

Building a Web Application Using Struts 2

A wounded deer leaps the highest.

—Emily Dickinson

The Struts framework is an old living tree whose ring patterns tell the story of the legacy Java web forests. Struts, released in June 2001, pioneered an essential evolution of the Model-2 development model to address the vicissitudes of web development. You can see Struts' DNA assimilated in many other architecturally diverse action-based frameworks that evolved to address Model-2 development. Struts gave birth to the Tiles framework, which can now be used with a myriad of web frameworks.

The growth in Struts' popularity began to lose momentum because of the ever-increasing complexities of web applications and because of the competition from other evolving web frameworks. The WebWork framework that was built on classic Struts later unified with it to create the Struts 2 framework. Struts 2 is a complete rewrite of the architecture of classic Struts, aimed at addressing the aforementioned needs. Struts 2 provides the architectural foundations for web applications, provides architectural mechanisms to automate recurring tasks and to separate cross-cutting concerns, and saves developers from maintaining a plethora of configuration code by means of convention over configuration.

One chapter is not enough to demonstrate the full capabilities of any framework, so my intention here is to demonstrate the fundamentals of Struts 2 web framework. Beginning with this chapter and through the subsequent chapters, you will progressively learn how modern web frameworks provide architectural foundations centered on Java EE web tier patterns.

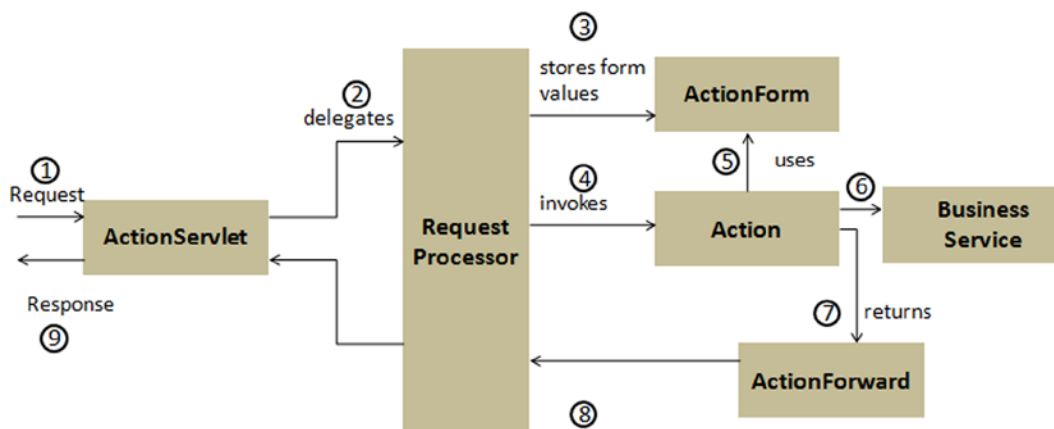
Struts 2 Framework Overview

Table 4-1 describes the key features of the Struts 2 framework.

Table 4-1. Key Features of the Struts 2 Framework

Features	Description
Ajax support	Struts 2 integrates Ajax support. Besides out-of-the-box Ajax, Struts 2 also supports a myriad of Ajax-centric plug-ins.
Convention over configuration	Struts 2 adheres to the principle of convention over configuration and eliminates unnecessary configuration.
Declarative architecture using annotations	Struts 2's declarative architecture using annotations reduces XML configuration and keeps the configuration closer to the action class.
Data conversion	Struts 2 provides automatic type conversion of string-based form field values to objects or primitivetypes, removing the need to provide conversion code in the action class.
Dependency injection	Struts 2 uses dependency injection for the action to collaborate with the components it needs.
Extensibility	Struts 2 is extensible, in that the classes in the framework are interface based.
Plug-in architecture	The core Struts 2 behavior can be enhanced with plug-ins. You can find a number of plug-ins available for Struts 2 here: https://cwiki.apache.org/S2PLUGINS/home.html
POJO forms and actions	Unlike classic Struts that used ActionForms, in Struts 2 you can use any POJO to receive the form input. And any POJO can work as an action.
View technologies	Struts 2 supports multiple views such as JSP, FreeMarker, Velocity, XSLT, and so on.

Before diving into Struts 2, it is worthwhile to understand the architecture of classic Struts (hereafter referred to as Struts). Struts is an MVC-based framework. The central component of a Struts framework is the `ActionServlet`, which implements the Front Controller web-tier Java EE pattern. Figure 4-1 illustrates the architecture of Struts.

**Figure 4-1.** Architecture of the Struts framework

The sequence of events in Struts is as follows:

1. The `ActionServlet` delegates request processing to a `RequestProcessor`.
2. The `RequestProcessor` processes the request and stores the form values in the `ActionForm`.
3. The `RequestProcessor` then invokes the `Action` class.
4. The `Action` class accesses the `ActionForm` to retrieve the form values.
5. The `Action` class invokes the call to the service layer.
6. The `Action` class returns the `ActionForward`, which is used to encapsulate the response view.

Struts 2, however, is different from Struts. Unlike Struts, which follows a push-MVC architecture where data is expected to be present in the page or scope, Struts 2 is a pull-MVC architecture; that is, the data can be pulled from the `Action`. Table 4-2 shows a one-to-one mapping of the Struts and Struts 2 framework elements.

Table 4-2. *One-to-One Mapping of Struts and Struts 2 Framework Elements*

Struts Framework Elements	Struts 2 Framework Elements
<code>ActionServlet</code>	Servlet filter
<code>RequestProcessor</code>	Interceptor
<code>Action</code>	<code>Action</code>
<code>ActionForm</code>	<code>Action</code> or POJOs
<code>ActionForward</code>	<code>Result</code>

Note You can find a comprehensive list of similarities and differences between Struts and Struts 2 at <http://struts.apache.org/release/2.3.x/docs/comparing-struts-1-and-2.html>.

Figure 4-2 illustrates the key elements of Struts 2 that provide a cleaner implementation of MVC.

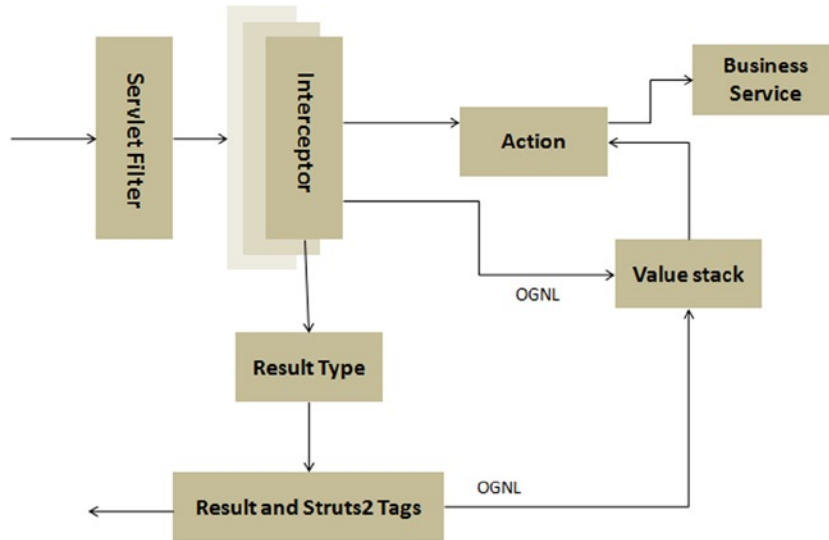


Figure 4-2. Architecture of Struts 2

As you can see from Figure 4-2, Struts 2 is also an MVC-based framework that implements the Front Controller pattern. The sequence of events in Struts 2 framework is as follows:

1. The request is mapped to the configuration metadata.
2. The request passes through a stack of interceptors that provide preprocessing and postprocessing for the request and cross-cutting features.
3. The Action and the method in the Action that provides the logic to process this request is invoked.
4. The Result is invoked to render the response.
5. The response is returned to the user.

The key elements of Struts 2 illustrated in Figure 4-2 are discussed next.

Action

Actions are the core of the action-oriented Struts 2 framework because they provide the necessary logic for request processing. Actions are not required to implement any interface or extend any class, and actions can be POJOs. Listing 4-1 illustrates the action called `HelloWorldAction`.

Listing 4-1. An Action as a POJO

```
public class HelloWorldAction{
//...
public String execute() {
return "success";
}
}
```

This action class is configured in the `struts.xml` file, as illustrated in Listing 4-2.

Listing 4-2. Configuring the Action Class in `struts.xml`

```
<package name="helloWorld" " extends="struts-default" namespace="/" >

<action name="hello" class=" HelloWorldAction">
<result name="success"> /hello.jsp</result>
</action>

<package>
```

The action mapping in Listing 4-2 uses the `name` attribute to define the name of the action you can use to access this action class and uses the `result` tag to define which result page should be returned to the user. Now you can access the action via an `.action` extension.

<http://localhost:8080/helloWorldExample/hello.action>

Even though you can use POJO actions, Struts 2 provides two action helpers that can be used: the `Action` interface and the `ActionSupport` class.

Action Interface

Struts 2 comes with an optional `Action` interface, as illustrated in Listing 4-3.

Listing 4-3. Action Interface

```
package com.opensymphony.xwork2;

public interface Action {
    public static final String ERROR = "error";
    public static final String INPUT = "input";
    public static final String LOGIN = "login";
    public static final String NONE = "none";
    public static final String SUCCESS = "success";

    public String execute();
}
```

This interface provides the common string-based return values as constants and the default `execute()` method that should be implemented by the implementing classes.

The action class that implements this interface can use the constant value directly, as illustrated in Listing 4-4.

Listing 4-4. Using the Action Interface

```
import com.opensymphony.xwork2.Action;
public class HelloWorldAction implements Action{
//..
public String execute() {
return SUCCESS;
}
}
```

ActionSupport Class

The ActionSupport class implements the Action interface and provides an implementation of the execute() method that returns the SUCCESS value. The ActionSupport class also implements some interfaces that provide support for validation, localization, and internationalization, as illustrated in the code fragment of the ActionSupport class in Listing 4-5.

Listing 4-5. ActionSupportClass

```
public class ActionSupport implements Action, Validateable, ValidationAware, TextProvider,
LocaleProvider, Serializable {
...
public String execute(){
return SUCCESS;
}
}
```

Listing 4-6 shows the code fragment through which the ActionSupport class can be used to provide validation.

