Tomcat for Apache
EFFICIENT, FLEXIBLE AND OPEN SOURCE, CHOOSE ANY THREE

DEAN WILSON AND KIM HAWTIN

Background

Jakarta is a selection of open source, Java-related technologies, such as XML parsers, XSLT processors, regular expressions and portal management.

Due to many projects being implemented in Java, they're mostly OS independent, which allows their widespread use.

Apache is the Web server of choice. It delivers the content of over half of the sites on the Web, but until recently it lacked an integrated way of serving dynamically generated Java content. With the rising popularity of the Tomcat server (from the Apache-Jakarta project) this need has begun to be met.

Tomcat is a product of the Apache-Jakarta project that functions as a Java Servlet and Java Server pages engine and is one of the most up-to-date implementations of the Java servlet. A Java servlet is a small application that receives and responds to http requests.

Although Tomcat can be run as a stand-alone servlet engine, its true power becomes clear when coupled with Apache to help cut down:

- Running the built in Tomcat Web server in addition to Apache.
- Forcing the user to enter non-standard details such as the port number in each request.
- Doubling the effort required to administer the servers.
- Exposing more potential security risks.

Using the two projects together means that you can harness Apaches features such as:

- Multiple virtual hosts.
- Fast delivery of static content (Tomcat is optimised for servlet processing).
- An almost infinite number of configuration options for multiple virtual Web servers that Apache users require.

Installing Java

This tutorial assumes that you already have an Apache site that you would like to expand by adding Java servlets, this will be achieved by adding the Java Software Development Kit (J2SDK) and Tomcat to the established site. It will not require a recompilation of your existing Apache set up if your Apache allows the use of **DSOs.** The Java Development Kit is available from *http://java.sun.com.* Download the newest version from this site (J2SDK at the time of going to press).

Although the J2SDK is free, you do have to sign an agreement before you can get the software. Once you have this on the computer that you have Apache installed on, you'll need to install the J2SDK. This procedure varies between Red Hat and Debian-based distributions; here we will cover the Red Hat install.

Installing the Java SDK is simple. You should run the following command:

```
# sh j2sdk-1_3_1-linux-i386-rpm.bin
```

You'll be confronted with a licensing agreement to be accepted in order to install. If you agree to the license the package will decompress, leaving a ready-for-install rpm. Next, login as root and install the rpm with a command similar to:

```
# rpm -i jdk-1.3.1.i386.rpm
```

1.3.1 in the filename will vary depending upon the version of the J2SDK you want to install.

Web sites used to be a way of claiming a personal piece of the new frontier, a number of html pages and a CGI script made a website. These days with the advent of P2P sites and e-Commerce solutions Java is coming into the limelight both on its own merits and as a rival to Microsoft's .net platform. Java's success in this field is based upon its ability to act as middleware between different vendors software and Open Source projects such as Apache, Enhydra, Cocoon and Tomcat.

*What is a **DSO**?*

A DSO is a Dynamic Shared Object, in the past whenever you wished to add new functionality to an existing Apache web server install you would be required to reconfigure, re-compile and perform a fresh install of the Apache binaries. With the inclusion of DSO's in the Apache server this process has been simplified into compiling an external module into a DSO that Apache can pick up and incorporate at run time without the base web server needing to be recompiled. If you are running a Debian based distribution then you'll need the Apache-dev package, if its Red Hat based then you'll need the web server source from Apache.

Next, add the Java binaries to your path. You can do this on a per user basis or a system wide level. To add Java to the path for a single user edit *.bash_profile* in their home directory. Check for the line that has PATH=$PATH at the start and add the path of the Java runtime binaries to the PATH environment variable.

The second step involves adding a line to set the JAVA_HOME environment variable. Both variables need to be explicitly exported. A simplified example is as follows:

```
PATH=$PATH:/usr/java/jdk1.3.0_02/bin/java
JAVA_HOME=/usr/java/jdk1.3.0_02/
export PATH JAVA_HOME
```

jdk1.3.0_02 is the version of the software downloaded. To do this on a system wide level you need to enter the same changes as before but in the */etc/profile* file. We recommend system-level application so that they apply to all users. This will make running Tomcat a much simpler task. You'll need to logout and log back in for the environmental changes to take effect.

