CHAPTER 1

Jakarta Tomcat

IN THIS CHAPTER, we

- Introduce the Jakarta Tomcat server
- Describe the Jakarta Tomcat architecture
- Define Java Web applications
- Discuss the requirements for installing and configuring Tomcat
- Describe the steps of installing and configuring Tomcat
- Test your Tomcat installation

The Jakarta Tomcat Server

The Jakarta Tomcat server is an open source, Java-based Web application container that was created to run servlet and JavaServer Page Web applications. It exists under the Apache-Jakarta subproject, where it is supported and enhanced by a group of volunteers from the open source Java community.

The Tomcat server has become the reference implementation for both the servlet and JSP specifications. It is very stable and has all of the features of a commercial Web application container. Tomcat also provides additional functionality that makes it a great choice for developing a complete Web application solution. Some of the additional features provided by Tomcat—other than being open source and free—include the Tomcat Manager application, specialized realm implementations, and Tomcat valves.

The Tomcat Manager Web Application

The Tomcat Manager Web application is packaged with the Tomcat server. It is installed in the context path of `/manager` and provides the basic functionality to manage Web applications running in the Tomcat server. Some of the provided

functionality includes the ability to install, start, stop, remove, and report on Web applications.

Specialized Realm Implementations

Tomcat provides two methods for protecting resources. The first `authentication` implementation provided with Tomcat is a memory realm. The class that implements the memory realm is `org.apache.catalina.realm.MemoryRealm`. The `MemoryRealm` class uses a simple XML file as a container of users.

The second authentication implementation included with Tomcat is a JDBC realm. A `JDBCRealm` class is much like the `MemoryRealm`, with the exception of where it stores its collection of users. A `JDBCRealm` stores all of its users in a user-defined, JDBC-compliant database.

Tomcat Valves

Tomcat valves are a new technology introduced with Tomcat 4. They allow you to associate an instance of a Java class with a particular Catalina container. Valves are proprietary to Tomcat and cannot, at this time, be used in a different servlet/JSP container.

Further Information

Throughout this text, we discuss all of these Tomcat-specific features and some other features that are common to all Web application containers. More information about Tomcat can be found on its homepage:

`http://jakarta.apache.org/tomcat/index.html`

Figure 1-1 shows the Tomcat homepage.

