Chapter 2. Developing and Configuring the Views and the Controller

This is Part 2 of a step-by-step tutorial on how to develop a web application from scratch using the Spring Framework. In Part 1 we configured the environment and set up a basic application that we will now flesh out.

This is what we have implemented so far:

- An introduction page, **'index.jsp'**, the welcome page of the application. It was used to test our setup was correct. We will later change this to actually provide a link into our application.
- A `DispatcherServlet` (front controller) with a corresponding **'springapp-servlet.xml'** configuration file.
- A page controller, `HelloController`, with limited functionality – it just returns a `ModelAndView`. We currently have an empty model and will be providing a full model later on.
- A unit test class for the page controller, `HelloControllerTests`, to verify the name of the view is the one we expect.
- A view, **'hello.jsp'**, that again is extremely basic. The good news is the whole setup works and we are now ready to add more functionality.

2.1. Configure JSTL and add JSP header file

We will be using the JSP Standard Tag Library (JSTL), so let's start by copying the JSTL files we need to our **'WEB-INF/lib'** directory. Copy **jstl.jar** from the **'spring-framework-2.5/lib/j2ee'** directory and **standard.jar** from the **'spring-framework-2.5/lib/jakarta-taglibs'** directory to the **'springapp/war/WEB-INF/lib'** directory.

We will be creating a 'header' file that will be included in every JSP page that we're going to write. We ensure the same definitions are included in all our JSPs simply by including the header file. We're also going to put all JSPs in a directory named **'jsp'** under the **'WEB-INF'** directory. This will ensure that views can only be accessed via the controller since it will not be not possible to access these pages directly via a URL. This strategy might not work in some application servers and if this is the case with the one you are using, move the **'jsp'** directory up a level. You would then use **'springapp/war/jsp'** as the directory instead of **'springapp/war/WEB-INF/jsp'** in all the code examples that will follow.

First we create the header file for inclusion in all the JSPs we create.

'springapp/war/WEB-INF/jsp/include.jsp':

```
<%@ page session="false"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
```

Now we can update **'index.jsp'** to use this include file and since we are using JSTL, we can use the `<c:redirect/>` tag for redirecting to our front `Controller`. This means all requests for **'index.jsp'** will go through our application framework. Just delete the current contents of 'index.jsp' and replace it with the following:

'springapp/war/index.jsp':

```
<%@ include file="/WEB-INF/jsp/include.jsp" %>

<%-- Redirected because we can't set the welcome page to a virtual URL. --%>
<c:redirect url="/hello.htm"/>
```

Move **'hello.jsp'** to the **'WEB-INF/jsp'** directory. Add the same include directive we added to **'index.jsp'** to **'hello.jsp'**. We also add the current date and time as output to be retrieved from the model passed to the view which will be rendered using the JSTL `<c:out/>` tag.

'springapp/war/WEB-INF/jsp/hello.jsp':

```
<%@ include file="/WEB-INF/jsp/include.jsp" %>

<html>
  <head><title>Hello :: Spring Application</title></head>
  <body>
    <h1>Hello - Spring Application</h1>
    <p>Greetings, it is now <c:out value="${now}"/></p>
  </body>
</html>
```

2.2. Improve the controller

Before we update the location of the JSP in our controller, let's update our unit test class first. We know we need to update the view's resource reference with its new location **'WEB-INF/jsp/hello.jsp'**. We also know there should be an object in the model mapped to the key "now".

'springapp/tests/HelloControllerTests.java':

```
package springapp.web;

import org.springframework.web.servlet.ModelAndView;

import springapp.web.HelloController;

import junit.framework.TestCase;

public class HelloControllerTests extends TestCase {

    public void testHandleRequestView() throws Exception{
        HelloController controller = new HelloController();
        ModelAndView modelAndView = controller.handleRequest(null, null);
        assertEquals("WEB-INF/jsp/hello.jsp", modelAndView.getViewName());
        assertNotNull(modelAndView.getModel());
        String nowValue = (String) modelAndView.getModel().get("now");
        assertNotNull(nowValue);
    }
}
```