Listing 4-6. Using ActionSupport

```
import com.opensymphony.xwork2.ActionSupport;

public class LoginAction extends ActionSupport{
private String username;
private String password;
public String getPassword() {
return password;
}
public void setPassword(String password) {
this.password = password;
}

public String getUsername() {
return username;
}

public void setUsername(String username) {
this.username = username;
}
}
```

```
//getters and setters
// ...
public String execute() {
    return "SUCCESS";
}

public void validate(){
    if("").equals(getUsername())){
        addFieldError("username", getText("username.required"));
    }
    if("").equals(getPassword())){
        addFieldError("password", getText("password.required"));
    }
}
}
```

Interceptors

Interceptors promote the separation of concerns by separating the implementation of the cross-cutting concerns from the action. Struts 2 comes with a set of prebuilt interceptors and interceptor stacks that you can use out of the box. Listing 4-7 illustrates the declaration of an action that belongs to a package that extends the `struts-default` package, which contains the default set of interceptors.

Listing 4-7. Declaring an Action

```
<package name="default" namespace="/" extends="struts-default">
<action name="helloAction"
class="HelloWorldAction" >
<result name="success">/hello.jsp</result>
</action>
</package>
```

When you extend your package from the `struts-default` package, by default the `defaultStack` will be used for all the actions in your package. The `defaultStack` is configured in the `struts-default.xml` file, and it provides all the core Struts 2 functionality. The `struts-default.xml` file is located in the `struts-2-core.jar` file. To map other interceptors to an action, you can use the `interceptor-ref` element, as illustrated in Listing 4-8.

Listing 4-8. Mapping Interceptors to the Action

```
<package name="default" namespace="/" extends="struts-default">
<action name="helloAction"
class="HelloWorldAction" >
<interceptor-ref name="logger"/>
<result name="success">/hellot.jsp</result>
</action>
</package>
```

In Listing 4-8, the action mapping maps the logger interceptors to the HelloWorldAction action class via the `interceptor-ref` element. Since the HelloWorldAction is declared its own interceptors, it loses the default set of interceptors, and you have to explicitly declare the defaultStack in order to use it, as illustrated in Listing 4-9.

Listing 4-9. Declaring a Default Stack

```
<package name="default" namespace="/" extends="struts-default">
<action name="helloAction"
class="HelloWorldAction" >
<interceptor-ref name="logger"/>
<interceptor-ref name="defaultStack"/>
<result name="success">/hello.jsp</result>
</action>
</package>
```

ValueStack and OGNL

The Object-Graph Navigation Language (OGNL¹) is a powerful expression language that is used to set and get properties from JavaBeans and to invoke methods from Java classes. It also helps in data transfer and type conversion. OGNL is similar to EL and JSTL, which evaluate expressions and navigate object graphs using dot notation. As you saw in Figure 4-2, OGNL and the ValueStack, though not part of MVC, are at the core of the Struts 2 framework. All the MVC components interact with ValueStack to provide contextual data. These components access ValueStack using OGNL syntax, and OGNL and ValueStack work together in Struts 2 to handle a request. Specifically, when a request is sent to the Struts 2 application, a ValueStack object is created for that request, and the references to all the objects that are created to serve that request as well as scope attributes are maintained in the ValueStack. All these objects are available to the view through OGNL. You will not find OGNL difficult to use because it is similar to EL and JSTL (which are covered in Chapter 3). Listing 4-10 illustrates how OGNL looks. Notice that OGNL uses #, unlike JSP EL, which uses \$.

Listing 4-10. Using OGNL

```
<s:property value="#book.bookTitle" />
```

ResultType and Result

In Struts 2, the rendering of the response consists of the result type and the result. The result type provides the implementation details for the type of view that is returned to the user. Each method on an action returns a result, which includes specifying the result type. If no result type is specified, then the default result type is used, which forwards to a JSP page, as illustrated in Listing 4-11.

Listing 4-11. Default Result Type

```
<result name="success">
  /hello.jsp
</result>
```

¹<http://commons.apache.org/proper/commons-ognl/>

Struts 2 comes with a number of predefined result types. Struts allows you to use other view technologies to present the results including Velocity, FreeMarker, and Tiles, as illustrated in Listing 4-12.

Listing 4-12. Declaring Tiles as the Result Type

```
<action name="login" class="com.apress.bookstore.action.LoginAction">
<result name="success" type="tiles">home</result>
<result name="error" type="tiles">login</result>
</action>
```

Struts 2 Tags

The Struts 2 framework provides a high-level and portable tag API that you can use with JSP. You will learn how the tags work and how to use OGNL to reference values on the ValueStack in the sections that follow. Table 4-3 describes the different categories of Struts 2 tag libraries.

Table 4-3. *Struts 2Tags*

Struts 2 Tags	Description
Ajax tags	Struts 2 provides Ajax support by means of Ajax tags.
Control tags	These are tags that provide ways to manipulate collections of elements.
Data tags	These are tags that render data from the action, internationalized text, and URLs.
Form tags	These are tags that provide wrappers for HTML form tags, as well as additional widgets such as a date picker.
Non form UI tags	The tags in this group are used in forms but are not directly form entry elements. They include error message displays, tabbed panels, and tree views.

Note You can find the full list of Struts 2 tags in the online documentation at <http://struts.apache.org/release/2.3.x/docs/tag-reference.html>.

Getting Started with Struts 2

In this section, you will develop a HelloWorld Struts 2 application. You will use the build and dependency management tool called Maven.² Maven is a command-line tool that is used to build and package projects and to manage dependencies. It makes life easier for developers working across multiple projects by providing the same directory structure across multiple projects.

²<http://maven.apache.org/>

The dependencies, explicitly configured and transitive, described in the Maven configuration file (`pom.xml`) will be accessed from a local repository or downloaded to the local repository during the build process. This feature allows developers to create new projects without creating the common directory structure, creating the configuration files, and coding default classes and tests from scratch. The benefit of using Maven for runtime dependencies is that you don't need to remember and search for required dependencies manually. You will use Maven 4 to import Struts 2 runtime dependencies. Maven 4 is integrated with Eclipse-kepler, as mentioned in Chapter 1. So, you do not have to install the Maven plug-in for Eclipse. To create the Maven project, click New ► Other, as illustrated in Figure 4-3.

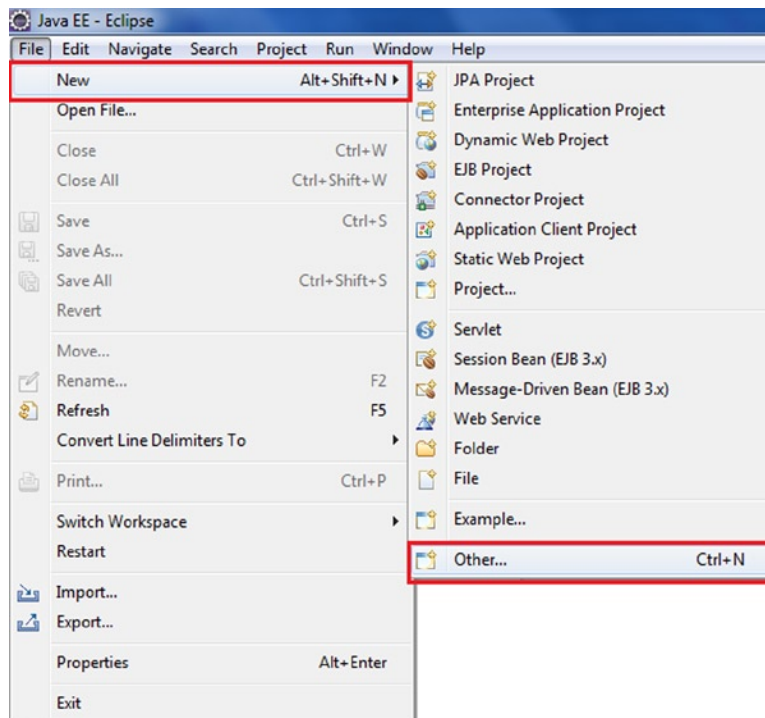


Figure 4-3. Selecting a Maven project

Select Maven Project in the wizard, as illustrated in Figure 4-4.

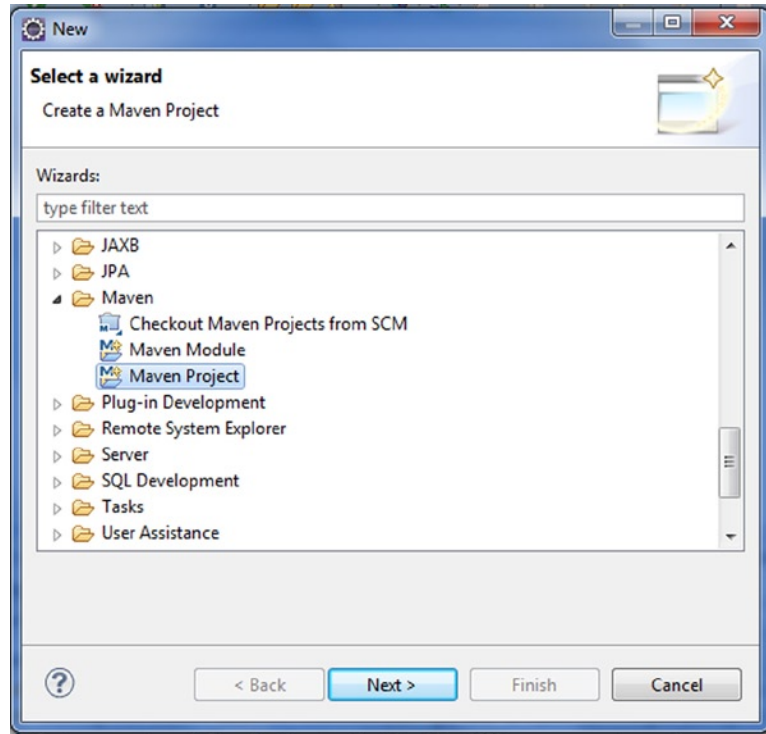


Figure 4-4. Selecting the option Maven Project

Click Next and configure the options as illustrated in Figure 4-5.

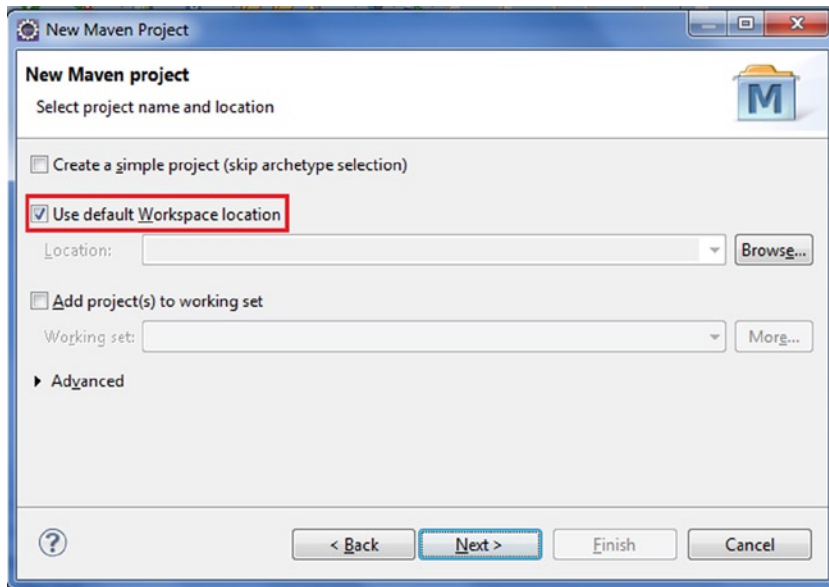


Figure 4-5. Configuring a Maven project

Click Next and as illustrated in Figure 4-6 select Catalog, GroupId, and ArtifactId.

