // with prinhted output.
  public int getMyAttribute() {
    System.out.println("getMyAttribute invoked.");
    return 999;
  }
}
```

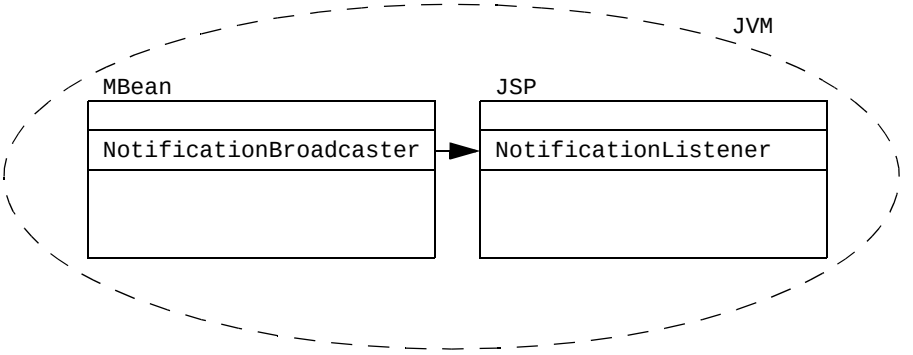
3 Using MBean Notifications

The following sections provide an overview of how to use the various notifications that can be broadcast from WebLogic Server MBeans:

Overview

All WebLogic Server MBeans implement the `javax.management.NotificationBroadcaster` interfaces, which means they can emit standard JMX notification types.

To observe MBean notifications, you simply implement the `NotificationListener` interface in your client application, and register the listener class with the MBeans whose notifications you want to receive. The figure below shows a basic system to monitor notifications using a JSP or Servlet.



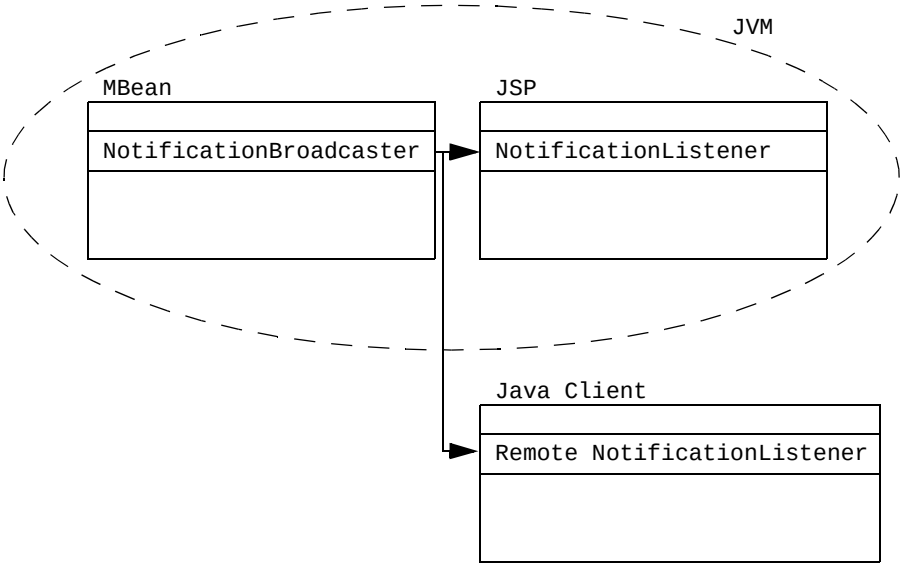
A listener class can optionally register a `NotificationFilter` class, which provides additional control over which notifications the listener receives.

Note: For an complete explanation of JMX notifications and how they work, see the Sun Microsystems J2EE JMX specification.

Making Notifications Available to Outside Clients

The JMX version 1.0 specification does not describe how to make notifications available to clients outside the broadcasting MBean's JVM. WebLogic Server version 6.1 makes notifications available externally via the `weblogic.management.RemoteNotificationListener` interface.

`RemoteNotificationListener` extends `javax.management.NotificationListener` and `java.rmi.Remote`, making MBean notifications available to external clients via RMI. Remote Java clients simply implement `RemoteNotificationListener`, rather than `NotificationListener`, to accept WebLogic MBean notifications, as shown below.



Registration of the remote Java client listener is accomplished through the standard JMX `addNotificationListener()` method.

MBean Notification Summary

WebLogic Server notifications use the standard notification classes identified in the JMX 1.0 specification. In addition, WebLogic Server provides additional notification classes and notification helper classes for working with WebLogic Server MBean log notifications. The following sections summarize the notification types and classes that JMX applications can use in WebLogic Server.

Basic JMX Notifications

All WebLogic Server MBeans implement the `NotificationBroadcaster` interface, and can generate the notification types described in the JMX 1.0 specification. These notification types include:

- `javax.management.AttributeChangeNotification`, for notifying when an MBean attribute value has been changed, and
- `javax.management.MbeanServerNotification`, for notifications delegated to the MBean server.

In addition, certain WebLogic Server MBeans support two additional notification types for attributes that have "add" and "remove" methods:

- `weblogic.management.AttributeAddNotification` is broadcast when an attribute's `add`*`AttributeName`* method is called.
- `weblogic.management.AttributeRemoveNotification` is broadcast when an attribute's `remove`*`AttributeName`* method is called.

WebLogic Server Log Notifications

WebLogic Server provides the `LogBroadcasterRuntime` MBean, whose sole responsibility is to broadcast log messages. Client applications that need to listen for log notifications can simply register with the `LogBroadcasterRuntime` MBean.

A notification that represents a WebLogic Server log message contains many additional pieces of information, such as:

- The name of the machine that issued the log message
- The name of the WebLogic Server that issued the log message
- The log message ID number

To help JMX applications extract and use this WebLogic Server log information, BEA provides the `WebLogicLogNotification` wrapper class. `WebLogicLogNotification` provides simple getter methods to extract parts of the log message, as well as methods to obtain the transaction ID, user ID, and version number associated with the message.

Working with WebLogic Server Log Notifications provides details on using the log notification supporting classes and interfaces.

Using Basic JMX Notifications

To receive WebLogic MBean notifications, an external client application must:

1. Implement the `RemoteNotificationListener` interface.
2. Register the listener class implementation with the MBean(s) whose notifications you want to receive.

The following sections describe these basic steps.

Creating a Notification Listener

The notification listener class is responsible for handling the JMX notifications broadcast by one or more MBeans. If your JMX application resides outside of the broadcasting MBean's JVM, the listener class should implement `weblogic.management.RemoteNotificationListener`, supplying a `handleNotification()` class to perform actions when notifications are received. An example implementation follows:

```
import javax.management.Notification;
import javax.management.NotificationFilter;
import javax.management.NotificationListener;
import javax.management.Notification.*;
...
public class WebLogicLogNotificationListener implements
        RemoteNotificationListener {
...
public void handleNotification(Notification notification, Object obj) {
```

```
        WebLogicLogNotification wln = (WebLogicLogNotification)notification;
        System.out.println("WebLogicLogNotification");
        System.out.println("         type = " +
                           wln.getType());
        System.out.println("   message id = " +
                           wln.getMessageId());
        System.out.println("  server name = " +
                           wln.getServername());
        System.out.println("    timestamp = " +
                           wln.getTimeStamp());
        System.out.println("      message = " +
                           wln.getMessage() + "\n");
    }
```

Registering Notification Listeners with MBeans

Because all WebLogic Server MBeans are notification broadcasters, it is possible to register a `NotificationListener` with any available MBean. Registering a `NotificationListener` can be accomplished by calling the MBean's `addNotificationListener()` method.

However, in most cases it is preferable to register a listener using the MBean server's `addNotificationListener()` method. Using the `javax.management.MBeanServer` interface saves the trouble of looking up a particular MBean simply for registration purposes. For example, to listener defined in Creating a Notification Listener registers itself using:

```
rmbs = home.getMBeanServer();
oname = new WebLogicObjectName("TheLogBroadcaster",
        "LogBroadcasterRuntime",
        DOMAIN_NAME,
        SERVER_NAME);
```