Next, we run the Ant 'tests' target and our test should fail.

```
$ ant tests
Buildfile: build.xml

build:

buildtests:
    [javac] Compiling 1 source file to /home/trisberg/workspace/springapp/war/WEB-INF/classes

tests:
    [junit] Running springapp.web.HelloControllerTests
    [junit] Testsuite: springapp.web.HelloControllerTests
    [junit] Oct 31, 2007 1:27:10 PM springapp.web.HelloController handleRequest
    [junit] INFO: Returning hello view
    [junit] Tests run: 1, Failures: 1, Errors: 0, Time elapsed: 0.046 sec
    [junit] Tests run: 1, Failures: 1, Errors: 0, Time elapsed: 0.046 sec
    [junit]
    [junit] ------------- Standard Error -----------------
    [junit] Oct 31, 2007 1:27:10 PM springapp.web.HelloController handleRequest
    [junit] INFO: Returning hello view
    [junit] ------------- ---------------- ---------------
    [junit] Testcase: testHandleRequestView(springapp.web.HelloControllerTests):        FAILED
    [junit] expected:<[WEB-INF/jsp/]hello.jsp> but was:<[]hello.jsp>
    [junit] junit.framework.ComparisonFailure: expected:<[WEB-INF/jsp/]hello.jsp> but was:<[]hello.jsp>
    [junit]     at springapp.web.HelloControllerTests.testHandleRequestView(HelloControllerTests.java:14)
    [junit]
    [junit]
    [junit] Test springapp.web.HelloControllerTests FAILED

BUILD FAILED
/home/trisberg/workspace/springapp/build.xml:101: tests.failed=true
            ***********************************************************
            ***********************************************************
```

```
            ****  One or more tests failed!  Check the output ...  ****
            ************************************************************
            ************************************************************

Total time: 2 seconds
```

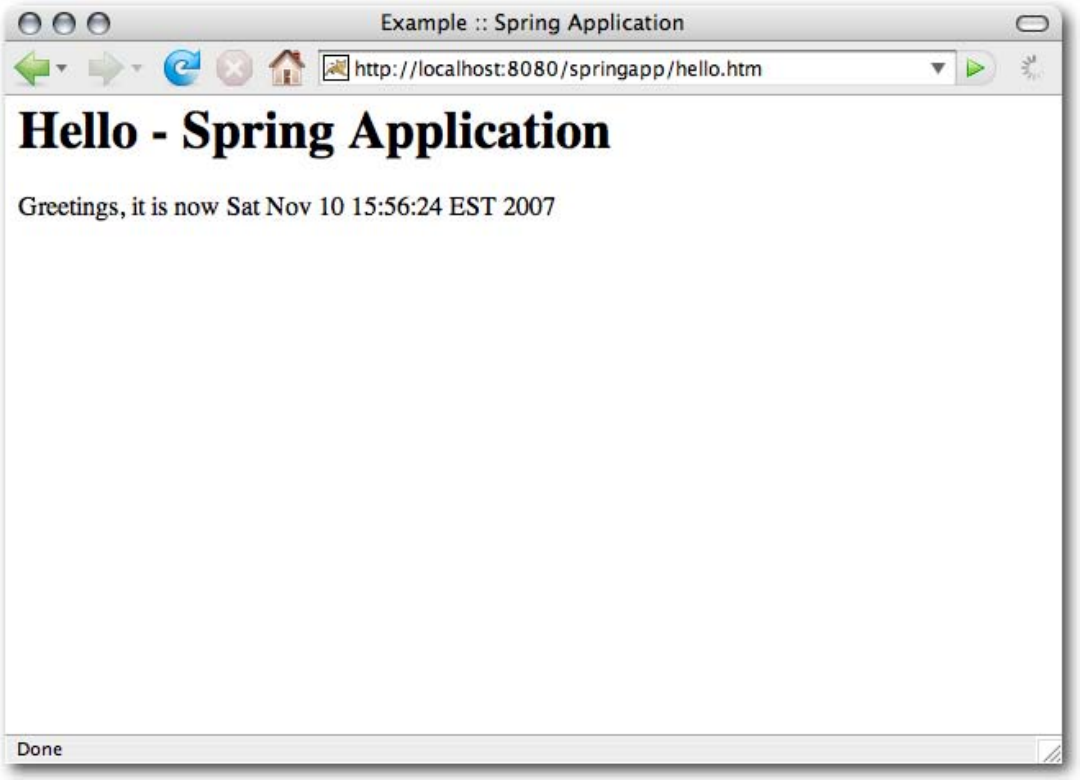
Now we update `HelloController` by setting the view's resource reference to its new location **`'WEB-INF/jsp/hello.jsp'`** as well as set the key/value pair for the current date and time value in the model with the key identifier: "now" and the string value: 'now'.

