

JBoss™

A Developer's Notebook™

Norman Richards
& Sam Griffith

- Installation
- App deployment
- Database config
- Security
- Logging
- Monitoring

O'REILLY®

Managing and Monitoring JBoss

No matter how well you design and implement an application, something always ends up going wrong. Your machine runs low on memory. The database stops accepting connections at 3:00 in the morning. Or maybe your application is running a bit more sluggish than normal. Things go wrong all the time. It's just a part of being a developer.

Fortunately, it is easy to get under the hood of JBoss and find out what is wrong. Because every service is loaded and managed by the JBoss microkernel, every service can be exposed and managed through the various JBoss management tools.

One interface to the service is the JMX Console. You've been using this simple web application throughout the book to inspect and manage JBoss services. Another web application is called the Web Console. It is a more advanced version of the JMX Console that provides basic monitoring and alerting functions. You'll learn how to use the Web Console in this chapter.

JBoss also provides a programmatic interface into management operations. You'll see how to write code that can remotely inspect and manage the server, and you'll learn how to use the twiddle application from the command line to access management features.

These capabilities might not seem quite as exciting as others we've talked about, but you'll appreciate having them when you are up late at night, working on a problem. So, let's dive in and see what's available.

Starting the Web Console

Since you've already been using the JMX Console, we'll skip right to its big brother, the Web Console. The Web Console is really just the standard JMX

In this chapter:

- *Starting the Web Console*
- *Monitoring Your Application*
- *Working with MBeans*
- *Monitoring MBeans*
- *Creating a Snapshot*
- *Creating a Monitor*
- *Creating an Email Alert*
- *Managing JBoss from the Command Line*

The Web Console application is named console-mgr.sar, and you can find it in the deploy/management directory.

Console you've been using, with the addition of an applet that provides a higher-level view of the server. Instead of showing the raw managed beans (MBeans), the Web Console provides multiple views into the server, giving you the ability to more quickly find things that interest you.

How do I do that?

The Web Console is a simple web application that lives under the web-console context root. To start it up, open your browser and go to `http://localhost:8080/web-console/`. The page loads a Java applet, and it can take a few seconds to completely start up. Once it does, you should see something pretty close to what is in Figure 8-1.

