1

Introducing the Jakarta Struts Project and Its Supporting Components

In this chapter, we lay the foundation for all of our further discussions. We start by providing a high-level description of the Jakarta Struts project. We then describe Java Web applications, which act as the packaging mechanism for all Struts applications. We conclude this chapter with a discussion of the Jakarta Tomcat JSP/servlet container, which we use to host all of our examples throughout the remainder of this text.

At the end of this chapter, you should have an understanding of what the Struts project is, be familiar with its packaging mechanism, and have an installed JSP/servlet container to run your Struts applications.

The Jakarta Struts Project

The Jakarta Struts project, an open-source project sponsored by the Apache Software Foundation, is a server-side Java implementation of the Model-View-Controller (MVC) design pattern. The Struts project was originally created by Craig McClanahan in May 2000, but since that time it has been taken over by the open-source community.

The Struts project was designed with the intention of providing an open-source framework for creating Web applications that easily separate the presentation layer and allow it to be abstracted from the transaction and data layers. Since its inception, Struts has received quite a bit of developer support and is quickly becoming a dominant factor in the open-source community. This book covers version 1.1 of the Jakarta Struts Project.

A small debate is going on in the development community as to the type of design pattern that the Struts project most closely resembles. According the documentation provided by the actual developers of the Struts project, it is patterned after the MVC, but some folks insist that it more closely resembles the Front Controller design pattern described by Sun's J2EE Blueprints Program. The truth is that it does very much resemble the Front Controller pattern, but for the purpose of our discussions, I am sticking with the developers. If you would like to examine the Front Controller yourself, you can find a good article on this topic at the Java Developer Connection site, http://developer.java.sun.com/developer/technicalArticles/J2EE/despat/.

Understanding the MVC Design Pattern

To gain a solid understanding of the Struts Framework, you must have a fundamental understanding of the MVC design pattern, which it is based on. The MVC design pattern, which originated from Smalltalk, consists of three components: a Model, a View, and a Controller. Table 1.1 defines each of these components.

Table 1.1 The Three Components of the MVC

Component	Description
Model	Represents the data objects. The Model is what is being manipulated and presented to the user.
View	Serves as the screen representation of the Model. It is the object that presents the current state of the data objects.
Controller	Defines the way the user interface reacts to the user's input. The Controller component is the object that manipulates the Model, or data object.

We discuss each of these components in more detail throughout this chapter. Some of the major benefits of using the MVC are:

Reliability: The presentation and transaction layers have clear separation, which allows you to change the look and feel of an application without recompiling Model or Controller code.

High reuse and adaptability: The MVC lets you use multiple types of views, all accessing the same server-side code. This includes anything from Web browsers (HTTP) to wireless browsers (WAP).

Very low development and lifecycle costs: The MVC makes it possible to have lower-level programmers develop and maintain the user interfaces.

Rapid deployment: Development time can be significantly reduced, because Controller programmers (Java developers) focus solely on transactions, and View programmers (HTML and JSP developers) focus solely on presentation.

Maintainability: The separation of presentation and business logic also makes it easier to maintain and modify a Struts-based Web application.

The Struts Implementation of the MVC

The Struts Framework models its server-side implementation of the MVC using a combination of JSPs, custom JSP tags, and a Java servlet. In this section, we briefly describe how the Struts Framework maps to each component of the MVC. When we have completed this discussion, we will have drawn a portrait similar to Figure 1.1.