`'springapp/src/springapp/web/HelloController.java'`:

```
package springapp.web;

import org.springframework.web.servlet.mvc.Controller;
import org.springframework.web.servlet.ModelAndView;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;

import java.io.IOException;
import java.util.Date;

public class HelloController implements Controller {

    protected final Log logger = LogFactory.getLog(getClass());

    public ModelAndView handleRequest(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {

        String now = (new Date()).toString();
        logger.info("Returning hello view with " + now);

        return new ModelAndView("WEB-INF/jsp/hello.jsp", "now", now);
    }

}
```

We rerun our `'tests'` target and the test passes.

Remember that the `Controller` has already been configured in **`'springapp-servlet.xml'`** file, so we are ready to try out our enhancements after we build and deploy this new code. When we enter http://localhost:8080/springapp/ in a browser, it should pull up the welcome file **`'index.jsp'`**, which should redirect to **`'hello.htm'`** and is handled by the `DispatcherServlet`, which in turn delegates our request to the page controller that puts the date and time in the model and then makes the model available to the view **`'hello.jsp'`**.



The updated application

2.3. Decouple the view from the controller

Right now the controller specifies the full path of the view, which creates an unnecessary dependency between the controller and the view. Ideally we would like to map to the view using a logical name, allowing us to switch the view without having to change the controller. You can set this mapping in a properties file if you like using a `ResourceBundleViewResolver` and a `SimpleUrlHandlerMapping` class. For the basic mapping of a view to a location, simply set a prefix and a suffix on the `InternalResourceViewResolver`. This second approach is the one that we will implement now, so we modify the **`'springapp-servlet.xml'`** and declare a `'viewResolver'` entry. By choosing the `JstlView`, it will enable us to use JSTL in combination with message resource bundles as well as provide us with the support for internationalization.

`'springapp/war/WEB-INF/springapp-servlet.xml'`:

```
<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

    <!-- the application context definition for the springapp DispatcherServlet -->

    <bean name="/hello.htm" class="springapp.web.HelloController"/>

    <bean id="viewResolver" class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="viewClass" value="org.springframework.web.servlet.view.JstlView"></property>
        <property name="prefix" value="/WEB-INF/jsp/"></property>
        <property name="suffix" value=".jsp"></property>
    </bean>

</beans>
```

We update the view name in the controller test class `HelloControllerTests` to `'hello'` and rerun the test to check it fails.

`'springapp/test/springapp/web/HelloControllerTests.java'`:

```
package springapp.web;

import org.springframework.web.servlet.ModelAndView;
```

```
import springapp.web.HelloController;

import junit.framework.TestCase;

public class HelloControllerTests extends TestCase {

    public void testHandleRequestView() throws Exception{
        HelloController controller = new HelloController();
        ModelAndView modelAndView = controller.handleRequest(null, null);
        assertEquals("hello", modelAndView.getViewName());
        assertNotNull(modelAndView.getModel());
        String nowValue = (String) modelAndView.getModel().get("now");
        assertNotNull(nowValue);
    }
}
```

We then remove the prefix and suffix from the view name in the controller, leaving the controller to reference the view by its logical name "hello".

'springapp/src/springapp/web/HelloController.java':

```
package springapp.web;

import org.springframework.web.servlet.mvc.Controller;
import org.springframework.web.servlet.ModelAndView;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;

import java.io.IOException;
import java.util.Date;

public class HelloController implements Controller {

    protected final Log logger = LogFactory.getLog(getClass());

    public ModelAndView handleRequest(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {

        String now = (new Date()).toString();
        logger.info("Returning hello view with " + now);

        return new ModelAndView("hello", "now", now);
    }

}
```

Rerun the test and it should now pass.

Let's compile and deploy the application and verify the application still works.