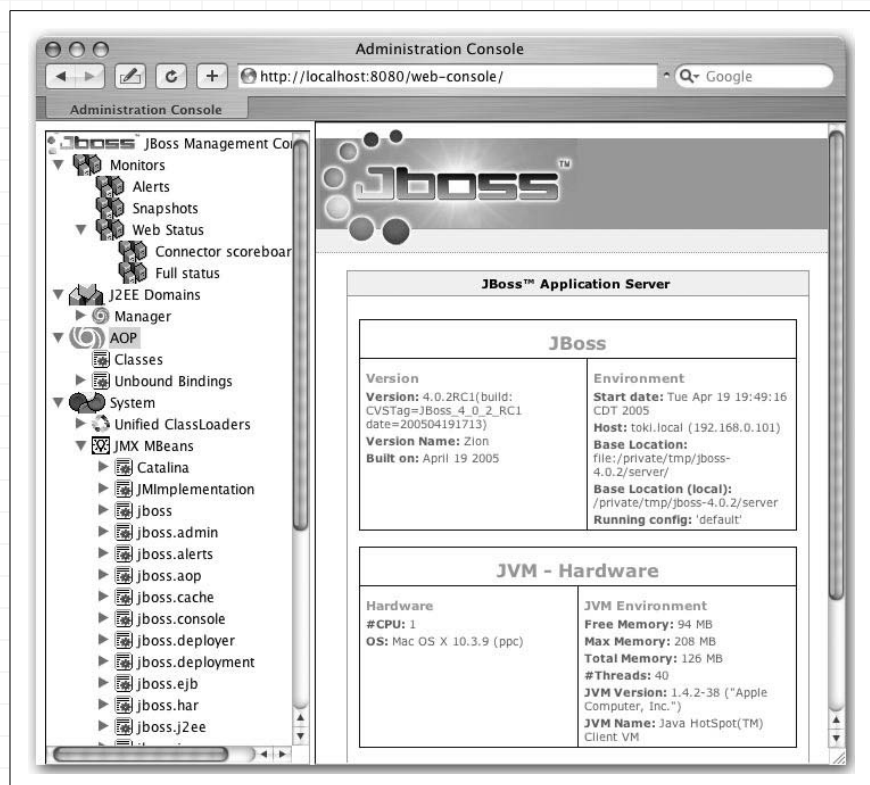


Figure 8-1. Web Console main page

There are two sections to the application. The lefthand side is the navigation panel. Clicking most items in the navigation panel will bring up a details page on the right. Clicking the JBoss logo will load the status page shown in Figure 8-1. If you right-click the logo, you'll have the

option to refresh the navigation panel and sync up with the server. You'll also find that the shutdown command is accessible from that menu.

The navigation panel provides four basic views into the system:

Monitors

The Monitors section provides a quick link to the alerts and snapshots you've created. Those will be empty now, but you'll see items there later, when we look at how to create them.

J2EE Domains

The J2EE Domains section shows all the deployed applications and services, organized by the name of the deployment package.

AOP

The AOP section shows the status of the AOP-based applications currently deployed.

System

The System section shows all the MBeans in the system, organized by domain.

Take a minute to familiarize yourself with how the Web Console operates before moving on. We're going to be covering a lot of ground here.

AOP stands for aspect-oriented programming. AOP application deployment is one of the many non-J2EE services available in JBoss. For more information, see <http://aop.jboss.org/>.

Monitoring Your Application

The most important thing in the server is your application, so we'll start our tour of JBoss's management and monitoring capabilities by looking at what JBoss can tell you about the state of currently running applications.

How do I do that?

If you drill down into the J2EE Domains section, you'll find an entry for the currently running JBoss instance. Inside that you'll find all the application packages deployed in the server: EAR files, WAR files, EJB jar files, and JBoss SAR (service archive) files. For deployments that are nested, such as with an EAR file, you can expand the parent archive to see the internal applications.

If the `ToDo` application is still deployed, you should find `todo.ear` near the top of the list. Selecting `todo.ear` will display some basic deployment information, including the `application.xml` deployment descriptor, in the details section on the right.

An EAR file is really just a container for other deployments, so there isn't much to see here. What you'll want to see are the contents of the EAR

file. When you expand *todo.ear*, you'll see the two primary application components: *todo.jar* and *todo.war*.

Selecting either of these will show deployment information for the archive, just as you saw for the EAR file. Even better, expanding either of these archives will show all the EJBs or servlets provided by that application. Figure 8-2 shows the view of the TaskMaster bean.