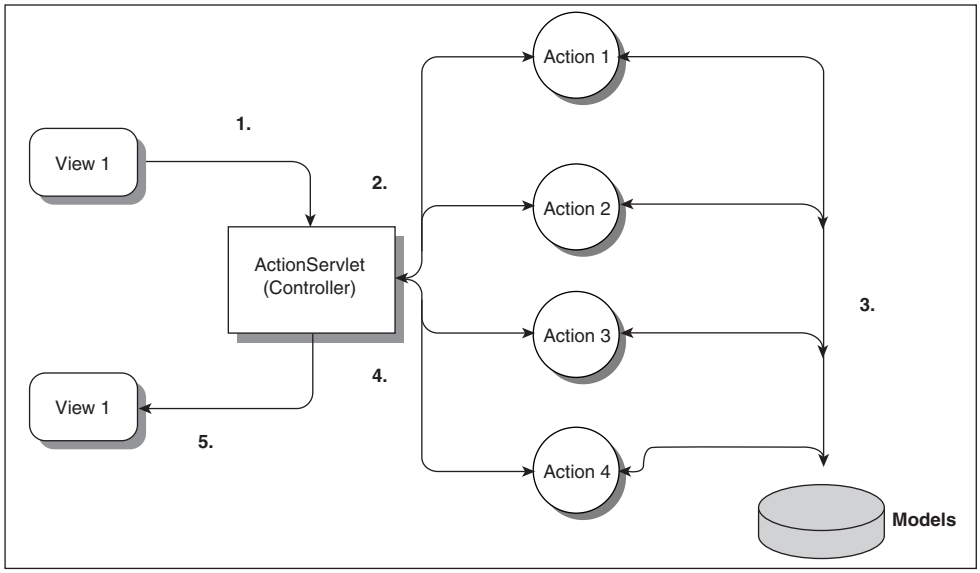

Figure 1.1 The Struts implementation of the MVC.

Figure 1.1 depicts the route that most Struts application requests follow. This process can be broken down into five basic steps. Following these steps is a description of the components involved in this sequence.

1. A request is made from a previously displayed View.
2. The request is received by the ActionServlet, which acts as the Controller, and the ActionServlet looks up the requested URI in an XML file (described in Chapter 3, "Getting Started with Struts"), and determines the name of the Action class that will perform the necessary business logic.
3. The Action class performs its logic on the Model components associated with the application.
4. Once the Action has completed its processing, it returns control to the ActionServlet. As part of the return, the Action class provides a key that indicates the results of its processing. The ActionServlet uses this key to determine where the results should be forwarded for presentation.
5. The request is complete when the ActionServlet responds by forwarding the request to the View that was linked to the returned key, and this View presents the results of the Action.

As you can probably surmise, the previous steps are a simplified representation of a Struts request. We cover these steps in much more detail, including a discussion on several other components, as this text progresses.

The Model

The Model components of the Struts Framework, as we stated earlier, represent the data objects of the Struts application. They often represent business objects or other backend systems and can be implemented as simple JavaBeans, Enterprise JavaBeans, object representations of data stored in a relational database, or just about anything that needs to be manipulated or presented using a Web application. We take a look at the Model component in greater detail in Chapter 11, "Integrating the Jakarta Commons Database Connection Pool (DBCP)."

The View

Each View component in the Struts Framework is mapped to a JSP that can contain any combination of HTML, JSP, and Struts custom tags. JSPs in the Struts Framework serve two main functions. The first is to act as the presentation layer of a previously executed Controller Action. This is most often accomplished using a set of custom tags that are focused around iterating and retrieving data forwarded to the target JSP by the Controller Action. This type of View is not Struts specific and does not warrant special attention.

The second of these functions, which is very much Struts specific, is to gather data that is required to perform a particular Controller Action. This is accomplished most often with a combination of Struts tag libraries and ActionForm objects. This type of View contains several Struts-specific tags and classes. The following code snippet contains a simple example of this type of Struts View:

```
<%@taglib uri="/WEB-INF/struts-html.tld" prefix="html">

<html:form action="loginAction.do"
  name="loginForm"
  type="com.wrox.loginForm" >

  User Id: <html:text property="username"><br/>
  Password: <html:password property="password"><br/>
  <html:submit />
</html:form>
```

As you can see, several JSP custom tags are being leveraged in this JSP. These tags are defined by the Struts Framework and provide a loose coupling to the Controller of a Struts application. We build a working Struts View in Chapter 3, and in Chapter 6, "Building the Presentation Layer," we examine the Struts Views in more detail.

While JSPs are the common presentation tool in a Struts application, they are not the only presentation method available. You can leverage many other tools, including template tools, the Jakarta Velocity Project, and XSLT, among others. For the purpose of this text, we focus on the default Struts presentation layer: JSPs.

The Controller

The Controller component of the Struts Framework is the backbone of all Struts Web applications. It is implemented using a servlet named org.apache.struts.action.ActionServlet. This servlet receives HTTP requests and delegates control of each request, based on the URI of the incoming request, to a user-defined org.apache.struts.action.Action class. The Action class is where the Model of the application is retrieved and/or modified. Once the Action class has completed its processing, it returns a key to the ActionServlet. This key is used to determine the View that will present the results of the Action class's processing. You can think of the ActionServlet as a factory that takes named requests for services and based upon these requests creates Action objects to perform the actual business logic required to complete these services.