2.4. Summary

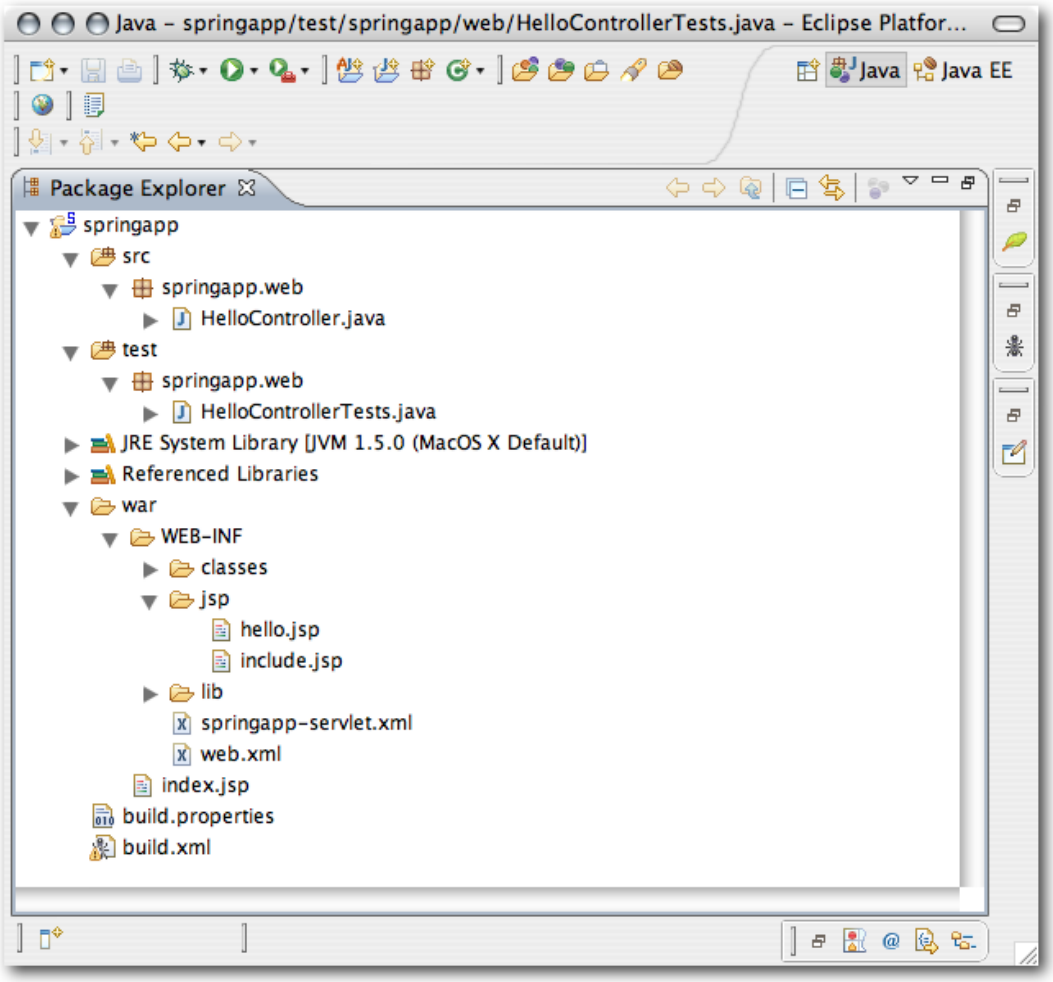
Let's take quick look at what we have created in Part 2.

- A header file **'include.jsp'**, the JSP that contains the taglib directives for the tag libraries we'll be using in our JSPs.

These are the existing artifacts we have changed in Part 2.

- The HelloControllerTests has been updated repeatedly as we make the controller reference the logical name of a view instead of its hard coded name and location.
- The page controller, HelloController, now references the view by its logical view name through the use of the 'InternalResourceViewResolver' defined in **'springapp-servlet.xml'**.

Find below a screen shot of what your project directory structure must look like after following the above instructions.



The project directory structure at the end of part 2

Prev
Chapter 1. Basic Application and Environment Setup

Home
Sponsored by SpringSource

Next
Chapter 3. Developing the Business Logic