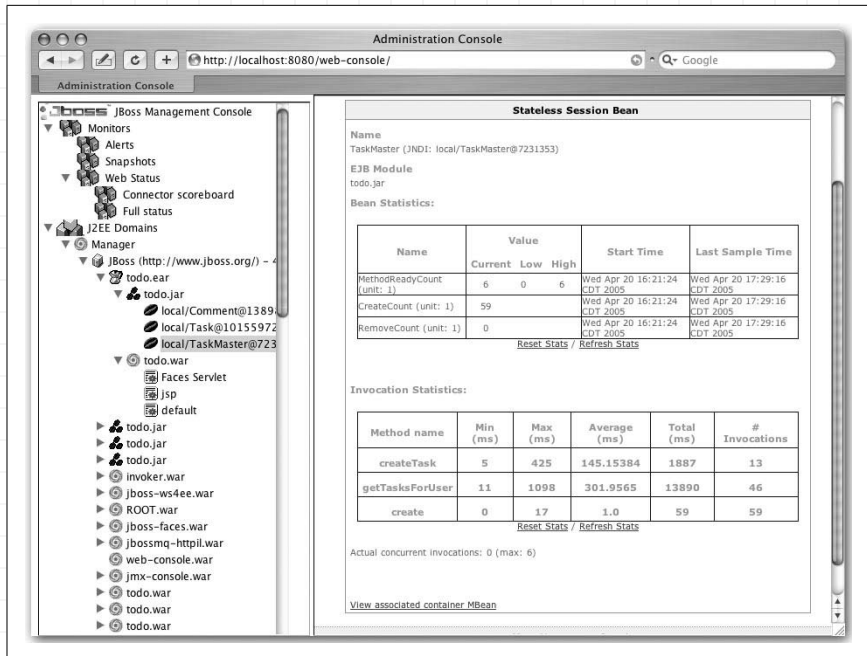


Figure 8-2. EJB statistics provided by the Web Console

The Bean Statistics section shows the status of the bean pool. JBoss currently has six instances of the TaskMaster bean waiting to serve requests. CreateCount shows the number of times a TaskMaster bean has been created, meaning an object was taken from the bean pool and made ready to service client requests.

Below that are the bean's invocation statistics. For each bean method, the total number of invocations processed is recorded, along with the minimum, maximum, and average processing time. You can see that, on average, it takes 145ms to create a new task, but 301ms to load all the tasks for a user. That load time seems a bit high, but it is actually skewed somewhat by the initial loading of data from the database. Once JBoss has cached the data, load times trend toward the minimum access time.

You can verify this by resetting the statistics, using the Reset Stats link, and accessing the application again.

You can find similar processing statistics for a servlet by selecting a servlet under *todo.war*. The *ToDo* application doesn't use servlets extensively. The only interesting servlet is *FacesServlet*, which processes JavaServer Faces requests. If you select it, you will see the minimum, maximum, and average response times for the servlet, as well as an invocation count.

If you are a J2EE guru, you might recognize this as the JSR-77 J2EE Management data.

What just happened?

You saw how to get basic usage statistics out of the server, down to the bean and servlet level. The available information isn't exhaustive, but it will help you locate performance problems for further investigation.

Working with MBeans

MBeans are the management interfaces to the services registered with the JBoss microkernel. Every service in JBoss is represented by an MBean, and you have the ability to jump in and interact with those services.

You've been using the JMX Console throughout the book to access MBeans. You've looked at datasource statistics, checked which classloader loaded a specific class, and even shut down the server through the JMX Console, so you should be feeling comfortable with general MBean operation. In this lab we'll look at MBeans through the eyes of the Web Console, which adds a few new twists to what you've seen so far.

Only attributes whose type has a JavaBeans property editor defined can be edited through the Web Console. You can define new property editors using PropertyEditor-ManagerService.

How do I do that?

To get to the MBeans, expand the JMX MBeans node in the System section of the navigation panel. You should find the same MBeans you saw in the JMX Console, but the tree view presentation in the Web Console makes it a bit easier to navigate.

Let's find the `jboss.system:type=ServerInfo` MBean we used earlier in the book. Find the `jboss.system` domain and expand it. You should see the MBeans shown in Figure 8-3.

When you select `jboss.system:type=ServerInfo`, the MBean page from the JMX Console will show up in the details panel. There are two parts to the management interface displayed: attributes and operations.

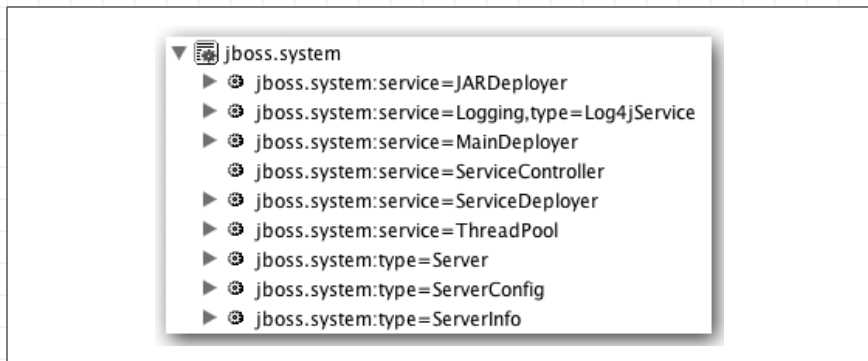


Figure 8-3. The MBeans in the jboss.system domain

Attributes typically represent the state or configuration of a service. Attributes are like fields on an object; they have a name, a type, and a value. They also have an access flag that specifies whether the attribute is readable or writable. Only writable attributes will show up as editable in the Web Console.