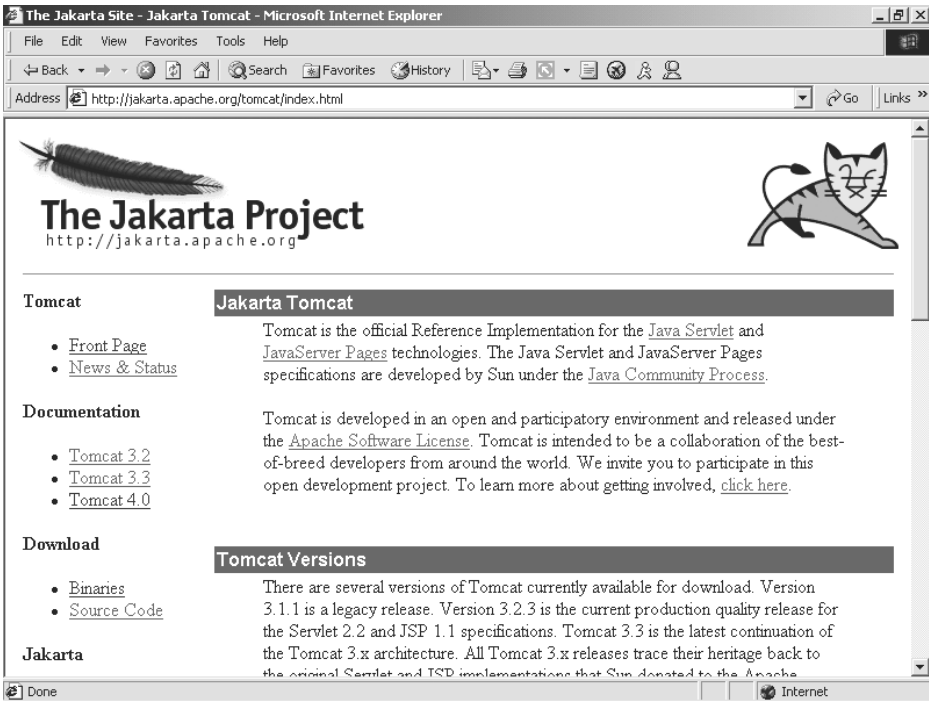

Figure 1-1. The Tomcat homepage

You can also subscribe to the Tomcat mailing lists, which can be found at the following URL:

```
http://jakarta.apache.org/site/mail2.html
```

This page contains all of the mailing lists controlled by the Apache Jakarta project. Once you are on the mailing lists page, scroll down until you find the Tomcat lists and select the list that you would like to subscribe to. Figure 1-2 shows the mailing list options for Tomcat.

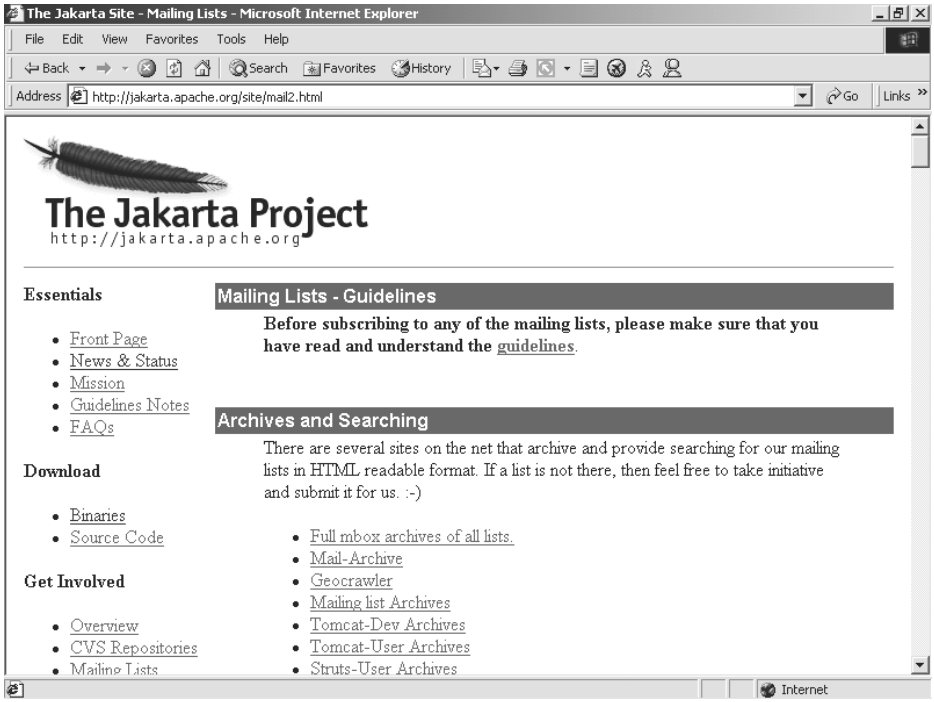

Figure 1-2. The Tomcat mailing lists

The Architecture of Tomcat

Tomcat 4 is a complete rewrite of its ancestors. At the core of this rewrite is the Catalina servlet engine, which acts as the top-level container for all Tomcat instances.

With this rewrite of Tomcat comes an entirely new architecture composed of a grouping of application containers, each with a specific role. The sum of all of these containers makes up an instance of a Catalina engine. The following code snippet provides an XML representation of the relationships between the different Tomcat containers:

```
<Server>

  <Service>

    <Connector />

    <Engine>
```

```
      <Host>

        <Context>
        </Context>

      </Host>

    </Engine>

  </Service>

</Server>
```

This instance can be broken down into a set of containers including a server, a service, a connector, an engine, a host, and a context. By default, each of these containers is configured using the `server.xml` file, which we describe later in more detail.