You can test that the Java binaries installed correctly by entering the simple test program (Boxout HelloWorld.java). Be sure to make the name of the file the same as the name you entered in the public class line at the top of the sample code otherwise the java compiler will give an error similar to "class HellWorld is public, should be declared in a file named HellWorld.java". To test the code you will need to type:

```
# javac HelloWorld.java
```

The compiler will then generate a file named "HelloWorld.class", if you then type:

```
# java HelloWorld
```

You'll see "Hello World" printed on the terminal. This means the J2SDK was successfully installed. If an error occurs work through the steps again.

Installing Tomcat

The precompiled version of Tomcat is at *http://jakarta.apache.org/ tomcat/index.html* Download the newest stable binary; at the time of writing this was Tomcat 3.2.2. Move the compressed tar file to "/usr/local/". Extract as shown. This creates *jakarta-tomcat-3.2.2*. Then create a symlink in the *jdk* external library directory (below) for the Tomcat servlet jar file and run the Tomcat startup script :

```
# mv jakarta-tomcat-3.2.2.tar.gz /usr/local/
# tar -zxvf jakarta-tomcat-3.2.2.tar.gz
# ln -s /usr/local/jakarta-tomcat-3.2.2/lib/servlet.jar
/usr/java/jdk1.3.0_02/jre/lib/ext/servlet.jar
# cd jakarta-tomcat-3.2.2/bin
# ./startup.sh
```

Once you have run the *startup.sh* script Tomcat will output diagnostic messages to the terminal. Tomcat will run in the background as a daemon but continue to print messages to that terminal. Any error messages at this stage mean you've set up the JAVA_HOME or the PATH statements in the user profiles incorrectly. To restart Tomcat run *./shutdown.sh*, then repeat the previous steps. Then run *startup.sh* again. Open up a browser and point it to *http://localhost:8080/index.html.* If Tomcat is installed correctly you should see a Tomcat test page.

In order to test that your environment is correctly set up and you are ready to move to the final stage of integrating Apache with Tomcat you should enter and run the sample servlet presented in the Sidebar. Open up your editor of choice and enter the sample code remembering to name the file *ServletHello.java*. You should then compile the java source:

```
# javac ServletHello.java
```

The compiler should create the *ServletHello.class* file. You then need to copy the class file to the Tomcat directory structure so that the Tomcat server can execute it when the request HTTP request is issued from your webbrowser. You should execute the following:

Resources

For the best place to learn more about the Apache and Jakarta projects:
http://www.apache.org/
http://jakarta.apache.org/
For more information on Java Servlets see:
http://java.sun.com/
For ideas on how to use Java Servlets or information on Tomcat configuration see:
http://www.oreilly.com/catalog/jservlet2/
http://www.jguru.com/

Listing 1:HelloWorld.java

```
public class HelloWorld
{
  public static void main(String[] args)
  {
    System.out.println("Hello Java World");
  }
}
```

Listing 2:ServletHello.java

```
/* ServletHello.java: Hello world example servlet */
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
//Import all required class's
public class ServletHello extends HttpServlet
  {
  public void doGet(HttpServletRequest request, HttpServletResponse response)
  throws IOException, ServletException
    {
    response.setContentType("text/html");
    //This sets the type of the response
    PrintWriter toClient = response.getWriter();
    //This sets the output character stream
    toClient.println("<html>");
    toClient.println("<head>");
    toClient.println("<title> This is a test page</title>");
    toClient.println("</head>");
    toClient.println("<body>");
    toClient.println("<p>Welcome to tomcat!</p>");
    toClient.println("</body></html>");
    }
}
```

```
# cp ServletHello.class ➥
/usr/local/jakarta-tomcat-3.2.2/webapps/exa➥
mples/WEB-INF/classes/
```

Point your web browser at the following:

```
http://localhost:8080/examples/servlet/Servl➥
etHello
```

You should be rewarded with a page with the following text "Welcome to Tomcat!", if the request succeeded. If you get a 'file not found error' or 404, then check the output of Tomcat on the terminal you started it on. An example of this is:

```
Ctx( /examples ): 404 R( /examples + /servlet➥
/ServletHell + null) null
```

ServletHell in this case is probably a typo. Check the URL that you entered in your browser. If these errors persist compare the name of the Java class you requested in your browser to the name of the class file you copied into the Tomcat directory tree.