The ServerInfo MBean provides information about the state of the server, so all the attributes are read-only. Some of them are static, such as OS name and version. Others, such as the free memory and thread count values, will change regularly. You'll likely see those values change if you refresh the page.

To see an editable attribute, go back to the navigation panel and find the `jboss.system:service=Logging,type=log4j` MBean you used in Chapter 6. You can adjust the `RefreshPeriod` attribute to ask JBoss to check the `log4j.xml` file for changes more frequently. To change the value, enter the new value in the input box and click Apply Changes. After JBoss changes the configuration of the underlying service, the page will refresh and you will see the new value reflected on the MBean.

MBeans also provide management operations. These are just like methods on a regular Java object. Some operations are intended to send messages to the service. An example of this would be asking a connection pool to flush its pool, or changing the log levels in the log4j service.

Other operations are purely informational. The ServerInfo MBean provides a `listThreadDump` operation that shows all the threads in the JVM. To invoke the operation, click the Invoke button next to the operation's name.

```
Total Threads: 42
Total Thread Groups: 7
```

```
Thread Group: system : max priority:10, demon:false
Thread: Reference Handler : priority:10, demon:true
```

You'll see a nicer way to monitor a changing value shortly.

If you are running Java 1.5, JBoss will provide a full stack trace for each thread too!

Thread: Finalizer : priority:8, demon:true
Thread: Signal Dispatcher : priority:10, demon:true
Thread: CompileThread0 : priority:10, demon:true
Thread: RMI TCP Accept-1098 : priority:5, demon:true
Thread: RMI Reaper : priority:5, demon:false
Thread: GC Daemon : priority:2, demon:true
Thread: RMI TCP Accept-4444 : priority:5, demon:true

Thread Group: main : max priority:10, demon:false
Thread: DestroyJavaVM : priority:5, demon:false

Thread Group: jboss : max priority:10, demon:false
Thread: Thread-0 : priority:5, demon:true
Thread: ScannerThread : priority:5, demon:true
Thread: Thread-2 : priority:5, demon:true
Thread: PooledInvokerAcceptor#0-4445 : priority:5, demon:false
Thread: ContainerBackgroundProcessor[StandardEngine[jboss.web]] : priority:
5, demon:true
Thread: JBossMQ Cache Reference Softner : priority:5, demon:true
Thread: Thread-3 : priority:5, demon:true
Thread: HSQldb Timer @4038e2 : priority:5, demon:true
Thread: Thread-5 : priority:5, demon:true
Thread: JCA PoolFiller : priority:5, demon:false
Thread: TimeoutFactory : priority:5, demon:true
Thread: Thread-6 : priority:5, demon:true
Thread: JBossLifeThread : priority:5, demon:false
Thread: http-0.0.0.0-8080 : priority:5, demon:true
Thread: http-0.0.0.0-8080-1 : priority:5, demon:true
Thread: TP-Processor1 : priority:5, demon:true
Thread: TP-Processor2 : priority:5, demon:true
Thread: TP-Processor3 : priority:5, demon:true
Thread: TP-Processor4 : priority:5, demon:true
Thread: TP-Monitor : priority:5, demon:true
Thread: http-0.0.0.0-8080-2 : priority:5, demon:true
Thread: http-0.0.0.0-8080-3 : priority:5, demon:true
Thread: http-0.0.0.0-8080-4 : priority:5, demon:true
Thread: http-0.0.0.0-8080-5 : priority:5, demon:true
Thread: http-0.0.0.0-8080-6 : priority:5, demon:true
Thread: http-0.0.0.0-8080-7 : priority:5, demon:true
Thread: http-0.0.0.0-8080-8 : priority:5, demon:true
Thread: http-0.0.0.0-8080-9 : priority:5, demon:true
Thread: http-0.0.0.0-8080-10 : priority:5, demon:true

Thread Group: JBoss Pooled Threads : max priority:10, demon:false
Thread: ClassLoadingPool(2)-1 : priority:5, demon:true
Thread: WorkManager(3)-1 : priority:5, demon:true