The Server

The first container element referenced in this snippet is the `<Server>` element. It represents the entire Catalina servlet engine and is used as a top-level element for a single Tomcat instance. The `<Server>` element may contain one or more `<Service>` containers.

The Service

The next container element is the `<Service>` element, which holds a collection of one or more `<Connector>` elements that share a single `<Engine>` element. *N*-number of `<Service>` elements may be nested inside a single `<Server>` element.

The Connector

The next type of element is the `<Connector>` element, which defines the class that does the actual handling requests and responses to and from a calling client application.

The Engine

The third container element is the <Engine> element. Each defined <Service> can have only one <Engine> element, and this single <Engine> component handles all requests received by all of the defined <Connector> components defined by a parent service.

The Host

The <Host> element defines the virtual hosts that are contained in each instance of a Catalina <Engine>. Each <Host> can be a parent to one or more Web applications, with each being represented by a <Context> component.

The Context

The <Context> element is the most commonly used container in a Tomcat instance. Each <Context> element represents an individual Web application that is running within a defined <Host>. There is no limit to the number of contexts that can be defined within a <Host>.

Java Web Applications

The main function of the Tomcat server is to act as a container for Java Web applications. Therefore, before we can begin our Tomcat-specific discussions, a brief introduction as to exactly what Web applications are is in order. The concept of a Web application was introduced with the release of the Java servlet specification 2.2. According to this specification, "a Web Application is a collection of servlets, html pages, classes, and other resources that can be bundled and run on multiple containers from multiple vendors." What this really means is that a Web application is a container that can hold any combination of the following list of objects:

- servlets
- JavaServer pages (JSPs)
- utility classes
- static documents including HTML, images, and so on

- client-side classes
- meta-information describing the Web application

One of the main characteristics of a Web application is its relationship to the `ServletContext`. Each Web application has one and only one `ServletContext`. This relationship is controlled by the servlet container and guarantees that no two Web applications will clash when accessing objects in the `ServletContext`. We discuss this relationship in much more detail in Chapter 3 ("Servlets, JSPs, and the `ServletContext`").

The Directory Structure

The container that holds the components of a Web application is the directory structure in which it exists. The first step in creating a Web application is creating this directory structure. Table 1-1 contains a sample Web application, named `/apress`, and a description of what each of its directories should contain. Each one of these directories should be created from the `<SERVER_ROOT>` of the Web application container. An example of a `<SERVER_ROOT>` using Tomcat would be `/jakarta-tomcat/webapps`.

Table 1-1. The Directories of a Web Application

DIRECTORY	DESCRIPTION
`/apress`	The root directory of the Web application. All JSP and HTML files should be stored here.
`/apress/WEB-INF`	Contains all resources related to the application that are not in the document root of the application. This is where your Web application deployment descriptor is located (defined in the next section). Note that the `WEB-INF` directory is not part of the public document. No files contained in this directory can be requested directly by a client.
`/apress/WEB-INF/classes`	Where servlet and utility classes are located
`/apress/WEB-INF/lib`	Contains Java Archive files that the Web application is dependent upon. For example, this is where you would place a JAR file that contained a JDBC driver or JSP tag library.

As you look over the contents of the Web application's directory structure, notice that Web applications allow for compiled objects to be stored in both the

/WEB-INF/classes and /WEB-INF/lib directories. Of these two, the class loader loads classes from the /classes directory first, followed by the JARs that are stored in the /lib directory. If duplicate objects in both the /classes and /lib directories exist, the objects in the /classes directory take precedence.