```
rmbs.addNotificationListener(oname,
        listener,
        null,
        null);
```

Working with WebLogic Server Log Notifications

To receive log messages, client applications can use the standard JMX API to register a notification listener with the WebLogic Server `LogBroadcasterRuntimeMBean`, as shown in the previous examples. `LogBroadcasterRuntimeMBean` is responsible for generating notifications for log messages generated by a server.

All notifications broadcast by `LogBroadcasterRuntimeMBean` are of the type `WebLogicLogNotification`. There is only one `LogBroadcasterRuntimeMBean` per server, named `TheLogBroadcaster`.

The `LogBroadcasterRuntimeMBean` can be accessed using the mechanisms described in Accessing MBeans from MBeanHome.

Contents of a WebLogicLogNotification

All JMX notifications for a WebLogic Server log message contain the following fields:

- Type—the type field to which the log notification is mapped. This field has the format:

  weblogic.logMessage.*`subSystem`*.*messageID*

  where *`subSystem`* indicates the WebLogic Server subsystem that issued the log message, and *`messageID`* indicates the internal WebLogic Server message ID.

- Time stamp—the time at which the log message causing this notification was generated by the server.

- Sequence number.
- Message—contains the actual message body of the log message.
- User data—the user data field is not currently used.

All log notifications are of the type `WebLogicLogNotification`. This helper class provides getter methods for all individual fields of a log message. Using the `WebLogicLogNotification` class, a client application can easily filter log notifications based on their severity, user ID, subsystem, and other fields.

The following `NotificationFilter` example uses the `WebLogicLogNotification` class to select only messages of a specific message ID (111000) to be sent as notifications.

```
import javax.management.Notification;
import javax.management.NotificationFilter;
import javax.management.Notification.*;
....
public class WebLogicLogNotificationFilter implements NotificationFilter,
                                                 java.io.Serializable {
public WebLogicLogNotificationFilter() {
        subsystem = "";
    }
public boolean isNotificationEnabled(Notification notification) {
        if (!(notification instanceof WebLogicLogNotification)) {
            return false;
        }
        WebLogicLogNotification wln = (WebLogicLogNotification)notification;
        if (subsystem == null ||
            subsystem.equals("")) {
            return true;
        }
```

```
        StringTokenizer tokens = new StringTokenizer(wln.getType(), ".");
        tokens.nextToken();
        tokens.nextToken();
        return (tokens.nextToken().equals(subsystem));
    }
    public void setSubsystemFilter(String newSubsystem) {
        subsystem = newSubsystem;
    }
}
```

Example Notification Listeners for WebLogic Server Error Messages

Client applications can create various custom `NotificationListeners` that receive log messages as notifications and perform actions such as:

- E-mailing the WebLogic administrator with critical log messages
- Adding log messages to a datastore

The basic form of the notification listener differs little from the example shown in Creating a Notification Listener. Simply replace the printed messages in that example with the necessary JDBC calls or paging operations to perform in response to the notification.

4 Monitoring WebLogic Server MBeans

The following sections provide an overview of how to monitor WebLogic Server MBean attributes:

Overview

A WebLogic Server client can set up monitors to monitor one or more MBean attributes. The various types of monitors are defined in the JMX documentation for the package `javax.management.monitor`. Standard JMX monitors are:

- `CounterMonitor`, for observing integer attributes
- `GaugeMonitor`, for observing integer or floating point attributes
- `StringMonitor`, for observing string attributes

JMX monitors frequently act as notification broadcasters, to indicate when a certain condition has been met in a monitor. For this reason, monitoring systems generally include standard notification constructs, such as notification listeners and notification filters, which are registered with the monitor.

Setting Up Monitoring

The following is an example of how to set up a counter monitor for receiving JMX Notifications. Because this example also uses a notification listener to observe the monitor's notifications, some of the information builds from the examples in Using MBean Notifications.

Creating a Notification Listener

The example begins by building a notification listener, named `CounterListener`, that will receive notifications emitted from a JMX monitor:

```
import java.rmi.Remote;
import javax.management.Notification;
import javax.management.monitor.MonitorNotification;
import weblogic.management.RemoteNotificationListener;

public class CounterListener implements RemoteNotificationListener {
    String message;
    public void handleNotification(Notification notification ,Object obj) {
        System.out.println("\njavax.management.Notification");
        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("\njavax.management.monitor.MonitorNotification");
            System.out.println("  observed attr = " +
                               monitorNotification.getObservedAttribute() );
            System.out.println("   observed obj =" +
                               monitorNotification.getObservedObject() );
            System.out.println("  trigger value =" +
                               monitorNotification.getTrigger() + "\n");
            message = "Mbean: " + monitorNotification.getObservedAttribute() +
                      "\n" +
                      "Attribute: " + monitorNotification.getObservedObject() +
                      "\n" +
                      "Trigger Value: " + monitorNotification.getTrigger();
        }
    }
}
```

Instantiating the Listener and Monitor

The example monitor class instantiates both the listener and monitor object, and registers the monitor to observe the `ServerSecurityRuntime.InvalidLoginAttemptsTotalCount` attribute. This attribute indicates the number of failed logins to the server

When the invalid login attempts exceed a threshold value, the `handleNotification` method is invoked by the notification listener, `CounterListener.handleNotification()`.

The sample monitor code is as follows:

```
import java.rmi.RemoteException;
import java.util.Set;
import java.util.Iterator;
import javax.management.*;
import javax.management.AttributeChangeNotification;
import javax.management.AttributeChangeNotificationFilter;
import javax.management.monitor.CounterMonitor;
import javax.naming.*;
import weblogic.jndi.Environment;
import weblogic.management.*;
import weblogic.management.configuration.ServerMBean;
import weblogic.management.monitor.*;
import weblogic.management.runtime.ServerRuntimeMBean;

public class InvalidLoginMonitor {
    public static void main (String args[]) {
        // make sure there is a password argument
        if (args.length != 3) {
            System.out.println("Usage: java InvalidLoginMonitor " +
                               "<system password> " +
                               "<domain name> " +
                               "<server name>");
            return;
        }
        String url      = "t3://localhost:7001";
        String username = "system";
```

```
String password = args[0];
MBeanHome home  = null;
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 the MBeanHome " + home);
    RemoteMBeanServer rmbs = home.getMBeanServer();
    CounterMonitor monitor = new CounterMonitor();
    CounterListener listener = new CounterListener();
    WebLogicObjectName monitorObjectName =
           new WebLogicObjectName("MyCounter",
                                  "CounterMonitor",
                                  args[1],
                                  args[2]);
    WebLogicObjectName securityRtObjectName =
           new WebLogicObjectName("myserver",
                                  "ServerSecurityRuntime",
                                  args[1],
                                  args[2]);
    Long t = new Long(2);
    Long offset = new Long(0);
    monitor.setThreshold((Number)t);
    monitor.setNotify(true);
```

```
            monitor.setOffset((Number)offset);
            monitor.setObservedAttribute("InvalidLoginAttemptsTotalCount");
            monitor.setObservedObject(securityRtObjectName);
            monitor.addNotificationListener(listener, null, null);
            monitor.preRegister(rmbs, monitorObjectName);
            monitor.start();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Note: The above example uses the hard-coded server name, “myserver,” when creating the `WebLogicObjectName`. You must pass “myserver” as an argument when running this example or modify the code to use a different server name.

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 appropriate MBean API documentation.

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.

JDBCConnectionPoolRuntime MBean Attribute	**Typical Monitoring Application**
`LeakedConnectionCount`	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 `close()` call; it is important to monitor the total number of leaked connections, as a leaked connection cannot be used to fulfill later connection requests.
`ActiveConnectionsCurrentCount`	Notify a listener when the current number of active connections to a specified JDBC connection pool reaches a predefined threshold.
`ConnectionDelayTime`	Notify a listener when the average time to connect to a connection pool exceeds a predefined threshold.
`FailuresToReconnect`	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.