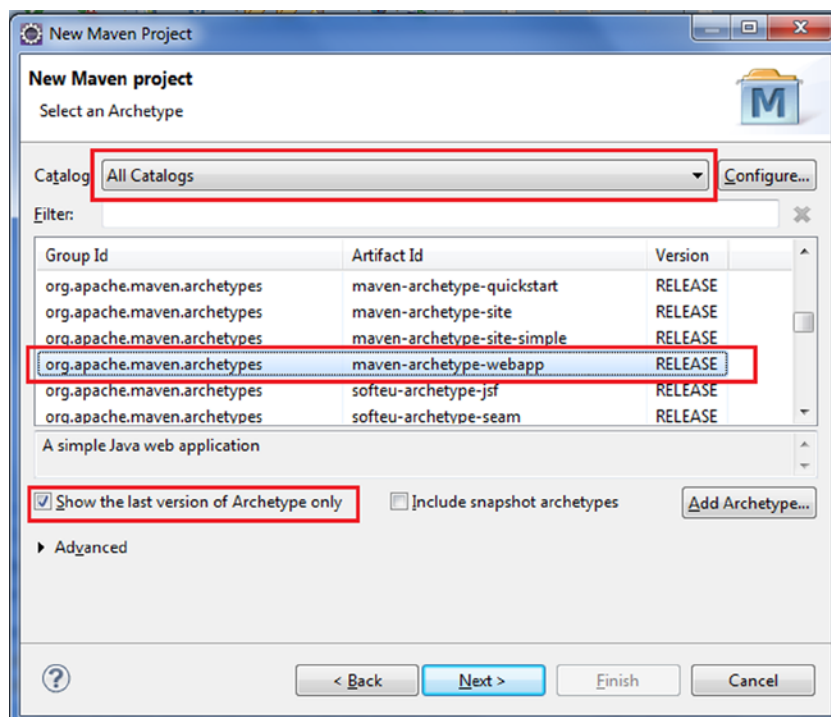


Figure 4-6. Selecting archetypes

Click Next and enter the group ID, artifact ID, and package details, as illustrated in Figure 4-7.

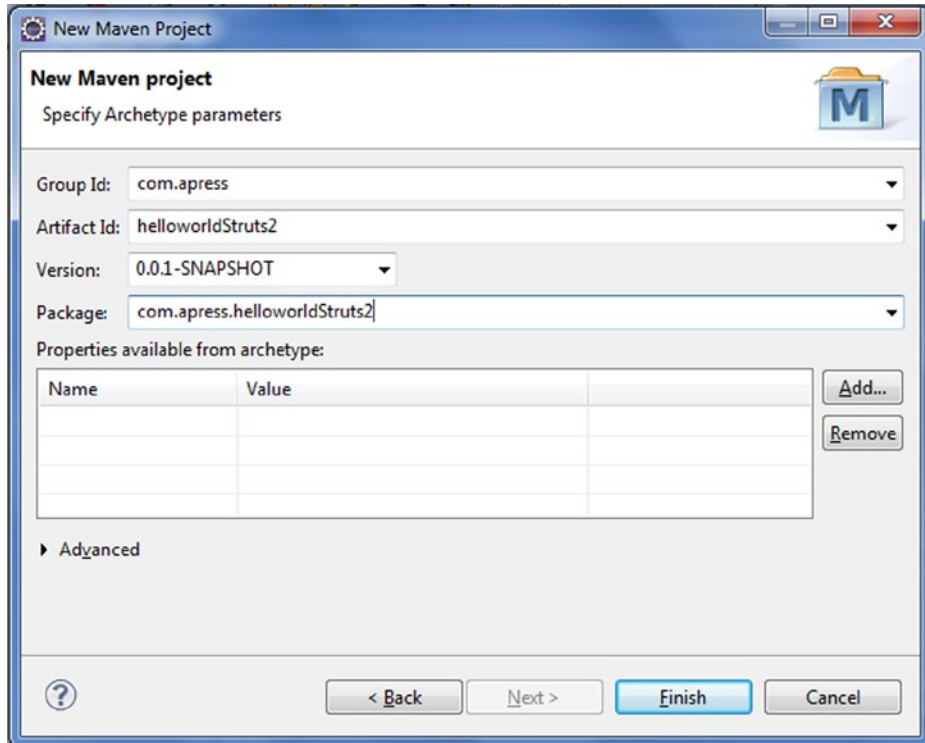


Figure 4-7. Specifying archetype parameters

Click Finish. The project with the name described in the Artifact Id field in Figure 4-7 is created. Figure 4-8 shows the directory structure of the project created.

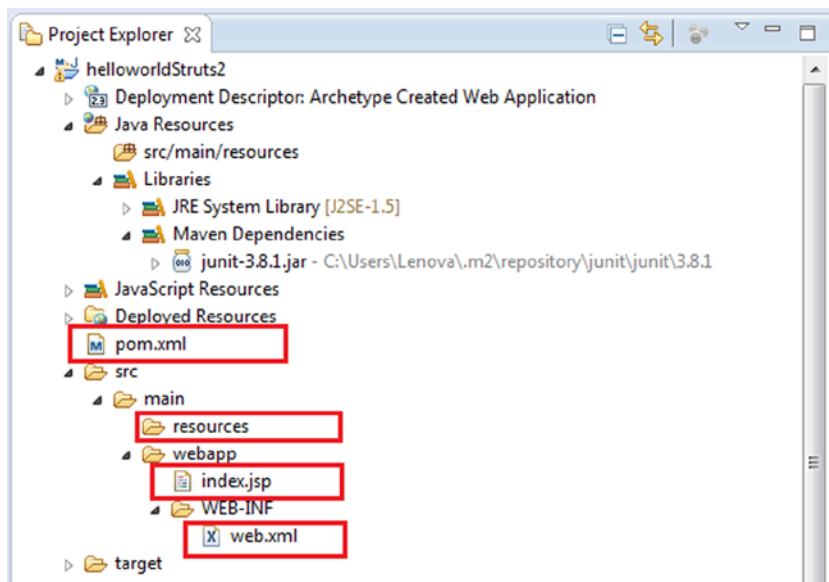


Figure 4-8. Directory structure of the created project

The directory structure illustrated in Figure 4-8 follows the normal Maven directory structure. Table 4-4 describes the directory structure of the HelloWorld application.

Table 4-4. Directory Structure of a Maven-Based Application

Directory	Description
Src	All source
: - main	Main source directory
: : - java	Java source
: : - helloworld	Package defined by the groupId parameter
: : - action	The package from the archetype
: :- resources	Resources(configuration, properties, and so on)
: - webapp	Web application files
: : - WEB-INF	WEB-INF folder

The directory structure of a non-Maven HelloWorld application might look like Table 4-5.

Table 4-5. Directory Structure of a Non-Maven-Based Application

Directory	Description
Src	All source
: - helloworld	Helloworld package
: - action	Action package
- struts.xml	Resources(configuration, properties, and so on)
Web	Web application files
: - WEB-INF	WEB-INF folder

You need to add the struts 2-core dependency to the generated `pom.xml` file of the HelloWorld application. Listing 4-13 shows the code fragment to add to `pom.xml`.

Listing 4-13. struts 2-core Dependency

```
<dependency>
<groupId>org.apache.struts</groupId>
<artifactId>struts 2-core</artifactId>
<version>2.3.15.1</version>
<type>jar</type>
<scope>compile</scope>
</dependency>
```

Your pom.xml file will look like Listing 4-14.

Listing 4-14. pom.xml

```

1.<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
2.  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
3.<modelVersion>4.0.0</modelVersion>
4.<groupId>com.apress</groupId>
5.<artifactId>helloworldStruts2</artifactId>
6.<packaging>war</packaging>
7.<version>0.0.1-SNAPSHOT</version>
8.<name>helloworldStruts2Maven Webapp</name>
9.<url>http://maven.apache.org</url>
10.<dependencies>
11.<dependency>
12.<groupId>junit</groupId>
13.<artifactId>junit</artifactId>
14.<version>3.8.1</version>
15.<scope>test</scope>
16.</dependency>
17.<dependency>
18.<groupId>org.apache.struts</groupId>
19.<artifactId>struts2-core</artifactId>
20.<version>2.3.15.1</version>
21.<type>jar</type>
22.<scope>compile</scope>
23.</dependency>
24.</dependencies>
25.<build>
26.<finalName>helloworldStruts2</finalName>
27.</build>
28.</project>

```

Note You can find the list of dependencies of Struts 2 at <http://struts.apache.org/development/2.x/struts2-core/dependencies.html>.

Listing 4-15 shows the empty deployment descriptor created in the project.

Listing 4-15. web.xml

```

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

```

You need to configure the servlet filter in the deployment descriptor, as illustrated in Listing 4-16.

Listing 4-16. web.xml

```

1.<!DOCTYPE web-app PUBLIC
2."-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
3."http://java.sun.com/dtd/web-app\_2\_3.dtd" >
4.
5.<web-app>
6.<display-name>Hello World Struts2 Web App</display-name>
7.<filter>
8.<filter-name>struts2</filter-name>
9.<filter-class>org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter</filter-class>
10.</filter>
11.
12.<filter-mapping>
13.<filter-name>struts2</filter-name>
14.<url-pattern>/*</url-pattern>
15.</filter-mapping>
16.</web-app>

```

- *Lines 7 to 10:* These lines define the `StrutsPrepareAndExecuteFilter` that is used as the servlet filter in Struts 2 to which all URLs are mapped.

Note The `FilterDispatcher` (`org.apache.struts2.dispatcher.FilterDispatcher`) was used in early Struts 2 development, and it has been deprecated since Struts 2.1.3.

Listing 4-17 illustrates the `struts.xml` file.