The Deployment Descriptor

At the heart of all Web applications is a deployment descriptor that is an XML file named web.xml. The deployment descriptor is located in the /<SERVER_ROOT>/applicationname/WEB-INF/ directory. It describes configuration information for the entire Web application. For our application, the web.xml file is in the /<SERVER_ROOT>/apress /WEB-INF/ directory. The information that is contained in the deployment descriptor includes the following elements:

- servlet definitions
- servlet initialization parameters
- session configuration parameters
- servlet/JSP Mappings
- MIME type mappings
- security configuration parameters
- a welcome file list
- a list of error pages
- resource and environment variable definitions

The following code snippet contains a limited example of a Web application deployment descriptor. As we move through this book, we will be looking at the web.xml file and its elements in much more detail.

```
<web-app>
  <display-name>The APress App</display-name>
  <session-timeout>30</session-timeout>
  <servlet>
    <servlet-name>TestServlet</servlet-name>
    <servlet-class>com.apress.TestServlet</servlet-class>
```

```
    <load-on-startup>1</load-on-startup>
    <init-param>
      <param-name>name</param-name>
      <param-value>value</param-value>
    </init-param>
  </servlet>
</web-app>
```

In this example, we are setting three application-level elements, the first of which is the `<display-name>`. This element simply describes the name of the Web application. It is functionally ineffective.

The second Web application-level element is the `<session-timeout>` element, which controls the lifetime of the application's `HttpSession` object. The `<session-timeout>` value that we have used above tells the JSP/servlet container that the `HttpSession` object will become invalid after 30 minutes of inactivity.

The last application-level element that we have defined is the `<servlet>` element, which defines a servlet and its properties. We will further define the `<servlet>` elements when we discuss deploying servlets and JSPs to Tomcat in Chapter 2 ("Deploying Web Applications to Tomcat").

Packaging

Now that you know what a Web application is, you need to package it for deployment. The standard method for packaging Web applications is to use a Web archive (WAR) file, which you can create by using Java's archiving tool jar. An example of this would be to change to the root directory of your Web application and type the following command:

```
jar cvf apress.war .
```

This command produces an archive file named `apress.war` that contains your entire Web application. Now you can deploy your Web application by simply distributing this file, which we will cover in Chapter 2.

Requirements for Installing and Configuring Tomcat

Before we get started performing the tasks outlined by this chapter, you need to download the items listed in Table 1-2.

Table 1-2. Tomcat Requirements

NAME	LOCATION
Tomcat 4	`http://jakarta.apache.org/site/binindex.html`
JDK 1.3 Standard Edition	`http://java.sun.com/j2se/1.3/`

Installing and Configuring Tomcat

In this section, we install Tomcat as a standalone server, which means that Tomcat will service all requests, including static content, JSPs, and servlets.

To install and configure Tomcat, first download the packages from the previously listed locations. You should choose the appropriate downloads based on your operating system. (We cover the steps involved in installing to both NT/2000 and Linux.)

NOTE *With the release of Tomcat 4, there is a Window installation application. If you choose to install Tomcat from this executable, you can skip the following section and pick up your reading at the section, "Testing Your Tomcat Installation."*

Manually Installing to Windows NT/2000

The first installation we will be performing is for Windows NT/2000. The first thing you need to do is install the JDK. For this example, I am installing the JDK to drive `D:`, so therefore my `JAVA_HOME` directory is `D:\jdk1.3`.

NOTE *Make sure you follow the instructions included with your OS-appropriate JDK.*

Now you need to extract the Tomcat server to the directory where you want it to run. Again, I am installing to drive `D:`, which makes my `TOMCAT_HOME` directory `D:\jakarta-tomcat`.

NOTE *Tomcat does not come packaged with any install scripts. Therefore, extraction equals installation.*

After you have extracted Tomcat, you need to add two environment variables to the NT/2000 system: `JAVA_HOME`, which is the root directory of your JDK installation, and `TOMCAT_HOME`, which is the root directory of your Tomcat installation. To do this under NT/2000, perform the following steps:

1. Open the NT/2000 control panel. You should see an image similar to that shown in Figure 1-3.

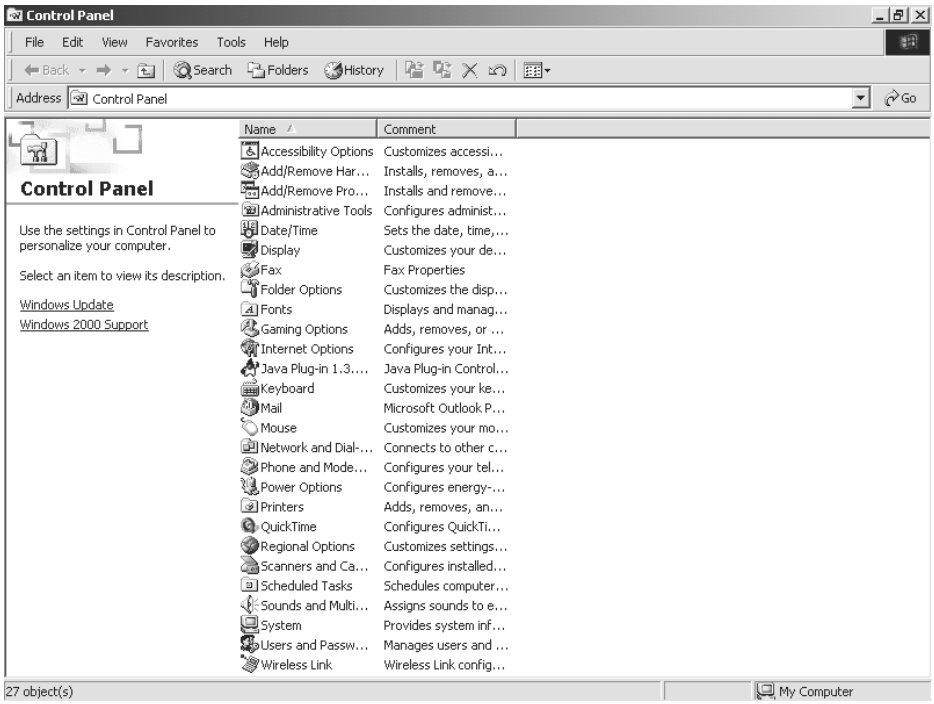

Figure 1-3. NT/2000 control panel

2. Now start the NT/2000 system application and click on the Advanced tab. You should see a screen similar to that shown in Figure 1-4.

