BEA WebLogic Server®

Monitoring and Managing with the J2EE Management APIs

Version 9.0
Revised: July 22, 2005

Contents

1. Using the J2EE Management APIs on WebLogic Server

Understanding the J2EE Management Model and APIs ii-1

JMO Hierarchy ii-2

JMO Object Names. ii-2

Optional Features ii-2

Accessing JMOs ii-3

The J2EE Management Model on WebLogic Server ii-3

Accessing the MEJB on WebLogic Server ii-3

Example: Querying Names of JMOs ii-4

CHAPTER 1

Using the J2EE Management APIs on WebLogic Server

The J2EE Management APIs enable a software developer to create a single Java program that can discover and browse resources, such as JDBC connection pools and deployed applications, on any J2EE Web application server. The APIs are part of the J2EE Management Specification, which requires all J2EE Web application servers to describe their resources in a standard data model.

The following sections describe how to use the J2EE Management APIs on WebLogic Server®:

- "Understanding the J2EE Management Model and APIs" on page 1-1
- "The J2EE Management Model on WebLogic Server" on page 1-3
- "Accessing the MEJB on WebLogic Server" on page 1-3

Understanding the J2EE Management Model and APIs

In the J2EE Management data model, each instance of a Web application server resource type is represented by a J2EE Managed Object (JMO). The J2EE Management Specification describes exactly which types of resources must be represented by a JMO. JMOs themselves contain only a limited set of attributes which are used to describe the location of the object in the data model.

Download the J2EE Management Specification from
http://jcp.org/aboutJava/communityprocess/final/jsr077/index.html.

JMO Hierarchy

The data model organizes JMOs hierarchically in a tree structure. The root JMO is `J2EEDomain`, which represents a collection of Web application server instances that are logically related. `J2EEDomain` contains the object names for all instances of the `J2EEServer` JMO, each of which represents a server instance in the collection.

Java applications can browse the hierarchy of JMOs recursively querying for object names and looking up the JMOs that are named by the query results.

JMO Object Names

Each JMO instance is identified by a unique object name of type `javax.management.ObjectName`. The names follow this pattern:

`*domain*:*name*=j2eeType=*value*,name=*value*,*parent-j2eeType[,property=value]**`

For example, `mydomain:J2EEtype=J2EEDomain,name=mydomain`

The J2EE Management Specification describes exactly which name/value pairs must be in the object names for each JMO type.

The object name for each child JMO contains name/value pairs from its parent JMO's object name. For example, if the JMO for a server instance is named `mydomain:j2eeType=J2EEServer,name=myserver`

then the JMO for a servlet that is part of an application deployed on that server instance would be named:

```
mydomain:J2EEApplication=myapplication,J2EEServer=myserver,WebModule=myapp
_mywebmodule,j2eeType=Servlet,name=myservlet_name
```

The name/value pairs can appear in any order.

Optional Features

The J2EE Management Specification, version 1.0, requires only that Web application servers implement JMOs and provide API access to the JMOs.

Optionally, the JMOs can be implemented to provide performance statistics, management operations, and to emit notifications when specified events occur.

Accessing JMOs

A Java application accesses the JMOs through `javax.management.j2ee.Management`, which is the remote interface for the Management Enterprise Java Bean (MEJB).

The J2EE Management Specification requires that the MEJB's home interface be registered in a server's JNIDI tree as `ejb.mgmt.MEJB`.

See the API Reference for the `javax.management.j2ee` package: http://java.sun.com/j2ee/1.4/docs/api/javax/management/j2ee/package-summary.html.

The J2EE Management Model on WebLogic Server

WebLogic Server 9.0 implements only the required features of the J2EE Management Specification, version 1.0. Therefore, the following limitations are in place:

- None of the JMOs provide performance statistics, management operations, or emit notifications.
- There are no mappings to the Common Information Model (CIM).
- There are no mappings to an SNMP Management Information Base (MIB).

The MEJB and JMOs are available only on the Administration Server. This is consistent with the J2EE Management Model, which assumes that most J2EE Web servers exist within some logically connected collection and that there is a central point within the collection for accessing or managing the server instances. From the Administration Server, a Java application can browse to the JMO that represents any resource on any server instance in the WebLogic Server domain.

Because WebLogic Server implements its JMOs as a wrapper for its MBeans, any changes in a WebLogic Server MBean that corresponds to a JMO is immediately available through the J2EE Management APIs.

For all JMO object names on WebLogic Server, the *`domain`*`:` portion of the object name corresponds to the name of the WebLogic Server domain.

Accessing the MEJB on WebLogic Server

To retrieve monitoring data through the MEJB:

1. Look up the `javax.management.j2ee.ManagementHome` interface through the Administration Servers JNDI tree under the name `ejb.mgmt.MEJB`.

2. Use `ManagementHome` to construct an instance of `javax.management.j2ee.Management`, which is the MEJB's remote interface.

Example: Querying Names of JMOs

The example class in accesses the MEJB for a WebLogic Server domain and invokes `javax.management.j2ee.Management.queryNames` method. This method returns the object name for all JMOs in the domain.

Listing 1-1 Querying Names of JMOs

```
import java.io.IOException;
import java.net.MalformedURLException;
import java.util.Iterator;
import java.util.Set;
import java.util.Properties;

import javax.management.j2ee.Management;
import javax.management.j2ee.ManagementHome;
import javax.management.AttributeNotFoundException;
import javax.management.InstanceNotFoundException;
import javax.management.ObjectName;
import javax.management.QueryExp;

import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;

import javax.ejb.CreateException;

public class GetJMONames {

   static String url = "t3://localhost:7001";
   static String user = "weblogic";
   static String password = "weblogic";

   public static void main(String[] args) {
      try {
         getAllJMONames();
      }catch(Exception e){
      System.out.println(e);
   }
}
public static Management getMEJBRemote()
    throws IOException, MalformedURLException,
    NamingException,CreateException
{
   Context context = getInitialContext();
   ManagementHome home = (ManagementHome)
       context.lookup("ejb.mgmt.MEJB");
   Management bean = home.create();
   return bean;
}
public static Context getInitialContext()
       throws NamingException
{
   Properties p = new Properties();
   p.put(Context.INITIAL_CONTEXT_FACTORY,
       "weblogic.jndi.WLInitialContextFactory");
   p.put(Context.PROVIDER_URL, url);
   if (user != null) {
      p.put(Context.SECURITY_PRINCIPAL, user);
      if (password == null)
         password = "";
         p.put(Context.SECURITY_CREDENTIALS, password);
      }
   return new InitialContext(p);
}
public static void getAllJMONames()
{
   try {
      Management rhome = getMEJBRemote();
      String string = "";
      ObjectName name = new ObjectName(string);
      QueryExp query = null;

      Set allNames = rhome.queryNames(name, query);
      Iterator nameIterator = allNames.iterator();
```

```
            while(nameIterator.hasNext()) {
               ObjectName on = (ObjectName)nameIterator.next();
               System.out.println(on.getCanonicalName() + "\n");
            }
         } catch (Exception ex) {
               ex.printStackTrace();
         }
      }
   }
```