Listing 4-17. struts.xml

```

1.<?xml version="1.0" encoding="UTF-8"?>
2.<!DOCTYPE struts PUBLIC
3."-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
4."http://struts.apache.org/dtds/struts-2.0.dtd">
5.
6.<struts>
7.<constant name="struts.devMode" value="true" />
8.<package name="basicstruts2" extends="struts-default">
9.<action name="index">
10.<result>/index.jsp</result>
11.</action>
12.</package>
13.</struts>

```

- *Lines 9 to 11:* These lines define the `index` action, which renders the result `index.jsp`.

Listing 4-18 illustrates index.jsp.

Listing 4-18. index.jsp

```
<html>
<body>
<h2>Hello World!</h2>
</body>
</html>
```

Deploy the web application to the servlet container Tomcat 7, open a web browser, and access <http://localhost:8080/helloworldStruts2/>, as illustrated in Figure 4-9.



Figure 4-9. *Running the HelloWorld application*

The HelloWorld application you developed didn't comprise any actions but a single JSP file that was generated when you created the project. The purpose of the HelloWorld application you created was to test the configuration of Struts 2. You will create actions in the following section.

You will create a HelloWorld project that displays a welcome message to the user, as illustrated in Figure 4-10.

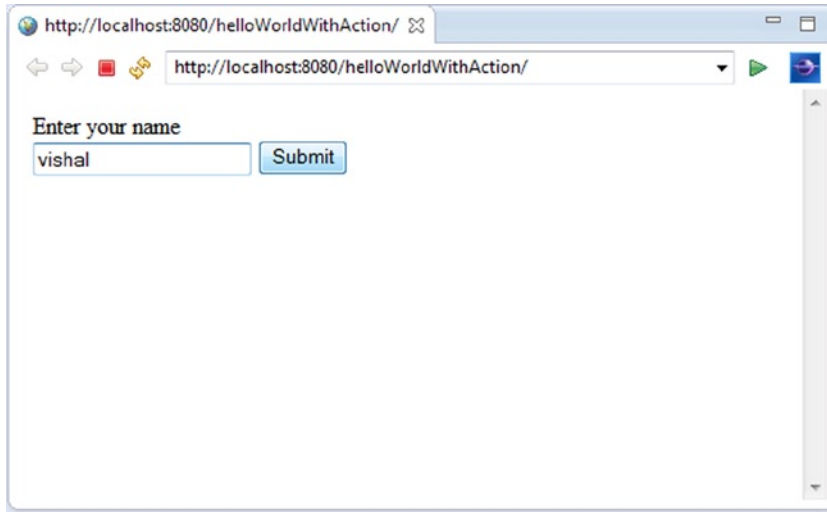


Figure 4-10. Form of the HelloWorld project

When you submit an HTML form in a Struts 2 web application, the input is sent to the Java class called Action. After the action executes, a result selects a resource to render the response, as illustrated in Figure 4-11.

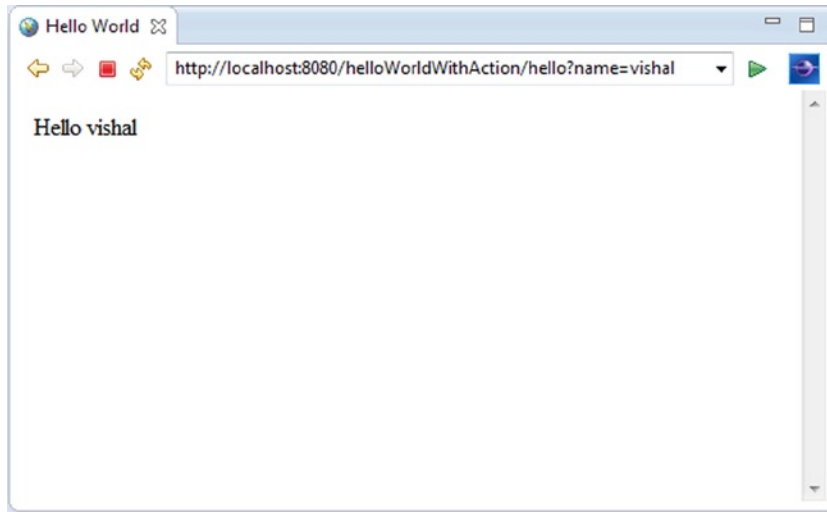


Figure 4-11. Greeting the user

Let's modify the HelloWorld project you created earlier to add an action, the form that accepts a user's input, and the view that greets the user, as illustrated in the directory structure shown in Figure 4-12.

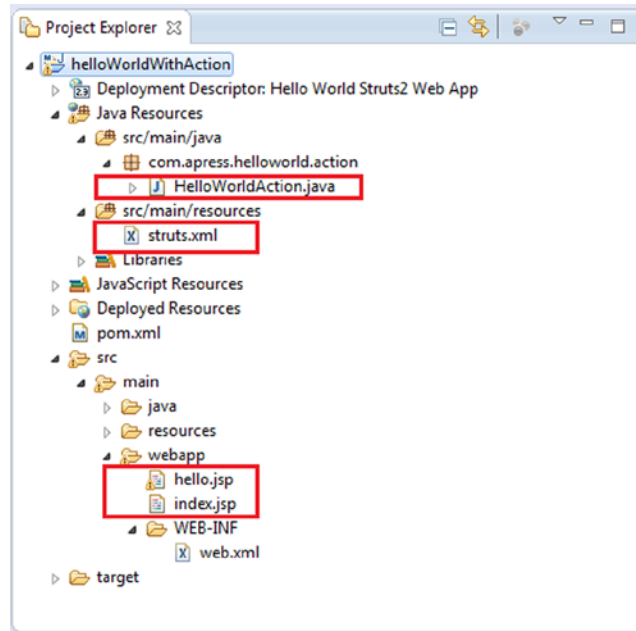


Figure 4-12. The directory structure of the project

Listing 4-19 illustrates the form that allows the user to enter a name and submit it.

Listing 4-19. *index.jsp*

```

1.<html>
2.<body>
3.
4.<form action="hello">
5.<label for="name">Enter your name</label><br /><input type="text"
6.name="name" /><input type="submit" value="Submit" />
7.</form>
8.</body>
9.</html>

```

- **Line 4:** When the user submits the form, the action name hello is sent to the container.

You need a mapping to map the URL to the HelloWorldAction controller. The mapping tells the Struts 2 framework which class will respond to the user's action, which method of that class will be executed, and which view will be rendered as the response. Listing 4-20 illustrates this mapping file.

Listing 4-20. *struts.xml*

```

1.<?xml version="1.0" encoding="UTF-8"?>
2.<!DOCTYPE struts PUBLIC
3.  "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
4.  "http://struts.apache.org/dtds/struts-2.0.dtd">
5.

```

```
6.<struts>
7.<constant name="struts.devMode" value="true" />
8.<package name="basicstruts2" extends="struts-default"
9.namespace="/">
10.
11.<action name="index">
12.<result>/index.jsp</result>
13.</action>
14.
15.<action name="hello" class="com.apress.helloworld.action.HelloWorldAction"
16.method="execute">
17.<result name="success">/hello.jsp</result>
18.</action>
19.</package>
20.</struts>
```

- *Line 15:* This line declares the action mapping for HelloWorldAction. HelloWorldAction is mapped to the action name hello.
- *Line 16:* This line declares that the execute() method of the action is to be executed.
- *Line 17:* This line declares that hello.jsp is designated as a success page and will be rendered as the response.

We need an Action class to act as the controller. The Action class responds to a user action of submitting the form and sending the hello action to the container. Listing 4-21 illustrates HelloWorldAction.

Listing 4-21. HelloWorldAction.java

```
1.package com.apress.helloworld.action;
2.
3.public class HelloWorldAction {
4.private String name;
5.
6.public String execute() throws Exception {
7.return "success";
8.}
9.
10.public String getName() {
11.return name;
12.}
13.
14.public void setName(String name) {
15.this.name = name;
16.}
17.}
18.
```

- *Lines 6 to 7:* The Struts 2 framework will create an object of the HelloWorldAction class and call the execute method in response to a user's action. And the execute method returns the success string, which is mapped to hello.jsp in struts.xml.

Listing 4-22 illustrates `hello.jsp`.

Listing 4-22. `hello.jsp`

```

1.<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
2.pageEncoding="ISO-8859-1"%>
3.<%@ taglib prefix="s" uri="/struts-tags"%>
4.<html>
5.<head>
6.<title>Hello World</title>
7.</head>
8.<body>
9.Hello
10.<s:property value="name" />
11.</body>
12.</html>

```

- **Line 3:** The taglib directive tells the servlet container that this page will be using the Struts 2 tags.
- **Line 10:** The `s:property` tag displays the value returned by calling the method `getName` of the `HelloWorldAction` class. The `getName` method returns a `String`. It is this `String` returned by `getName` that will be displayed by the `s:property` tag.

Now you will learn a different technique of declarative configuration provided by Struts 2: annotations. You can create the new project or modify the project created earlier. Figure 4-13 illustrates the directory structure.

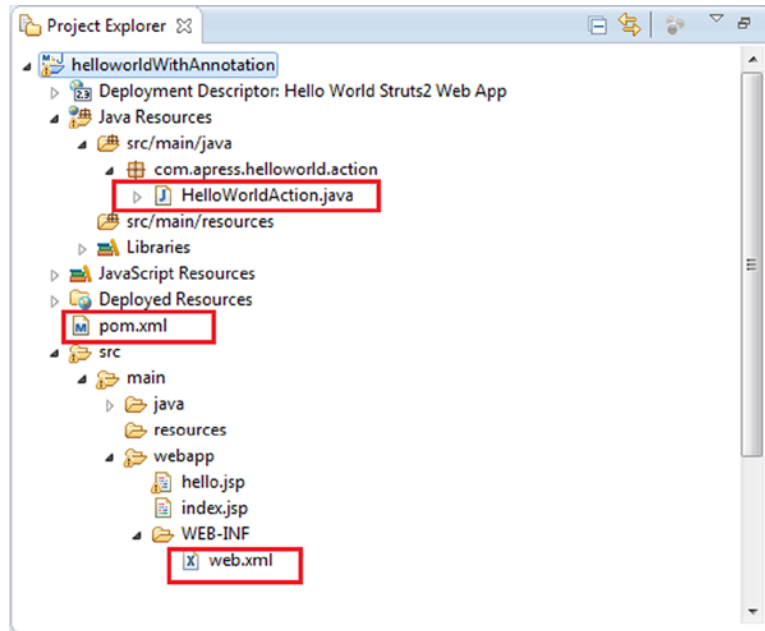


Figure 4-13. *HelloWorld project using Struts 2 annotations*

To use Struts 2 annotations, you need the plug-in called `struts 2-convention-plugin`. Add the dependency to the `pom.xml` file using the fragment illustrated in Listing 4-23.