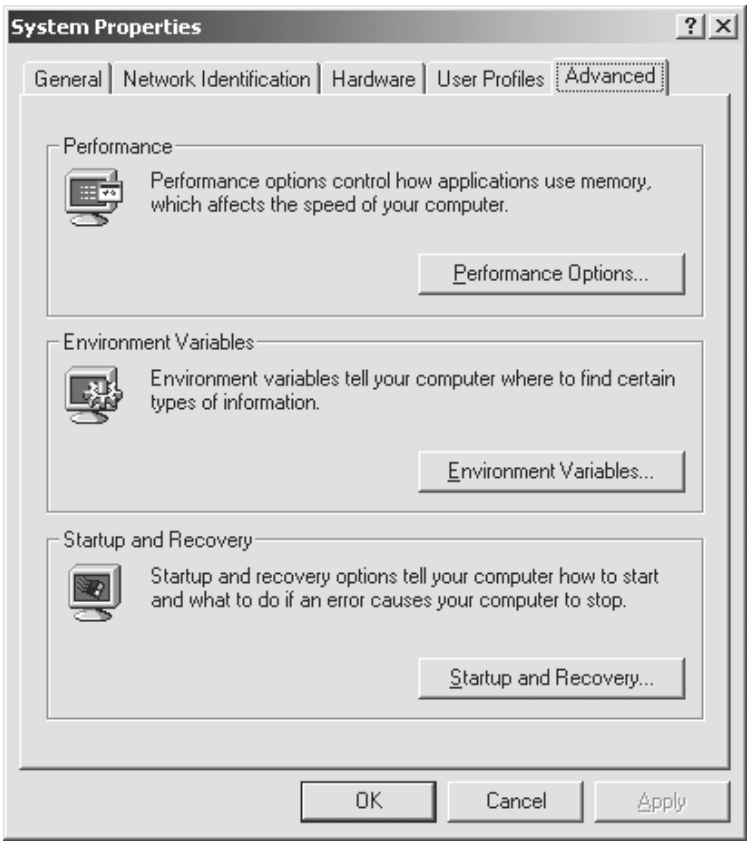

Figure 1-4. NT/2000 system application

3. Next, click on the Environment Variables button. You will see a screen similar to that shown in Figure 1-5.

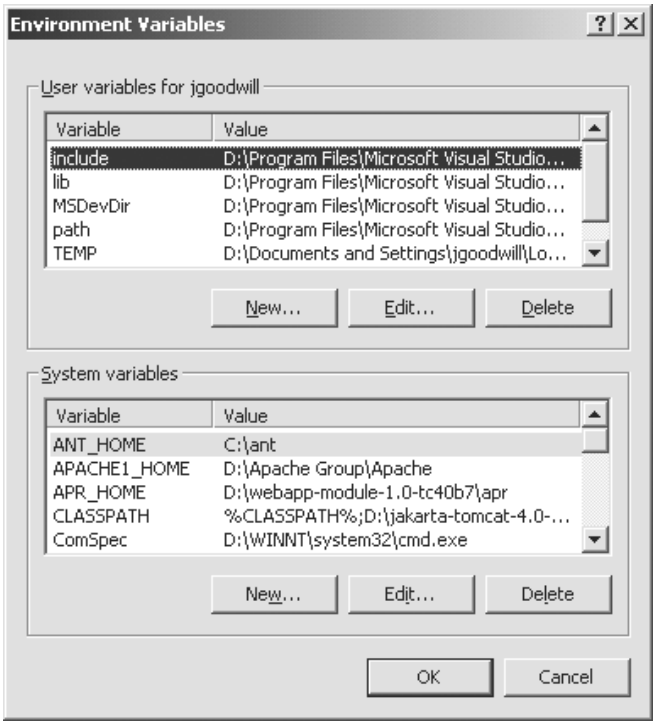

Figure 1-5. Environment variables dialog box

4. Now, click on the New button on the System Variables section of the Environment Variables dialog box. Add a variable named `JAVA_HOME` and set its value to the location of your JDK installation. Figure 1-6 shows the settings associated with my installation.

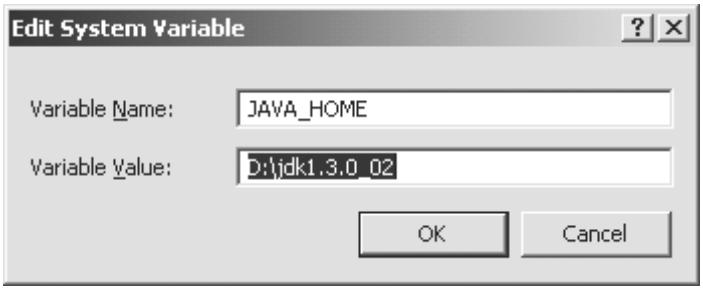

Figure 1-6. `JAVA_HOME` *environment settings*

5. Your final step should be to repeat Step 4, but this time using TOMCAT_HOME for the variable name and the location of your Tomcat installation as the value. For my installation, I am setting the value to D:\jakarta-tomcat.

That is all there is to it. If you are not going to perform a Linux installation, you should skip the following section "Installing to Linux" and move on to the section "Testing Your Tomcat Installation."

Installing to Linux

A Linux installation is a much simpler process compared to a Windows installation. The first thing you need to do is install the downloaded JDK. It is assumed that the JDK is installed to /user/java/jdk1.3.0_02.

After the JDK has been installed, you need to set the JAVA_HOME environment variable. To do this under Linux, find the shell that you are using in Table 1-3 and type the matching command. You need to replace /user/java/jdk1.3.0_02 with the root location of your JDK installation.

