

JBoss™

A Developer's Notebook™

Norman Richards
& Sam Griffith

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

O'REILLY®

Creating a Monitor

A snapshot is good for collecting information about one aspect of your server over time, to gain an understanding of your server's behavior, and to find anomalies. However, when you are looking for a specific problem, you generally want to know about it as soon as that problem has occurred. Then you can go to a long-running snapshot and examine how the server behaved during the time leading up to the problem.

The Web Console gives you the ability to create an alert to monitor an MBean attribute value for a specific condition. When that condition is met, JBoss triggers a notification event to alert you to the condition. In this lab, you will create a monitor on the free memory of the JBoss application server.

How do I do that?

To create the alert, find the `FreeMemory` attribute of the `jboss.system:type=ServerInfo` MBean one more time and choose `Create Monitor` from the menu. The details section of the Web Console will contain the monitor creation form shown in Figure 8-8.

The screenshot shows a web form titled "Create Threshold MBean Monitor". It contains the following fields and options:

- Monitor Name:** FreeMemoryMonitor. Description: The name of the monitor and how it will be references within web console.
- Object Name:** jboss.system:type=ServerInfo. Description: The MBean javax.management.ObjectName of the MBean you are monitoring.
- Attribute:** FreeMemory. Description: The MBean Attribute you are monitoring.
- Threshold:** 50000000. Description: The value that will trigger an alert when the Comparison Equation is reached for the attribute value.
- Time Period:** 1000. Description: How often should threshold be tested.
- Comparison Equation:** < +/- (selected). Description: Boolean expression to use when testing threshold hit.
- Persisted:** . Description: Should this monitor be created for next JBoss reboot?
- Enable Monitor:** . Description: Should this monitor be enabled.
- Alerts:** Console Alert. Description: Alert Listeners to trigger.

A "Create" button is located at the bottom left of the form.

Figure 8-8. Creating an alert

JBoss has filled in the MBean name and attribute name. The Threshold and Comparison Equation values determine the trigger for the event. Monitors are purely numeric and can be compared to any fixed value. This monitor will trigger an alert when free memory goes below

50,000,00, which is approximately 50MB. JBoss checks the monitor every Time Period milliseconds.

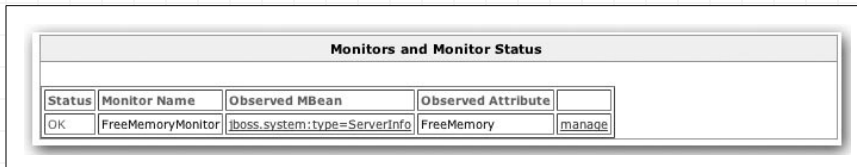
If the alert is triggered, JBoss will send an alert to the alert handlers selected in the Alerts select box. Right now only one alert handler is listed: Console Alert. It sends an alert message to the console log. You'll see some other alert options later.

Select the Persist Changes and Enable Monitor boxes to make sure the alert will be persisted and enabled before clicking the Create button.

You've created a monitor to generate an alert when free memory falls below a certain threshold, but where did the monitor go? All the monitors are listed in the Alerts section of the navigation pane. However, you may need to refresh the tree to see the Alerts section.

Right-click the JBoss logo to refresh the tree.

When you select the Alerts node, you'll see a list of all active monitors. The list looks like that shown in Figure 8-9.



Status	Monitor Name	Observed MBean	Observed Attribute	
OK	FreeMemoryMonitor	jboss.system:type=ServerInfo	FreeMemory	manage

Figure 8-9. The list of active monitors

The green OK status means the monitor has not yet triggered an alert.

To prove that our alert works, we need to force JBoss to trigger it. To do that, we need a way to consume lots of memory, driving the free memory below the 50MB threshold. We've created a simple JSP to do that:

```
<%@ page import="java.util.*" %>

<h1> Memory Eater </h1>
<%! int count; %>
<%
    try {
        ArrayList list = new ArrayList();
        while (true) {
            list.add(new Object());
            count++;
        }

    } catch (Throwable t) {
    }

%>

Created <%= count %> objects...
```

The console alert listener logs to the org.jboss.monitor.alerts.ConsoleAlertListener category.

This JSP continually creates objects until memory runs so low that an exception is thrown. Copy this JSP directly into *ROOT.war* in the *deploy/jbossweb-tomcat55.sar* directory to make it available.

If you name the file *memory.jsp*, you can access it at <http://localhost:8080/memory.jsp>. When you access it, the server will spin for a while as it eats up memory. Unless you have a large amount of memory and a slow CPU, it shouldn't take more than about 15 seconds to deplete the free memory in the JVM. When the page finally loads, you should see that the alert shows up in your console log as expected:

```
14:16:39,171 INFO [ConsoleAlertListener] FreeMemoryMonitor was triggered for attribute FreeMemory.
```

If you return to the list of monitors, you will see that the status has changed from a green OK to a red ALERT. If you click Manage, you'll see the monitor with the Triggered Value filled in. This is the value at the time the threshold was exceeded. Figure 8-10 shows the value in this case.