Listing 4-23. struts 2-convention-plugin

```
<dependency>
<groupId>org.apache.struts</groupId>
<artifactId>struts2-convention-plugin</artifactId>
<version>2.3.15.1</version>
</dependency>
```

Listing 4-24 illustrates `HelloWorldAction` configured with Struts 2 annotations.

Listing 4-24. HelloWorldAction

```
1.package com.apress.helloworld.action;
2.
3.import org.apache.struts2.convention.annotation.Action;
4.import org.apache.struts2.convention.annotation.Result;
5.
6.@Action(value = "/hello", results = { @Result(name = "success", location = "/hello.jsp") })
7.public class HelloWorldAction {
8.private String name;
9.
10.public String execute() throws Exception {
11.return "success";
12.}
13.
14.public String getName() {
15.return name;
16.}
17.
18.public void setName(String name) {
19.this.name = name;
20.}
21.}
```

- *Line 6:* `@Action` defines the URL of an action. Since the value of the action annotation is `"/hello"`, the action will be invoked for the request URL `"/hello"`.
- *Line 6:* `@Result` defines a result for an action. The result annotation maps the result code to the result page. Here the result code `"success"` is mapped to the result `"/hello.jsp"`.

Listing 4-24 uses action and result annotations just to show you how to use them. You can also use the intelligent defaults provided by the Convention plug-in. If you set the `actionPackages` filter init parameter to a comma-separated list of packages containing action classes in `web.xml`, as illustrated in Listing 4-25, the packages and their subpackages will be scanned. All classes in the designated packages that implement `Action` or the POJO actions that don't implement the `Action` interface and end with `Action` are examined.

Listing 4-25. actionPackages Init Parameter

```
<init-param>
<param-name>actionPackages</param-name>
<param-value>com.apress.helloworld.action</param-value>
</init-param>
```

The Convention plug-in uses the Action class name to map the action URL. The Convention plug-in first removes the word Action at the end of the class name and then converts the camel-case name to dashes. So, by default HelloWorldAction will be invoked for the request URL hello-world. But if you want the action to be invoked for a different URL, then you can do this by using the action annotation.

Tip You can find the list of all Struts 2 annotations at <http://struts.apache.org/release/2.3.x/docs/annotations.html>.

In the section that follows, you will develop the bookstore web application using Struts 2.

Bookstore Web Application

In this section, you will progressively develop the bookstore application including the following functionality:

- The login functionality
- Templates
- Integration with the data access layer
- Login via a database
- Selection of categories from a database
- Listing books by category

The complete code for the application is available in the downloadable archive on the Apress website.

The Login Functionality

Figure 4-14 shows the initial login screen.

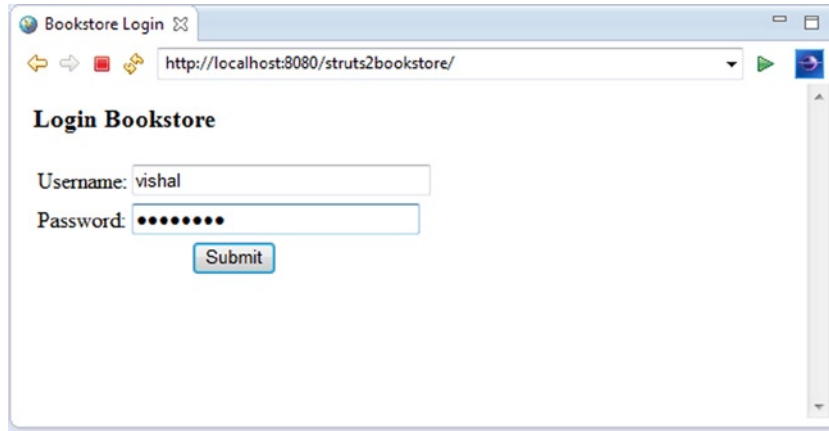


Figure 4-14. Login page of the bookstore application

When the user submits a valid name and password combination, the user will be logged in, and the user's name is displayed as illustrated in Figure 4-15.

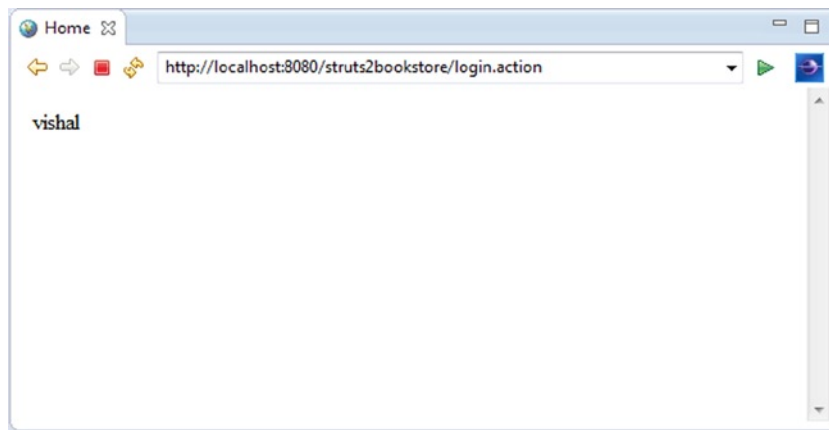


Figure 4-15. Login successful

If the user enters an incorrect username or password, an error message will be displayed to the user, as illustrated in Figure 4-16.

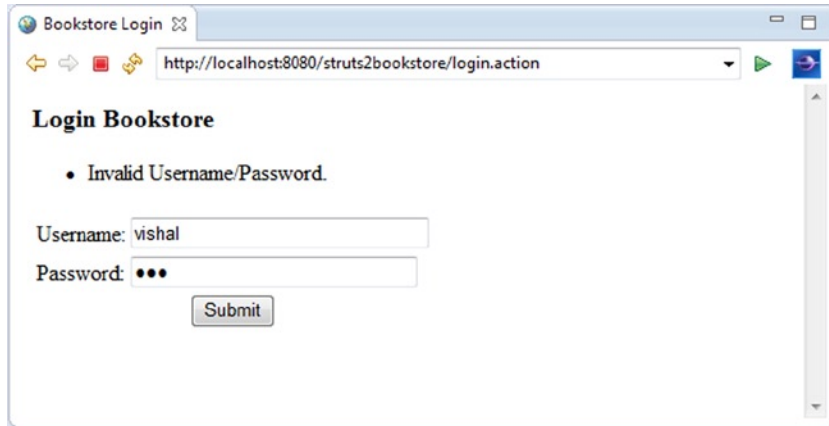


Figure 4-16. Login failed

Figure 4-17 illustrates the directory structure of the application. You can download the source code for the application from the Apress website, and then as you move along with each section, you can refer to the source for imports and other artifacts.

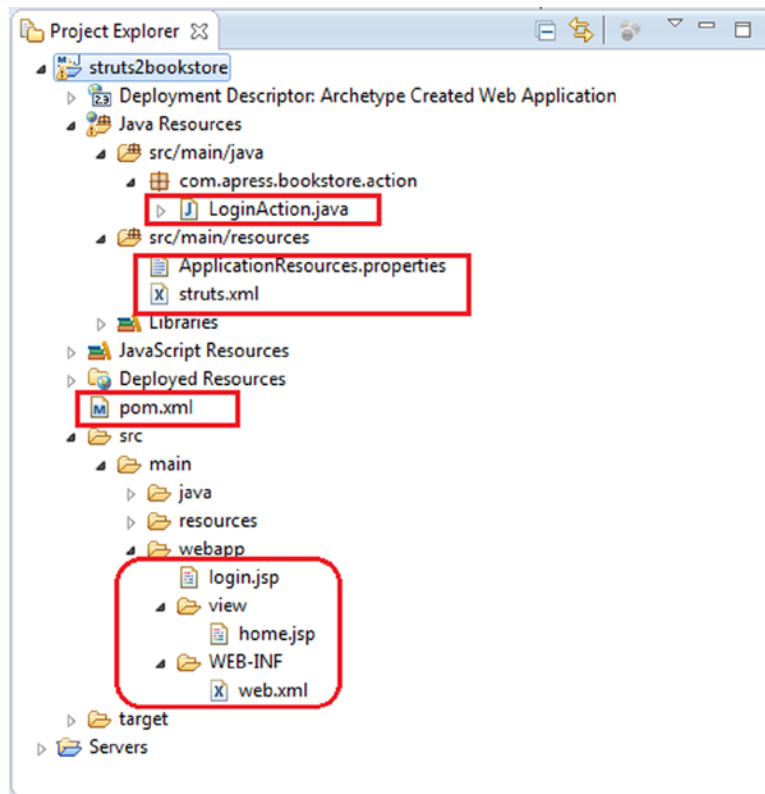


Figure 4-17. Directory structure of the bookstore application

You can add the struts 2-core dependency to your pom.xml file, as illustrated in Listing 4-26.

Listing 4-26. struts 2-core Dependency

```
<dependency>
<groupId>org.apache.struts</groupId>
<artifactId>struts 2-core</artifactId>
<version>2.3.15.1</version>
<type>jar</type>
<scope>compile</scope>
</dependency>
```

You can use the same web.xml file used earlier in the HelloWorld project. Modify the welcome-file-list file, as illustrated in Listing 4-27.

Listing 4-27. Welcome File

```
<welcome-file-list>
<welcome-file>login.jsp</welcome-file>
</welcome-file-list>
```

Listing 4-28 illustrates the login.jsp file.

Listing 4-28. login.jsp

```
1.<%@ page contentType="text/html; charset=UTF-8"%>
2.<%@ taglib prefix="s" uri="/struts-tags"%>
3.<html>
4.<head>
5.<title>Bookstore Login</title>
6.</head>
7.<body>
8.<h3>Login Bookstore</h3>
9.<s:actionerror />
10.<s:form action="login.action" method="post">
11.<s:textfield name="username" key="label.username" size="30" />
12.<s:password name="password" key="label.password" size="30" />
13.<s:submit method="execute" align="center" />
14.</s:form>
15.</body>
16.</html>
```

Listing 4-28 illustrates the usage of several Struts 2 tags. This is a login form that allows the user to input a username and password. In line 10, the name of the action login is sent to the container when the user submits the form. This action name is mapped to LoginAction via struts.xml, as illustrated in Listing 4-29.