Thread Group: System Threads : max priority:10, demon:false
Thread: JBoss System Threads(1)-1 : priority:5, demon:true

Thread Group: JBossMQ Server Threads : max priority:10, demon:false
Thread: UILServerILService Accept Thread : priority:5, demon:false

Thread Group: RMI Runtime : max priority:10, demon:false

Another good informational bean is the JNDIView MBean. Find the `jboss:service=JNDIView` MBean and invoke the `list` operation. You'll get a complete dump of the JNDI tree for the server, including the enterprise naming contexts for your EJBs. Here's the part of the output for *todo.jar*:

```
Ejb Module: todo.jar

java:comp namespace of the Task bean:
+- env (class: org.jnp.interfaces.NamingContext)

java:comp namespace of the Comment bean:
+- env (class: org.jnp.interfaces.NamingContext)

java:comp namespace of the TaskMaster bean:
+- env (class: org.jnp.interfaces.NamingContext)
  | +-.ejb (class: org.jnp.interfaces.NamingContext)
  | | +- CommentLocal[link -> local/Comment@13898049]
  | | (class: javax.naming.LinkRef)
  | | +- TaskLocal[link -> local/Task@10155972]
  | | (class: javax.naming.LinkRef)
```

You can clearly see how the JNDI references were resolved. The JNDI-View MBean is very useful when you are trying to find where an object is in the JNDI tree.

What just happened?

You saw how to interact with MBeans in the server through the Web Console. You can view and modify attributes and invoke management operations on any service in JBoss. There is very little you can't do through the MBean management interfaces.

Monitoring MBeans

What's the big deal with MBeans? Why are they interesting? MBeans provide a standardized form of metadata that management clients can use to inspect the state of the server. These self-describing services make it easy to write management applications, such as the Web Console, that can manage services regardless of their type.

In this environment, a management application can perform its actions on arbitrary services. It can monitor free memory in the system as easily as it can watch your connection pool size, and it can monitor services inside your application as easily as it monitors the JBoss-provided services.

With that in mind, we'll look at the ability to monitor a service, live, in JBoss.

How do I do that?

One of the things that can be valuable to watch in JBoss is the amount of free memory available to you. To do that, find the `jboss.system:type=ServerInfo` MBean in the navigation panel. Once you are there, expand the `ServerInfo` MBean so that you can see all the attributes. Right-click the `FreeMemory` attribute, and then bring up the context-sensitive menu, as shown in Figure 8-4.



Figure 8-4. Free Memory graph menu

Choose the `Graph` menu item. JBoss will begin graphing the amount of free memory over time. If you leave it open for a while, you should see a graph that looks like Figure 8-5, showing free memory growing and shrinking as you go through the normal Java garbage-collection cycles.

You can right-click the graph to change its look, and even save or print the current view.

What just happened?

You saw how to get a live chart of the free memory in the JBoss application server. Although the charts themselves are somewhat limited, they are easy to set up. If you are trying to monitor a queue or connection pool size, it is easy to watch the response of the desired metric as you are accessing the application.

If you exposed parts of your application through MBeans, you can easily monitor application-specific values.