Table 1-3. JAVA_HOME *Environment Commands*

SHELL	JAVA_HOME
bash	JAVA_HOME=/user/java/jdk1.3.0_02;export JAVA_HOME
tsh	setenv JAVA_HOME /user/java/jdk1.3.0_02

NOTE *You should also add the location of the Java interpreter to your* PATH *environment variable.*

You now need to extract the Tomcat server to a directory of your choosing. This directory will become the TOMCAT_HOME directory. For this installation, we assume that Tomcat is installed to /var/tomcat.

The last step is to set the TOMCAT_HOME environment variable. Find the shell that you are using in Table 1-4 and type the matching command. You need to replace /var/tomcat with the directory of your Tomcat installation.

Table 1-4. TOMCAT_HOME *Environment Commands*

SHELL	TOMCAT_HOME
bash	TOMCAT_HOME=/var/tomcat;export TOMCAT_HOME
tsh	setenv TOMCAT _HOME /var/tomcat

And that is all there is to the Linux installation. You should now be able to move on to the section, "Testing Your Tomcat Installation."

Testing Your Tomcat Installation

To test the Tomcat installation, you need to first start the Tomcat server. Table 1-5 contains the `startup` and `shutdown` commands for both operating systems.

Table 1-5. Tomcat Startup/Shutdown Commands

OS	STARTUP	SHUTDOWN
Windows NT/2000	TOMCAT_HOME\bin\startup.bat	TOMCAT_HOME\bin\shutdown.bat
Linux	TOMCAT_HOME /bin/startup.sh	TOMCAT_HOME /bin/shutdown.sh

NOTE *If you have installed Tomcat on Windows, a folder was placed in your Windows "Start" menu with shortcuts that allow you to start and stop your Tomcat server from there.*

Once Tomcat has started, open your browser to the following URL:

```
http://localhost:8080/
```

You should see a page similar to that shown in Figure 1-7.

Figure 1-7. The Tomcat default page

If you would like to have all requests serviced on the default HTTP port of 80 instead of port 8080, you need to make the following change to the `TOMCAT_HOME/conf/server.xml` file and restart Tomcat:

From:

```
<!-- Define a non-SSL HTTP/1.1 Connector on port 8080 -->
  <Connector className="org.apache.catalina.connector.http.HttpConnector"
  port="8080" minProcessors="5" maxProcessors="75"
  acceptCount="10" debug="0"/>
```

To:

```
    <!-- Define a non-SSL HTTP/1.1 Connector on port 80 -->
  <Connector className="org.apache.catalina.connector.http.HttpConnector"
  port="80" minProcessors="5" maxProcessors="75"
  acceptCount="10" debug="0"/>
```

Now you should be able to open your browser to the following URL and see results similar to those shown in Figure 1-8:

```
http://localhost
```

The next step is to verify the installation of your JDK. You do this by executing one of the JSP examples provided with the Tomcat server. To execute an example JSP, start from the page shown in Figure 1-7 and choose JSP Examples. You should see a page similar to that shown in Figure 1-8.