Listing 4-29. struts.xml

```

1.<?xml version="1.0" encoding="UTF-8" ?>
2.<!DOCTYPE struts PUBLIC
3."-//Apache Software Foundation//DTD Struts Configuration 2.3//EN"
4."http://struts.apache.org/dtds/struts-2.3.dtd>
5.
6.
7.<struts>
8.<constant name="struts.enable.DynamicMethodInvocation" value="false" />
9.<constant name="struts.devMode" value="true" />
10.<constant name="struts.custom.i18n.resources" value="ApplicationResources" />
11.
12.<package name="default" extends="struts-default" namespace="/">
13.<action name="login" class="com.apress.bookstore.action.LoginAction">
14.<result name="success">view/home.jsp</result>
15.<result name="error">login.jsp</result>
16.</action>
17.</package>
18.</struts>

```

In Listing 4-29, line 13 maps the `LoginAction` class of the login name to the URL, and line 14 renders `home.jsp` when the String success is returned by the `LoginAction` class. If the `LoginAction` class returns the String error, then `login.jsp` is rendered again with the error message.

Listing 4-30 illustrates `LoginAction`.

Listing 4-30. LoginAction.java

```

1.package com.apress.bookstore.action;
2.import com.opensymphony.xwork2.ActionSupport;
3.public class LoginAction extends ActionSupport {
4.private String username;
5.private String password;
6.public String execute() {
7.if (this.username.equals("vishal") && this.password.equals("password")) {
8.return "success";
9.} else {
10.addActionError(getText("error.login"));
11.return "error";
12.}
13.}
14.public String getUsername() {
15.return username;
16.}
17.public void setUsername(String username) {
18.this.username = username;
19.}
20.
21.public String getPassword() {
22.return password;
23.}
24.

```

```
25.public void setPassword(String password) {  
26.this.password = password;  
27.}  
28.}
```

In lines 7 to 9 of Listing 4-30, the username and password are hard-coded. Later you will see how to authenticate against a database. If the username or password is invalid, the `addActionError` method in line 10 gets the `error.login` message mapped to the `ApplicationResources.properties`, illustrated in Listing 4-31, and returns the `String` error.

Listing 4-31. ApplicationResources.properties

```
label.username= Username  
label.password= Password  
error.login= Invalid Username/Password
```

Listing 4-32 illustrates `home.jsp`, which is rendered when the `LoginAction` class returns the `String` success.

Listing 4-32. home.jsp

```
1.<%@ page contentType="text/html; charset=UTF-8"%>  
2.<%@ taglib prefix="s" uri="/struts-tags"%>  
3.<html>  
4.<head>  
5.<title>Home</title>  
6.</head>  
7.<body>  
8.<s:property value="username" />  
9.</body>  
10.</html>
```

Listing 4-32 displays the username using the `s:property` tag when `LoginAction` returns the `String` success.

Developing Templates

In this section, you will see how to integrate the Tiles framework with Struts 2. We will add Tiles support to the HelloWorld Struts application created in the previous section. Tiles is a templating system that reduces code duplication and maintains a consistent look and feel across all the pages of a web application. With Tiles, you define a common layout in a configuration file, and this layout is extended across all the webpages of the web application. This means you can change the look and feel of all the pages of a web application by changing just the template file, instead of changing all the pages. For the bookstore application, you will add a header and menu, as illustrated in Figure 4-18.

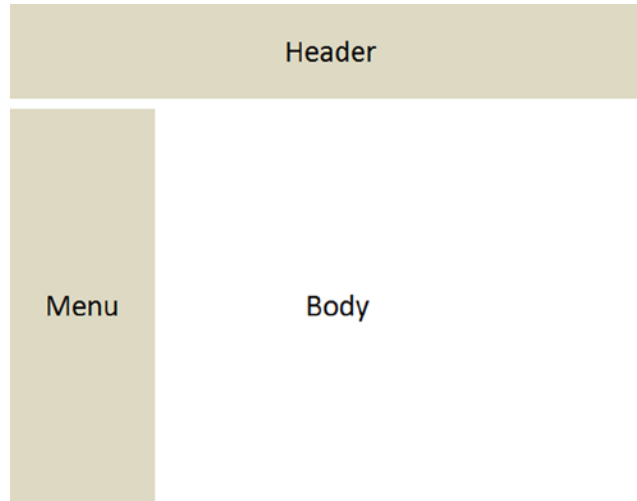


Figure 4-18. Template for the bookstore application

Figure 4-19 illustrates the login screen with the header. When the authentication is successful, the user is logged in, and the home page with the menu bar is rendered, as illustrated in Figure 4-20. If the username or password is invalid, the user remains on the login screen, and an error message is displayed, as illustrated in Figure 4-21.

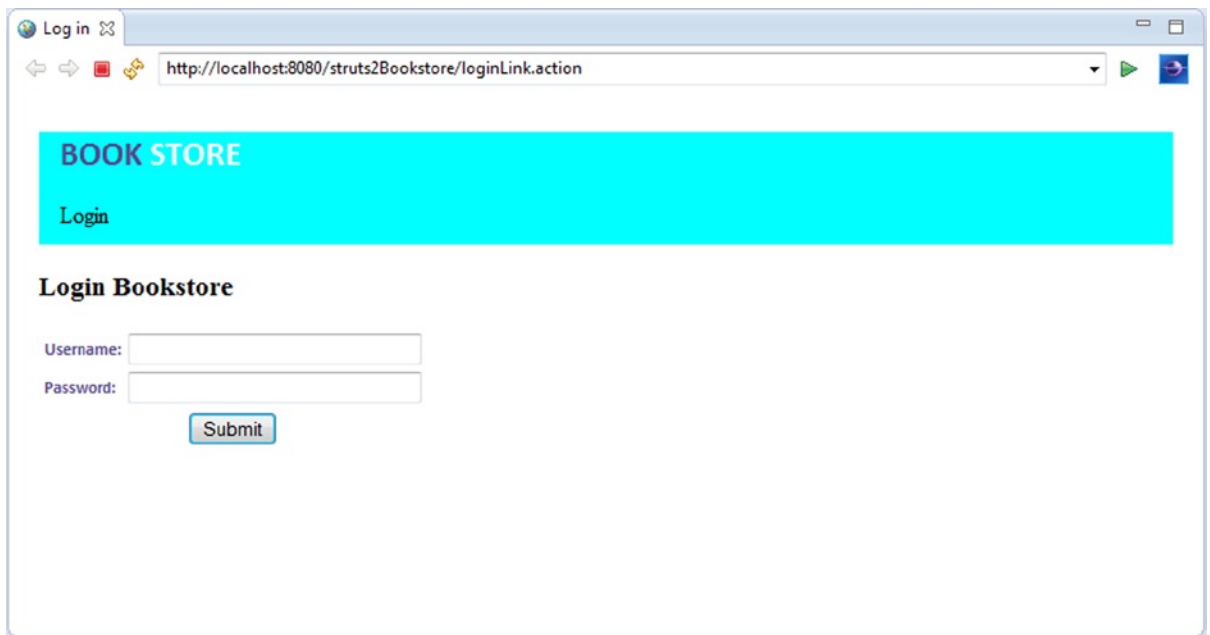


Figure 4-19. Login screen with header

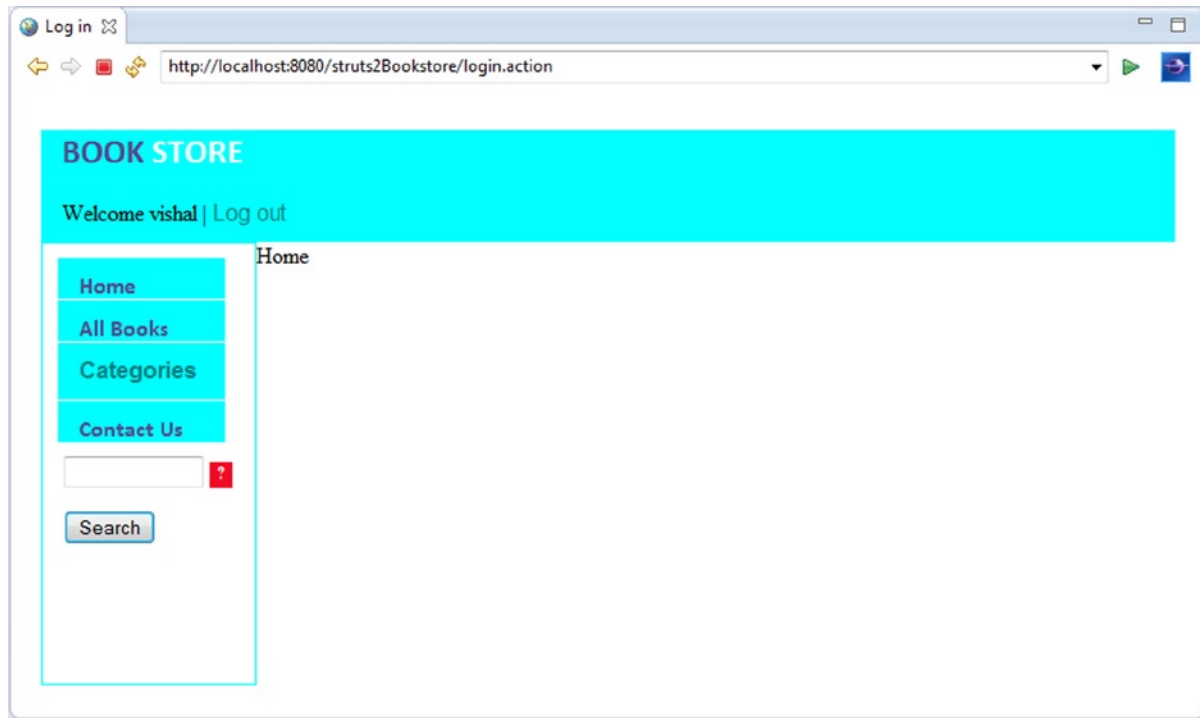


Figure 4-20. Menu displayed on successful login

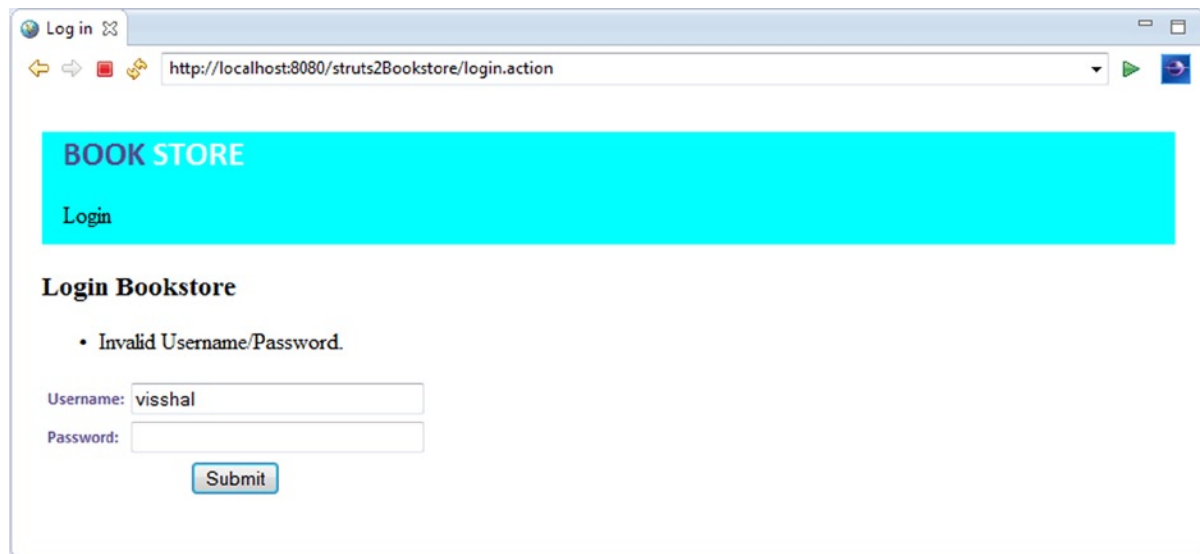


Figure 4-21. Login failed with header

When the user is logged in, the user's name is displayed on the header, and the menu appears, as illustrated in Figure 4-20.