The screenshot shows a web console window titled "Manage Threshold MBean Monitor". It contains several configuration fields with labels and descriptions:

- Monitor Name:** FreeMemoryMonitor (The name of the monitor and how it will be referenced within web console)
- Monitor's Object Name:** jboss.monitor.service=FreeMemoryMonitor (The MBean javax.management.ObjectName)
- Object Name:** jboss.system.type=ServerInfo (The MBean javax.management.ObjectName of the MBean you are monitoring)
- Attribute:** FreeMemory (The MBean Attribute you are monitoring)
- Triggered Value:** 39227288 (The attribute value that triggered the threshold.)
- Threshold:** 50000000 (The value that will trigger an alert when the Comparison Equation is reached for the attribute value)
- Time Period:** 1000 (How often should threshold be tested.)
- Comparison Equation:** < +/- (Boolean expression to use when testing threshold hit.)
- Persist Changes:** (Should changes be reflected in deployment file.)
- Enable Monitor:** (Should this monitor be enabled.)
- Alerts:** Console Alert (Alert Listeners to trigger.)

At the bottom of the form are three buttons: "Update Monitor", "Remove Monitor", and "Clear Alert".

Figure 8-10. The monitor after it fires an alert

As you can see, the value was well below 50MB by the time JBoss noticed. The JSP was consuming memory very rapidly, and even though the monitor was very aggressive in checking the value every second, it didn't notice until the measure was well below the threshold.

Normally, the values you are monitoring won't change so rapidly, but you do need to be careful to pick a time period for the monitor that is likely to

pick up the change. If we monitored free memory every minute, the event may have come and gone by the time JBoss checked the value again, and you would have missed the event.

What just happened?

You created a monitor and triggered an alert using the Web Console. There's not much more to say, but we should point out that once a monitor has issued an alert, it remains in that state until you clear it. The Clear Alert button clears the alert and restarts the monitor.

Creating an Email Alert

When you created the monitor, you chose to send the alert to the console alert listener, and JBoss logged the alert as an INFO message waiting to be read. Alert listeners don't need to be so passive. In this lab, you'll configure alerts to be sent by email.

If you need an email server for testing, you can try the JBoss Mail Server.

How do I do that?

You send email alerts using the email service, so first you'll need to make sure the email service is properly configured. Look in the *mail-service.xml* file. If you have a local mail server that relays without authentication, you can put the hostname of the server in the `mail.smtp.host` property and be done with the mail service.

If you do need to authenticate to the mail server, you will need to set the User and Password attributes with the required login, and set `mail.smtp.auth` to true:

```
<mbean code="org.jboss.mail.MailService"
  name="jboss:service=Mail">
  <attribute name="JNDIName">java:/Mail</attribute>
  <attribute name="User">username</attribute>
  <attribute name="Password">my password</attribute>
  <attribute name="Configuration">
    <configuration>
      <!-- ... -->
      <property name="mail.smtp.host"
        value="my.mail.server"/>
      <property name="mail.smtp.auth"
        value="true" />
    </configuration>
  </attribute>
</mbean>
```

Any standard JavaMail configuration options can be placed here. If your mail server is particularly demanding, you might need to consult the JavaMail documentation at <http://java.sun.com/products/javamail/> to find the correct properties. Here's an example of a JavaMail configuration to connect to Gmail:

GMail is unusual in that it requires you to connect securely using SSL.

```
<property name="mail.smtp.host" value="smtp.gmail.com"/>
<property name="mail.smtp.port" value="465"/>
<property name="mail.smtp.auth" value="true" />
<property name="mail.smtp.starttls.enable" value="true" />
<property name="mail.smtp.socketFactory.port" value="465" />
<property name="mail.smtp.socketFactory.fallback" value="false" />
<property name="mail.smtp.socketFactory.class" value="javax.net.ssl.SSLSocketFactory" />
```

That gets the mail service running. You can configure the email alert listener by uncommenting the EmailAlertListener MBean in *monitoring-service.xml*. You'll need to set the To and From addresses for the email message as shown here:

```
<mbean code="org.jboss.monitor.alerts.EmailAlertListener"
name="jboss.alerts:service=EmailAlertListener">
<depends>jboss:service=Mail</depends>
<attribute name="MessageTemplate"><![CDATA[
%(MONITOR_NAME) was triggered for attribute %(ATTRIBUTE).
]]></attribute>
<attribute name="AlertName">Email Alert</attribute>
<attribute name="To">jbossnotebook@gmail.com</attribute>
<attribute name="From">jbossnotebook@gmail.com</attribute>
<attribute name="ReplyTo">jbossnotebook@gmail.com</attribute>
<attribute name="SubjectTemplate">
<![CDATA[[jboss-alert] %(MONITOR_NAME)]]>
</attribute>
</mbean>
```

If you have problems sending mail, set mail.smtp.debug to true to see the connection attempts.