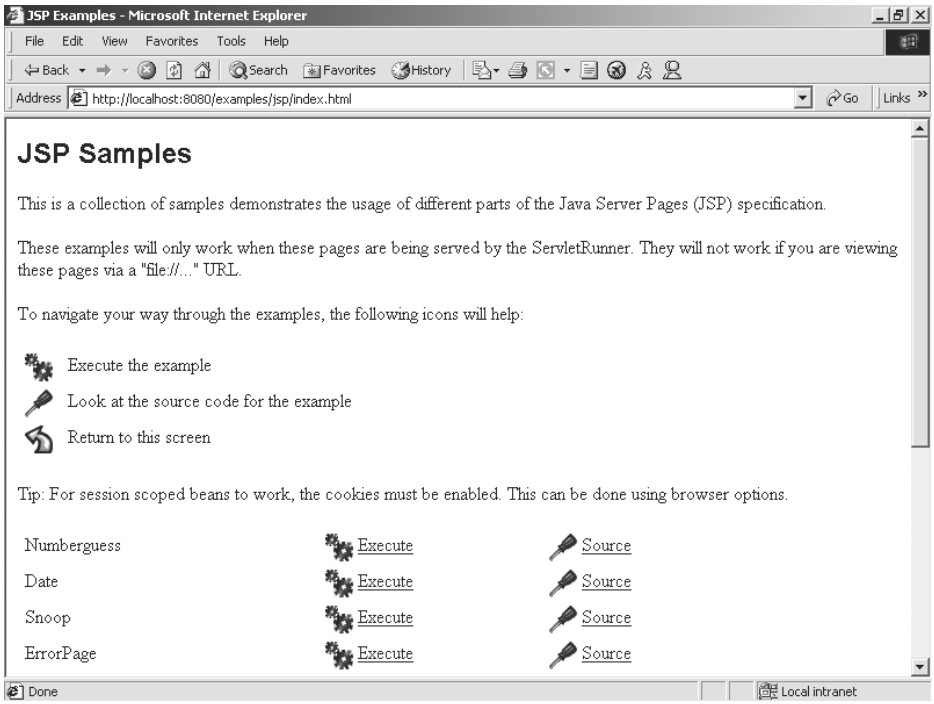

Figure 1-8. The JSP examples page

Now choose the JSP example Date and select the Execute link. If everything was installed properly, you should see a page similar to Figure 1-9 (with a different date, of course).

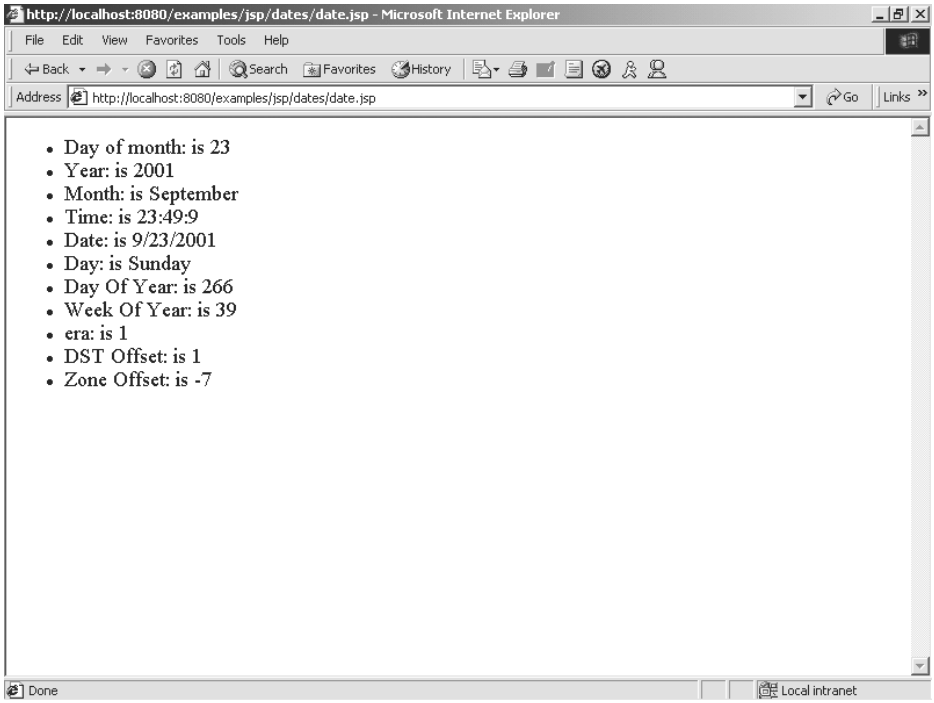

Figure 1-9. The JSP date page

If you do not see the previous page, make sure that the location of your `JAVA_HOME` environment variable matches the location of your JDK installation.

Summary

In this chapter, we introduced the Jakarta Tomcat server and discussed its main uses. We briefly discussed Java Web applications, which are at the core of the Tomcat server. We went on to install and configure Tomcat on both Windows NT/2000 and Linux. We also discussed some simple steps to test your new installation. In the next chapter, "Deploying Web Applications to Tomcat," we begin our discussions on how to create and deploy real Web applications using the Tomcat server.