Figure 4-22 illustrates the directory structure with the template files.

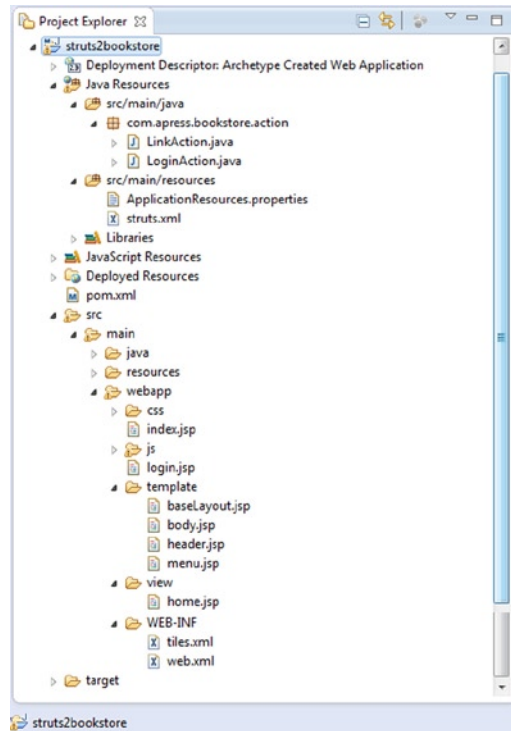


Figure 4-22. The directory structure with the template files

You need to modify the `pom.xml` file with `struts 2-tiles3-plugin` and `slf4j-log4j12`, as illustrated in Listing 4-33. These plug-ins are required to integrate Tiles with Struts 2.

Listing 4-33. *struts-tiles and slf4j-log4j12 Plug-Ins*

```
<dependency>
<groupId>org.apache.struts</groupId>
<artifactId>struts2-tiles3-plugin</artifactId>
<version>2.3.15.1</version>
</dependency>
<dependency>
<groupId>org.slf4j</groupId>
<artifactId>slf4j-log4j12</artifactId>
<version>1.5.6</version>
</dependency>
```

Listing 4-34 illustrates the `struts.xml` file.

Listing 4-34. struts.xml

```
1.<?xml version="1.0" encoding="UTF-8" ?>
2.<!DOCTYPE struts PUBLIC
3."-//Apache Software Foundation//DTD Struts Configuration 2.3//EN"
4."http://struts.apache.org/dtds/struts-2.3.dtd>
5.
6.
7.<struts>
8.<constant name="struts.enable.DynamicMethodInvocation" value="false" />
9.<constant name="struts.devMode" value="true" />
10.<constant name="struts.custom.i18n.resources" value="ApplicationResources" />
11.
12.<package name="default" extends="struts-default" namespace="/">
13.<result-types>
14.<result-type name="tiles"
15.class="org.apache.struts2.views.tiles.TilesResult" />
16.</result-types>
17.<action name="*Link" method="{1}"
18.class="com.apress.bookstore.action.LinkAction">
19.<result name="login" type="tiles">login</result>
20.<result name="allBooks" type="tiles">allBooks</result>
21.</action>
22.<action name="login" class="com.apress.bookstore.action.LoginAction">
23.<result name="success" type="tiles">home</result>
24.<result name="error" type="tiles">login</result>
25.</action>
26.<action name="logout">
27.<result name="success" type="tiles">logout</result>
28.</action>
29.</package>
30.</struts>
```

- *Lines 19, 20, 23, 24, and 27:* These lines define the result type as Tiles and the tile name, such as `home` and `login`, which is defined in `tiles.xml`. Depending on the result string returned by the action, in other words, `success` or `error`, the corresponding JSP file defined in `tiles.xml` is mapped with the definition name in `tiles.xml` to the tile name in `struts.xml`.

Listing 4-35 illustrates the `tiles.xml` file.

Listing 4-35. tiles.xml

```
1.<?xml version="1.0" encoding="UTF-8" ?>
2.
3.<!DOCTYPE tiles-definitions PUBLIC "-//Apache Software Foundation//DTD Tiles Configuration 3.0//EN"
4."http://tiles.apache.org/dtds/tiles-config\_3\_0.dtd>
5.<tiles-definitions>
6.
```

```

7.<definition name="baseLayout" template="/template/baseLayout.jsp">
8.<put-attribute name="title" value="Template" />
9.<put-attribute name="header" value="/template/header.jsp" />
10.<put-attribute name="menu" value="/template/menu.jsp" />
11.<put-attribute name="body" value="/template/body.jsp" />
12.</definition>
13.
14.<definition name="login" extends="baseLayout">
15.<put-attribute name="title" value="Log in" />
16.<put-attribute name="menu" value="" />
17.<put-attribute name="body" value="/login.jsp" />
18.</definition>
19.<definition name="error" extends="baseLayout">
20.<put-attribute name="title" value="Log in" />
21.<put-attribute name="menu" value="" />
22.<put-attribute name="body" value="/login.jsp" />
23.</definition>
24.<definition name="home" extends="baseLayout">
25.<put-attribute name="title" value="Log in" />
26.<put-attribute name="menu" value="/template/menu.jsp" />
27.<put-attribute name="body" value="/view/home.jsp" />
28.</definition>
29.<definition name="logout" extends="baseLayout">
30.<put-attribute name="title" value="Log in" />
31.<put-attribute name="menu" value="" />
32.<put-attribute name="body" value="/login.jsp" />
33.</definition>
34.
35.<definition name="allBooks" extends="baseLayout">
36.<put-attribute name="title" value="All Books" />
37.<put-attribute name="body" value="/allBooks.jsp" />
38.</definition>
39.
40.
41.</tiles-definitions>

```

Listing 4-36 illustrates the `baseLayout.jsp` file, which defines where the header, footer, and body content should be inserted.

Listing 4-36. Base layout

```

1.<%@ taglib uri="http://tiles.apache.org/tags-tiles" prefix="tiles"%>
2.<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
3.  "http://www.w3.org/TR/html4/loose.dtd">
4.
5.<html>
6.<head>
7.<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
8.<link rel="stylesheet" href="css/bookstore.css" type="text/css" />
9.<script type="text/javascript" src="js/jquery-1.9.1.js"></script>
10.<script src="js/bookstore.js"></script>
11.<title><tiles:insertAttribute name="title" ignore="true" /></title>
12.</head>

```

```
13.<body>
14.<div id="centered">
15.
16.
17.<tiles:insertAttribute name="header" />
18.
19.<tiles:insertAttribute name="menu" />
20.
21.<tiles:insertAttribute name="body" />
22.
23.
24.</div>
25.</body>
26.</html>
```

Listing 4-37 illustrates the `header.jsp` file.

Listing 4-37. header.jsp

```
1.<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
2.pageEncoding="ISO-8859-1"%>
3.<%@ taglib prefix="s" uri="/struts-tags"%>
4.
5.<div class="header">
6.<h2>
7.<span style="margin-left: 15px; margin-top: 15px;" class="label">BOOK
8.<span style="color: white;">STORE</span>
9.</span>
10.</h2>
11.<span style="color: black; margin-left: 15px;">
12.
13.
14.<s:if test="%{username!=null && !hasActionErrors() }">Welcome <s:property value="username" /> |
<a href='<s:url action="logout.action"/>'>Log out</a></s:if>
15.<s:else>
16.Login
17.</s:else>
18.
19.
20.
21.
22.</span>
23.
24.</div>
25.
```

Lines 14 to 17 use Struts 2 `s:if`, `s:else`, and `s:property` tags to welcome the user when logged in and allow the user to log out. Since this code is in the header, this functionality will be available to all the pages in the application.

Listing 4-38 illustrates the web.xml file.

Listing 4-38. web.xml

```

1.<!DOCTYPE web-app PUBLIC
2."-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
3."http://java.sun.com/dtd/web-app_2_3.dtd" >
4.
5.<web-app>
6.<display-name>Archetype Created Web Application</display-name>
7.<context-param>
8.<param-name>org.apache.tiles.impl.BasicTilesContainer.DEFINITIONS_CONFIG</param-name>
9.<param-value>/WEB-INF/tiles.xml</param-value>
10.</context-param>
11.
12.<filter>
13.<filter-name>struts2</filter-name>
14.<filter-class>org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter</filter-class>
15.</filter>
16.<filter-mapping>
17.<filter-name>struts2</filter-name>
18.<url-pattern>/*</url-pattern>
19.</filter-mapping>
20.<listener>
21.<listener-class>org.apache.tiles.extras.complete.CompleteAutoloadTilesListener</listener-class>
22.</listener>
23.<welcome-file-list>
24.<welcome-file>index.jsp</welcome-file>
25.</welcome-file-list>
26.
27.</web-app>

```

Lines 7 to 10 and lines 21 to 22 are required to configure Tiles with Struts 2.

Integrating the Data Access Layer

In this section, you will integrate the web application you created with the data access layer. You can use the data access layer you created in Chapter 1. You will then authenticate the user against the database. Next, you will show the categories in the bookstore database on the menu bar. Then you will retrieve the list of books from the database by category. Figure 4-23 illustrates the directory structure of the application.