The Controller is the most important component of the Struts Framework. We discuss the Controller in Chapter 3, "Getting Started with Struts," and in even greater detail in Chapter 4, "Actions and the ActionServlet."

Web Applications

All Struts applications are packaged using the Java Web application format. Therefore, before we continue, let's take a brief look at Java Web applications.

Java Web applications are best described by the Java Servlet Specification 2.2, which introduced the idea using the following description: "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." In simpler terms, a Java Web application is a collection of one or more Web components that have been packaged together for the purpose of creating a complete application to be executed in the Web layer of an enterprise application. Here is a list of the common components that can be packaged in a Web application:

- Servlets
- JavaServer Pages (JSPs)
- JSP custom tag libraries
- Utility classes and application classes
- Static documents, including HTML, images, and JavaScript
- Metainformation describing the Web application

The Directory Structure

All Web applications are packed into a common directory structure, and this directory structure is the container that holds the components of a Web application. The first step in creating a Web application is to create this structure. Table 1.2 describes a sample Web application, named wroxapp, and lists the contents of each of its directories. Each one of these directories will be created from the <*SERVER_ROOT*> of the Servlet/JSP container.

Table 1.2 The Web Application Directory Structure

Directory	Contains
/wroxapp	This is the root directory of the Web application. All JSP and HTML files are stored here.
/wroxapp/WEB-INF	This directory 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. You should note that the WEB-INF directory is not part of the public document. No files contained in this directory can be served directly to a client.
/ wroxapp/WEB-INF/classes	This directory is where servlet and utility classes are located.
/ wroxapp/WEB-INF/lib	This directory contains Java Archive (JAR) files that the Web application is dependent on.

If you're using Tomcat as your container, the default root directory is <*CATALINA_HOME*>/webapps/. Figure 1.2 shows the wroxapp as it would be hosted by a Tomcat container.

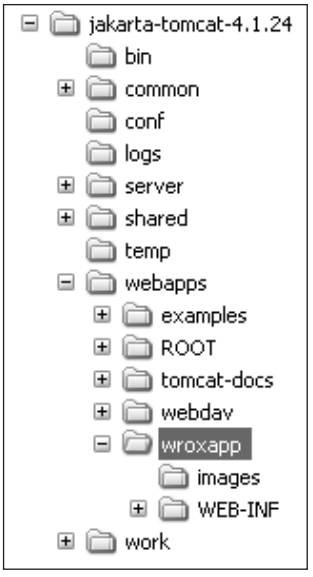

Figure 1.2 The wroxapp Web application hosted by Tomcat.

Web applications allow compiled classes to be stored in both the /WEB-INF/classes and /WEB-INF/lib directories. Of these two directories, the class loader will load classes from the /classes directory first, followed by the JARs in the /lib directory. If you have duplicate classes in both the /classes and /lib directories, the classes in the /classes directory will take precedence.

The Web Application Deployment Descriptor

The backbone of all Web applications is its deployment descriptor. The Web Application deployment descriptor is an XML file named web.xml that is located in the */<SERVER_ROOT>/application-name*/WEB-INF/ directory. The web.xml file describes all of the components in the Web application. If we use the previous Web application name, wroxapp, then the web.xml file would be located in the */<SERVER_ROOT>*/wroxapp /WEB-INF/ directory. The information that can be described in the deployment descriptor includes the following elements:

- ServletContext init parameters
- Localized content
- Session configuration
- Servlet/JSP definitions
- Servlet/JSP mappings
- Tag library references
- MIME type mappings
- Welcome file list
- Error pages
- Security information

This code snippet contains a sample deployment descriptor that defines a single servlet. We examine the web.xml file in much more detail as this text progresses.