Configuring Apache

In order for the Apache server to be able to communicate with the Tomcat Server you need to download a DSO called *mod_jk.so*. This is available as part of the Jakarta project and can be downloaded from:

```
http://jakarta.apache.org/builds/jakarta-tom➥
cat/release/v3.2.2/bin/linux/i386/"
```

You should have two options, a *mod_jk.so-eapi* and *mod_jk.so-noeapi*. Download the *mod_jk.so-eapi* version. This then needs to be moved to the correct place under the Apache directory tree. The Tomcat and Apache configuration files need to be updated to reflect the new functionality.

```
# mv mod_jk.so-eapi /usr/lib/apache/mod_jk.so
```

By moving the module to this location Apache knows that it needs to load it when restarted. You then need to make the changes to the Apache and Tomcat configuration files before restarting the Apache server. In *httpd.conf* (Which you can find by issuing a *locate httpd.conf*) you need to make the following amendments. At the end of the *LoadModule* section you should put the following line:

```
LoadModule jk_module modules/mod_jk.so
```

Add to the end of the AddModule section:

```
AddModule mod_jk.c
<IfModule mod_jk.c>
 JkWorkersFile /usr/local/src/jakarta-tomcat➥
-3.2.2/conf/workers.properties
 JkLogFile logs/jk.log
 JkLogLevel warn
 JkMount /examples/* ajp13
</IfModule>
```

You need to ensure that the path to the *workers.properties* file is correct, you can check this by issuing a *locate workers.properties* command and editing the path in the configuration file as required. You also need to create a log file for the jk module, this can be done with a simple:

```
touch /usr/local/src/jakarta-tomcat-3.2.2/lo➥
gs/jk.log
```

Configuring Tomcat

For each **context** in your Tomcat configuration file you need to add a JkMount line to your *httpd.conf* file. This is so that Apache can forward requests to the correct Tomcat handler for processing. Apache supports the Ajp12 protocol used with JServ. Tomcat uses the new Ajp13 protocol for new functionality, and so you need to add support for the new protocol.

In the configuration file for Tomcat, *server.xml* you need to add after the existing Ajp12 section the following request handler:

```
 <!-- Apache AJP13 support. This is also use➥
d to shut down tomcat. -->
 <Connector className="org.apache.tomcat.se➥
rvice.PoolTcpConnector">
    <Parameter name="handler"
value="org.apache.tomcat.service.connector.➥
Ajp13ConnectionHandler"/>
    <Parameter name="port" value="8009"/>
 </Connector>
```

After adding these lines to your configuration file you will need to restart the Tomcat server with the *shutdown.sh* and *startup.sh* scripts. Once the Tomcat server has successfully re-initialised you can open your web browser and point it at:

```
http://localhost/examples/servlet/ServletHello
```

You should see the servlet output sent straight to your browser from Apache.

Server Initialisation

So that Apache and Tomcat start and stop like other servers, we recommend that you add Tomcat to the Apache initialisation script or create a separate initialisation script .

Conclusion

We've only covered the basic Tomcat and Apache set-ups. Servlets and Java Server Pages are powerful tools for producing dynamic content for Web-based applications. The joint use of Tomcat and Apache gives all the strengths of Apache's fast static content delivery and near infinite configuration options and Tomcats flexible content delivery to produce an open source solution that rivals the best commercial offerings. ■

***Contexts** in Tomcat*

Tomcat can manage multiple web applications. Each web application is a collection of files such as Java Servlets, html, JSP files and other resources that are required for the web application to function. Each web application can be deployed separately, in a context. This is useful to test your Java servlet with different versions of supporting Jar files. For example using several different XML parsers, or XSLT processors.

■

About the Authors:

Dean Wilson: Professional Developer, using Linux as a serious alternative to commercial offerings.
Kim Hawtin: UNIX Systems Administrator, dabbling in all things network-related, preferably without wires.