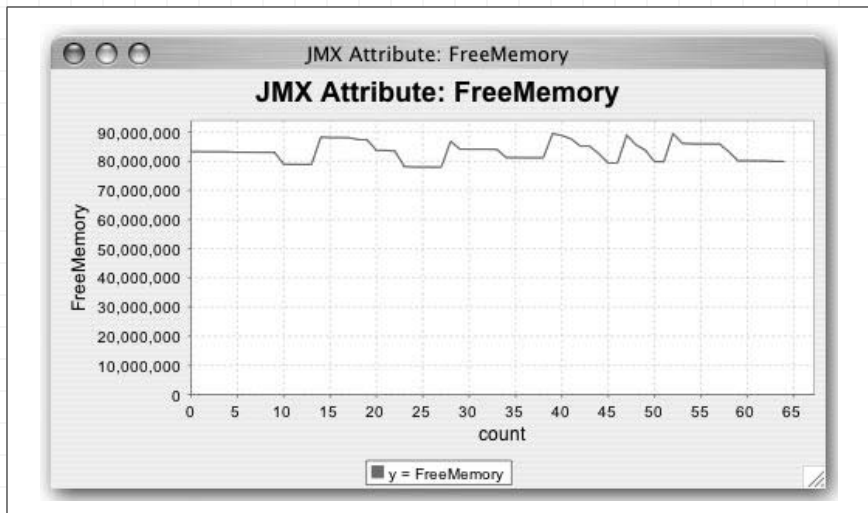


Figure 8-5. Free Memory graph

Creating a Snapshot

Live charts are powerful, but you aren't always sitting with the Web Console open, waiting for your application to run out of database connections. No, that always happens at 3:00 in the morning! Other problems manifest themselves slowly, over time.

JBoss gives you the ability to capture data over regular intervals. JBoss will collect observations of a specific MBean attribute. You can start or stop the data collection process according to your needs. Then you can return later to analyze the collected values.

Only numeric values can be graphed or monitored.

How do I do that?

To create a snapshot over time, choose the Create Snapshot menu item on the specific attribute you want to monitor. You'll be presented with a simple configuration form. You only need to enter the measurement time period. This is the time between measurements, in milliseconds. To monitor the value every two seconds, enter 2000 in the corresponding text field and click the Create button.

You can see the Manage Snapshot page in Figure 8-6.

To start the snapshot of memory usage, click the Start Snapshot button. Wait a few seconds for JBoss to collect measurements, and then click the Graph Dataset button. That will give you a graph like the one shown in Figure 8-7.

Manage Snapshot

Monitor Name	<input type="text" value="FreeMemory Snapshot"/>	<small>The name of the monitor and how it will be references within web console</small>
Monitor's Object Name	<input type="text" value="jboss.snapshot:name=FreeMemory Snapshot"/>	<small>The MBean javax.management.ObjectName</small>
Object Name	<input type="text" value="jboss.system:type=ServerInfo"/>	<small>The MBean javax.management.ObjectName of the MBean you are monitoring</small>
Attribute	<input type="text" value="FreeMemory"/>	<small>The MBean Attribute you are monitoring</small>
Time Period	<input type="text" value="2000"/>	<small>How often should threshold be tested.</small>

Figure 8-6. Manage snapshot

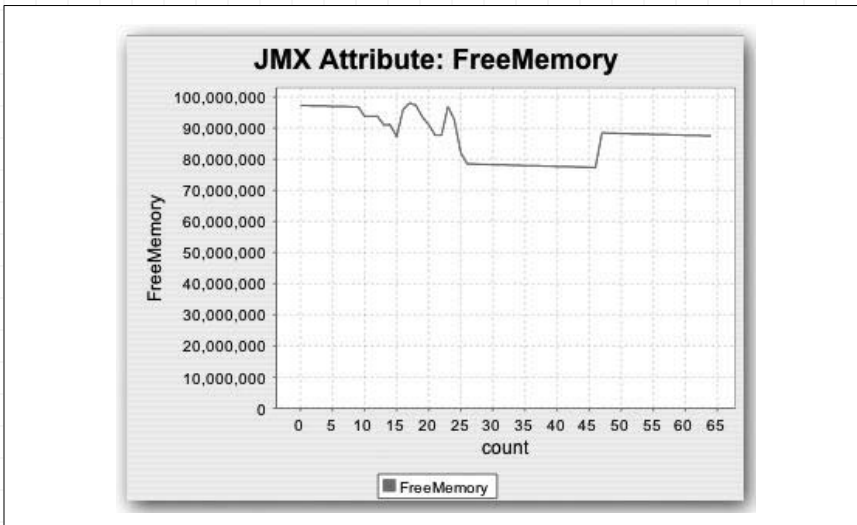


Figure 8-7. Snapshot graph

Notice that the graph isn't a live graph. It's a fixed graph of the data collected up to that point. This graph will give you a good idea of what happened over the specified time period. If you need better analysis capabilities, click Show Dataset, which displays all the collected data. You can select the data in the dataset view, and copy and paste it to an application such as Excel that can perform more sophisticated numerical analysis.