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE web-app PUBLIC
  '-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN'
  'http://java.sun.com/dtd/web-app_2_3.dtd'>

<servlet>
  <servlet-name>SimpleServlet</servlet-name>
  <servlet-class>com.wrox.SimpleServlet</servlet-class>
</servlet>

</web-app>
```

Packaging a Web Application

The standard packaging format for a Web application is a Web Archive file (WAR). A WAR file is simply a JAR file with the extension .war as opposed to .jar. You can create a WAR file by using jar, Java's archiving tool. To create a WAR file, you simply need to change to the root directory of your Web application and type the following command:

```
jar cvf wroxapp.war .
```

This command produces an archive file named wroxapp.war that contains the entire wroxapp Web application. You can deploy your Web application by simply distributing this file.

The Tomcat JSP/Servlet Container

The Tomcat JSP/Servlet container is an open-source Java-based Web application container created to run servlet and JavaServer Page Web applications. It has become Sun's reference implementation for both the Servlet and JSP specifications. We use Tomcat for all of our examples in this book.

Before we get started with the installation and configuration of Tomcat, you need to make sure you have acquired the items listed in Table 1.3.

Table 1.3 Tomcat Installation Requirements

Component	Location
Jakarta-Tomcat 4.1.24 LE	http://jakarta.apache.org/
JDK 1.4.1 Standard Edition	http://java.sun.com/j2se/1.4/

Installing and Configuring Tomcat

Once you have Tomcat and the JDK downloaded, go ahead and install them according to their packaged instructions. For our purposes, we are installing Tomcat as a stand-alone server on a Windows XP operating system (OS). We are installing Tomcat in the directory C:\Tomcat 4.1 and the JDK in the directory C:\j2sdk1.4.1_02.

After we have Tomcat and the JDK installed, the next step is to set the JAVA_HOME environment variable. This variable is used to compile your requested JSPs. To do this under XP, perform these steps:

1. Open the Windows XP Control Panel.
2. Start the System Application, and then select the Advanced tab.
3. Click the Environment Variables button. You will see a screen similar to Figure 1.3.

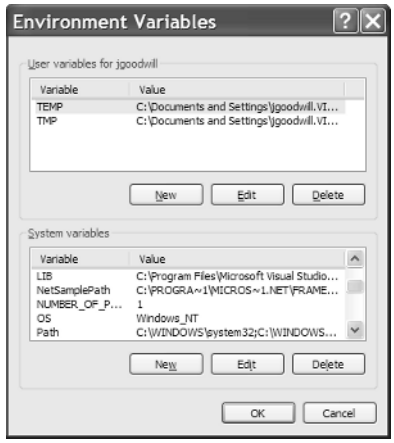

Figure 1.3 The Windows NT/2000 Environment Variables dialog box.

4. Click the New button in 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 1.4 shows the settings associated with our installation.

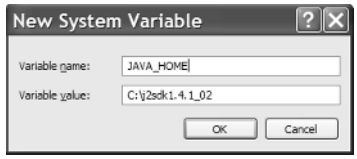

Figure 1.4 The JAVA_HOME environment settings for our installation.

That's all there is to it. You can now move on to the next section, in which we test the Tomcat installation.

Testing Your Tomcat Installation

Before continuing, let's test the steps we have just completed. To begin, first start the Tomcat server by typing the following command (be sure to replace *<CATALINA_HOME>* with the location of your Tomcat installation):