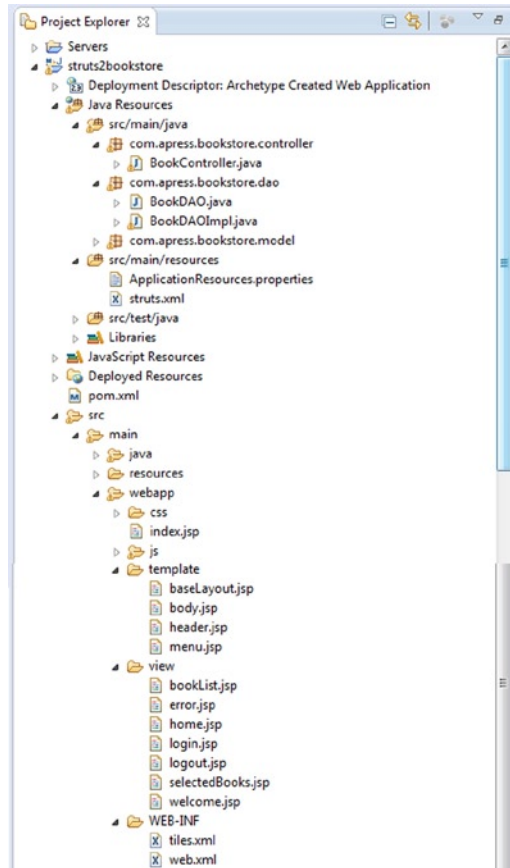


Figure 4-23. Directory structure with the data access layer

As you can see in the directory structure there is a single `BookController` in the application. The methods in this controller are illustrated in the Listing 4-39.

Listing 4-39. `BookController`

```

1.    public class BookController extends ActionSupport {
2.
3.        //properties
4.
5.        public String login() {}
6.
7.        public String executelogin() {}
8.
9.        public String error() {}
10.
11.       public String allBooks() {}
12.
13.       public String booksByCategory() {}
14.

```

```

15.         public String searchByKeyword() {}
16.
17.         public String home() {}
18.
19.         public String selectedBooks(){}
20.
21.         public String logout() {}
22.
23.         // getters and setters
24.
25.     }

```

- *Line 5:* responsible for displaying the login form.
- *Line 7:* authenticates the user against the database.
- *Line 9:* displays the error message, for example if the user is invalid.
- *Line 11:* lists all the books in the bookstore.
- *Line 13:* lists the books by category.
- *Line 15:* allows user to search the books by keyword: book title or author's name.
- *Line 17:* displays the home page when clicked on home link.
- *Line 19:* displays the list of selected books.
- *Line 21:* allows the user to log out.

Login Using Database

Listing 4-40 illustrates the `executelogin()` method responsible for authenticating user against the database. For this you need to add the `USER` table to the data model you developed in chapter 1 using the following DDL:

```

CREATE TABLE USER(
ID INT NOT NULL AUTO_INCREMENT,
FIRST_NAME VARCHAR(60) NOT NULL,
LAST_NAME VARCHAR(60) NOT NULL,
USERNAME VARCHAR(60) NOT NULL,
PASSWORD VARCHAR(60) NOT NULL,
PRIMARY KEY (ID)
);

```

Listing 4-40. *executelogin() method in the BookController*

```

1.     public String executelogin() {
2.         String executelogin = "failed";
3.         session = ActionContext.getContext().getSession();
4.         dao = new BookDAOImpl();
5.         user = new User();
6.         user.setUserName(getUsername());

```

```

7.         user.setPassword(getPassword());
8.         setUser(user);
9.         if (dao.isUserAllowed(user)) {
10.
11.             setCategoryList(dao.findAllCategories());
12.             session.put("username", username);
13.             session.put("categoryList", getCategoryList());
14.             executelogin = "success";
15.         }
16.         else {
17.             addActionError(getText("error.login"));
18.             return "error";
19.         }
20.         // return result;
21.         return "executelogin";
22.     }

```

Figure 4-24 illustrates the USER table.

Field	Type	Null	Key	Default	Extra
ID	int(11)	NO	PRI	NULL	auto_increment
FIRST_NAME	varchar(60)	NO		NULL	
LAST_NAME	varchar(60)	NO		NULL	
USERNAME	varchar(60)	NO		NULL	
PASSWORD	varchar(60)	NO		NULL	

Figure 4-24. USER table

- *Line 9:* The user is authenticated against the database. As you can see in the Listing 4-31 by invoking `isUserAllowed()` on the DAO. The `isUserAllowed()` method selects the username and password from the USER table (as shown in Figure 4-24) in the resultset based on the username and password entered in the login form.
- *Line 11-14:* If the user is valid, categories are retrieved from the database, and the list of categories is stored in the session.
- *Lines 16-19:* Line 16 sets the valid boolean variable to true if the resultset contains the username and password. If the user is not valid, the String error is returned.

Displaying Categories Retrieved from the Database

Listing 4-41 illustrates the menu that shows the categories returned from the database. In Listing 4-40, you retrieved the categories and stored them in the session for a valid user. These categories will be displayed in `menu.jsp`.

Listing 4-41. menu.jsp

```

1.<li><div>
2.
3.<span class="label" style="margin-left: 15px;">
4.<a href="<s:url action="">">Categories</a></span>
5.</div>
6.<ul>
7.<li><s:form action=" booksByCategoryLink">
8.
9.<s:select name="category" list="#session['categoryList']"
10.listValue="categoryDescription" listKey="id" />
11.<s:submit value="Select" />
12.</s:form><a class="label" href=""><span class="label"
13.style="margin-left: 30px;"></span></a></li>
14.
15.</ul></li>

```

- *Lines 9 to 10:* These lines display the categories in the list that was stored in the session in LoginAction using the expression language of Struts 2.
- *Line 11:* When the user clicks the select button, the selected category and the action name in line 7, booksByCategoryLink, are sent to the container, and then mapped via struts.xml to the action, which retrieves the list of books.

Listing Books by Category

In this section, you will retrieve the list of books by category from the database. The selected category and the action name booksByCategoryLink are sent to the container, and then mapped via struts.xml to BookController. Listing 4-42 illustrates the code fragment of struts.xml.

Listing 4-42. Declaring BookController in struts.xml

```

1.      <action name="*Link" method="{1}"
2.                  class="com.apress.bookstore.controller.BookController">
3.          <result name="login" type="tiles">login</result>
4.          <result name="allBooks" type="tiles">booklist</result>
5.          <result name="booksByCategory" type="tiles">booklist</result>
6.          <result name="searchByKeyword" type="tiles">booklist</result>
7.          <result name="home" type="tiles">home</result>
8.          <result name="executellogin" type="tiles">executellogin</result>
9.          <result name="selectedBooks" type="tiles">selectedBooks</result>
10.         <result name="logout" type="tiles">logout</result>
11.         <result name="error" type="tiles">error</result>
12.     </action>

```

The * in the Line 1 of the Listing 4-42 is the wildcard character. Any action name values that end in Link will be handled by this action mapping. Whatever value is before Link will be the value used for the method attribute (the {1} place holder will be replaced with that value). So instead of writing nine separate action mapping nodes in the configuration file for this simple application you can simply use the wildcard character, *, in your name value and an attribute value place holder ({1}) for the

method value. This enables the Struts 2 framework to dynamically select the correct method to call at runtime. Listing 4-43 illustrates `booksByCategory()`.

Listing 4-43. `booksByCategory()`

```
public String booksByCategory() {  
    dao = new BookDAOImpl();  
    setBookList(dao.findBooksByCategory(category));  
    return "booksByCategory";  
}
```

When `booksByCategory()` in the Listing 4-43 returns it is mapped to the tile name `booklist` as it was illustrated in Line 5 of Listing 4-42. This maps to `booklist` defined in the `tiles.xml` as illustrated in Listing 4-44 which renders the `booklist.jsp` file. Listing 4-44 illustrates the code fragment in `tiles.xml`.

Listing 4-44. `tiles.xml`

```
1.<definition name="booklist" extends="baseLayout">  
2.<put-attribute name="title" value="Log in"/>  
3.<put-attribute name="menu" value="/menu.jsp"/>  
4.<put-attribute name="body" value="/view/bookList.jsp"/>  
5.</definition>
```

Listing 4-45 illustrates the `bookList.jsp`.

Listing 4-45. `booklist.jsp`

```
1.<%@ taglib uri="/struts-tags" prefix="s"%>  
3.<body>  
5.<div id="centered">  
8.<s:form action="selectedBooksLink" theme="simple">  
9.<center>  
10.<table id="grid">  
11.<thead>  
12.<tr>  
13.<th id="th-title">Book Title</th>  
14.<th id="th-author">Author</th>  
15.<th id="th-price">Price</th>  
16.</tr>  
17.</thead>  
20.<tbody>  
22.<s:iterator value="bookList" id="book">  
23.<tr>  
25.<td>  
26.<s:checkbox name="selectedBooks" fieldValue="%{bookId}" />  
27.<s:property value="#book.bookTitle" />  
29.</td>
```

```

30.<td>
31.<s:iterator value="#book.authors" id="author">
32.<s:if test="%{#book.id == #author.bookId}">
33.<s:property value="#author.firstName" />
34.<s:property value="#author.lastName" />
35.</s:if>
36.</s:iterator>
37.</td>
39.<td><s:property value="price" /></td>
40.</tr>
42.</s:iterator>
43.</tbody>
45</table>
47.</center><br>
49.<s:submit value="Add to the shopping cart" />
51.</s:form>
52.
53.</div>
54.</body>

```

Listing 4-45 illustrates the usage of Struts 2 tags and how Struts 2 OGNL is used to navigate the property `bookTitle` on line 27 and the properties `firstName` and `lastName` (of the author) on lines 33 and 34. Line 31 illustrates the usage of a simple and nested `s:iterator`. Line 26 allows the user to select books to add to the shopping cart. The user can check the books to select and submit them by clicking the button on line 49. When the user clicks the Add to Cart button, the action name `selectedbooksLink` on line 8 and the `bookId` on line 26 are sent to the container, which is then mapped to the `AddToCart` action via `struts.xml`. The `AddToCart` action then stores the book to the cart in the database, and the `AddToCart` action returns the `String success`, which is mapped via action mapping in `struts.xml` and the tile in the `tiles.xml` file to the view `selectedBooks.jsp`. The user can click the Purchase button on the `selectedBooks.jsp` page to call `PurchaseAction` in the same manner. This chapter sketched a brief overview of Struts 2. For more detailed coverage of Struts 2, I recommend *Practical Apache Struts 2 Web 2.0 Projects* by Ian Roughley.

Summary

In this chapter, you saw the Struts 2 framework's core components, which follow the MVC design pattern, and you saw how Struts 2 separates the cross-cutting concerns by means of interceptors. You saw how the Struts 2 framework provides a declarative architecture in two forms: XML and annotations. In the next chapter, you will learn another action-oriented framework, called Spring Web MVC.