Once you have made these changes, the free memory monitor will show an Email Alert option in the Alerts box. After you add the email listener to the monitor, you should access *memory.jsp* again to trigger the alert. If your email service is configured properly, you should soon have an email alert in your inbox. Figure 8-11 shows an email alert viewed from within Gmail.

It's not the most informative of messages, but it works. You can change the message title and body by changing the MessageTemplate and SubjectTemplate attributes on the EmailAlertListener MBean.

What just happened?

You configured a monitor to send an alert message by email. To do that you had to first enable the mail service to know what mail server to use

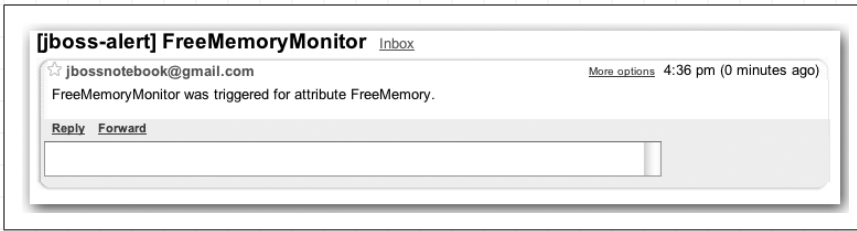


Figure 8-11. An email alert viewed in Gmail

to send mail. Then you activated the email alert listener service. The Web Console immediately saw the email alert listener and let you connect your monitor to it.

If email isn't good enough, you can write your own listener to process alerts.

Managing JBoss from the Command Line

The Web Console gives you access to all the services in JBoss, but it does have one huge limitation. It is a web application that requires that an interactive user do the pointing and clicking. That's fine most of the time, but sometimes you'll want to automate access to the server.

JBoss provides a very simple command-line application, called `twiddle`, that lets you query MBeans, get and set attribute values, and even invoke operations. If you need to automate access to JBoss, `twiddle` is the easiest and best tool to use.

How do I do that?

The `twiddle` script sits in the `bin` directory, next to the startup and shutdown scripts. You can run it from any terminal window, and it's easy to use. The `get` command lets you query an MBean by name. Pass in the name of the MBean and a list of attributes to retrieve:

```
[bin]$ ./twiddle.sh get jboss.system:type=ServerInfo FreeMemory ActiveThreadCount
FreeMemory=90167064
ActiveThreadCount=46
```

If you don't specify any attributes, you'll get all of them:

```
[bin]$ ./twiddle.sh get jboss.system:type=ServerInfo
HostAddress=192.168.0.101
AvailableProcessors=1
OSArch=ppc
OSVersion=10.3.9
HostName=toki.local
JavaVendor=Apple Computer, Inc.
JavaVMName=Java HotSpot(TM) Client VM
```

```
FreeMemory=90898472
ActiveThreadGroupCount=6
TotalMemory=132775936
JavaVMVersion=1.4.2-38
ActiveThreadCount=45
JavaVMVendor="Apple Computer, Inc."
OSName=Mac OS X
JavaVersion=1.4.2_05
MaxMemory=218103808
```

If you need to use a value in a script, use the `--noprefix` flag:

```
[bin]$ ./twiddle.sh get --noprefix jboss.system:type=ServerInfo FreeMemory
92063536
```

The `set` command sets an attribute value on an MBean. This command sets the connection pool size for `DefaultDS` to 25:

```
[bin]$ ./twiddle.sh set jboss.jca:name=DefaultDS,service=ManagedConnectionPool \
MaxSize 25
MaxSize=25
```

You can invoke MBean operations using the `invoke` command. The output will be the return value, if any, of the method. This command asks for garbage collection to be run:

```
[bin]$ ./twiddle.sh invoke jboss.system:type=Server runGarbageCollector
```

If you check the console log, you will see the results of running garbage collection:

```
18:28:57,779 INFO [Server] Total/free memory: 132775936/91869984
18:28:59,429 INFO [Server] Hinted to the JVM to run garbage collection
18:28:59,431 INFO [Server] Total/free memory: 132775936/92366288
```

The following example invokes the `clearAlert` method on the free memory monitor:

```
[bin]$ ./twiddle.sh invoke jboss.monitor:service=FreeMemoryMonitor \
clearAlert
```

If you are on a remote machine, add the `-s` option to specify the host you are trying to talk to:

```
[bin]$ ./twiddle.sh -s hostname invoke jboss.system:type=Server shutdown
```

You can run that command from any remote machine to shut down your JBoss instance.

What just happened?

You accessed the MBeans on your JBoss instance from a remote machine using the `twiddle` command-line script. `twiddle` gives you fast, scriptable access to any JBoss instance.