```
<CATALINA_HOME>\bin\startup.bat
```

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

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

You should see the default Tomcat home page, which is displayed in Figure 1.5.

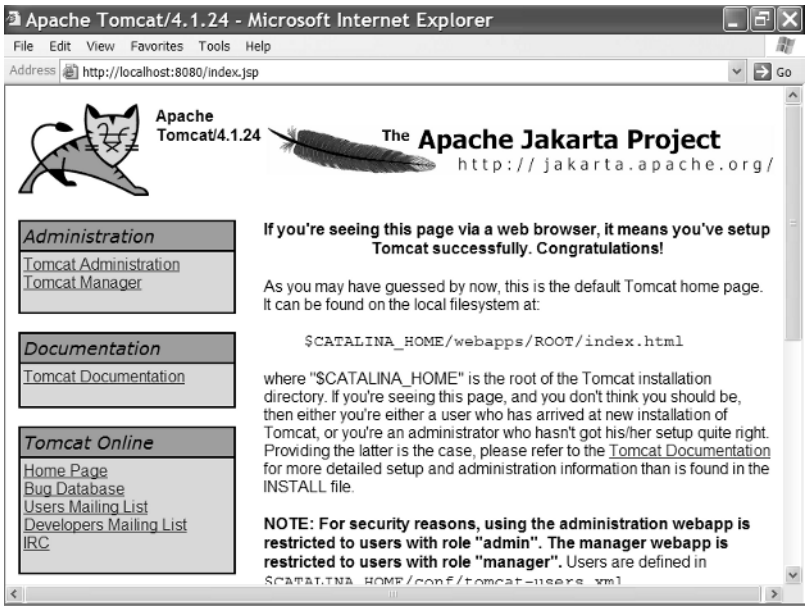

Figure 1.5 The default Tomcat home page.

The next step is to verify the installation of our JDK. The best way to do this is to execute one of the JSP examples provided with the Tomcat server. To execute a sample JSP, start from the default Tomcat home page, shown in Figure 1.5, and choose JSP Examples. You should see a page similar to Figure 1.6.

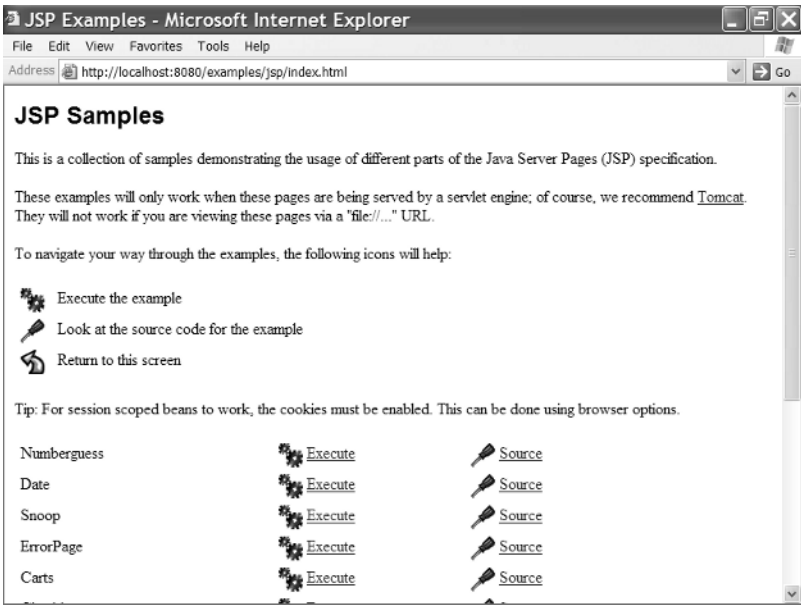

Figure 1.6 The JSP Examples page.

Now choose the JSP example Snoop and click the Execute link. If everything was installed properly, you should see a page similar to the one shown in Figure 1.7.

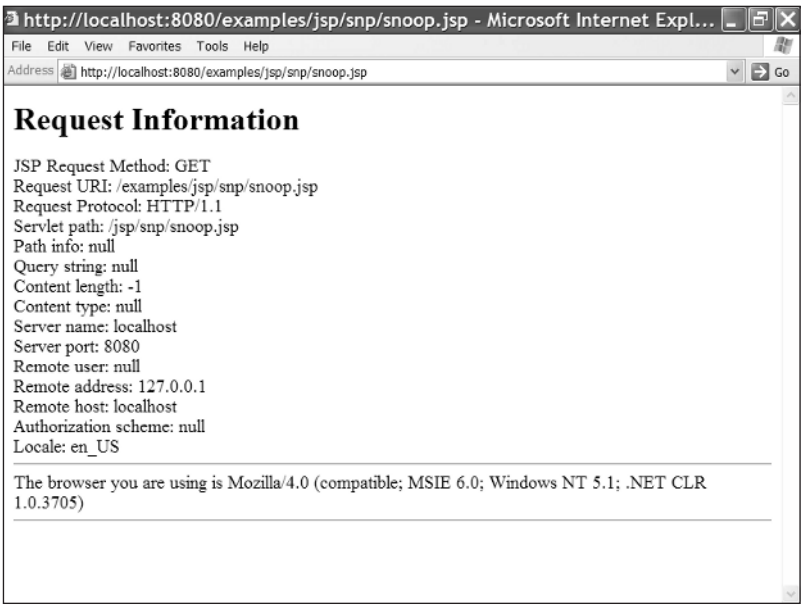

Figure 1.7 The results of executing the Snoop JSP example.

If you do not see the page shown in Figure 1.7, make sure that the location of your JAVA_HOME environment variable matches the location of your JDK installation.

What's Next?

Our next chapter is devoted to a brief tutorial of JSPs and servlets. The goal of this chapter is to provide you with the foundational technologies that you leverage throughout the remainder of this book. If you are already familiar with both of these technologies, you may want to skip to Chapter 3, "Getting Started with Struts."