Capitulo 2. Desarrollando y Configurando la Vista y el Controlador

Esta es la Parte 2 del tutorial paso a paso sobre como desarrollar una aplicacion web desde cero usando Spring Framework. En la Parte 1 hemos configurado el entorno y montado una aplicacion basica que ahora vamos a desarrollar.

Esto es lo que hemos implementado hasta ahora:

- Una pagina de inicio, "index.jsp", la pagina de bienvenida de nuestra aplicacion. Fue usada para comprobar que nuestra configuracion era correcta. Mas tarde la cambiaremos para proveer un enlance a nuestra aplicacion.
- Un controlador frontal, DispatcherServlet con el correspondiente archivo de configuracion "springapp-servlet.xml".
- Un controlador de pagina, HelloController, con funcionalidad limitada – simplemente devuelve un objeto ModelAndView. Actualmente tenemos un modelo vacio, mas tarde proveeremos un modelo completo.
- Una unidad de test para la pagina del controlador, HelloControllerTests, para verificar que el nombre de la vista es el que esperamos.
- Una vista, "hello.jsp", que de nuevo es extremadamente sencilla. Las buenas noticias son que el conjunto de la aplicacion funciona y que estamos listos para añadir mas funcionalidad.

2.1. Configurar JSTL y añadir un archivo de cabecera JSP

Vamos a usar la Libreria Estandar JSP (JSP Standard Tag Library - JSTL), asi que comencemos copiando los archivos JSTL que necesitamos en el directorio "WEB-INF/lib". Copia los archivos jstl.jar ubicado en el directorio "spring-framework-2.5/lib/j2ee" asi como standard.jar ubicado en el directorio "spring-framework-2.5/lib/jakarta-taglibs" al directorio "springapp/war/WEB-INF/lib".

Vamos a crear un archivo de 'cabecera' que sera embebido en todas las paginas JSP que vamos a escribir. Asi estaremos seguros que las mismas definiciones son incluidas en todos nuestros JSP al insertar el archivo de cabecera. Tambien vamos a poner todos nuestros archivos JSP en un directorio llamado "jsp" bajo el directorio "WEB-INF". Esto asegurara que nuestras vistas solo pueden ser accedidas a traves del controlador, por lo que no sera posible acceder a estas paginas a traves de una direccion URL. Esta estrategia podria no funcionar en algunos servidores de aplicaciones; si ese es tu caso, mueve el directorio "jsp" un nivel hacia arriba y usa "springapp/war/jsp" como el directorio de JSP en todos los ejemplos que encontraras a lo largo del tutorial, en lugar de "springapp/war/WEB-INF/jsp".

Primero creamos un archivo de cabecera para incluir en todos los archivos JSP que vamos a crear.

"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" %>
```

Ahora podemos actualizar "index.jsp" para que incluya este archivo, y puesto que estamos usando JSTL, podemos usar la etiqueta <c:redirect/> para recireccionar hacia nuestro controlador frontal: Controller. Esto significa que todas nuestras solicitudes a "index.jsp" iran a traves de nuestra aplicacion. Elimina los contenidos actuales de 'index.jsp' y reemplazalos con los siguientes:

"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"/>
```

Mueve "hello.jsp" al directorio "WEB-INF/jsp". Añade la misma directiva include que hemos añadido en "index.jsp" a "hello.jsp". Vamos a añadir tambien la fecha y hora actual, que seran leidas desde el modelo que pasaremos a la vista, y que mostraremos usando la etiqueta JSTL <c:out/>.

"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. Mejorar el controlador

Antes de actualizar la localizacion del JSP en nuestro controlador, actualicemos nuestra unidad de test. Sabemos que necesitamos actualizar la referencia a la vista con su nueva localizacion, "WEB-INF/jsp/hello.jsp". Tambien sabemos que deberia haber un objeto en el modelo mapeado a la clave "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);
    }
}
```

A continuacion, ejecutamos Ant con la opcion `'tests'` y nuestro test debe fallar.

```
$ 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
```

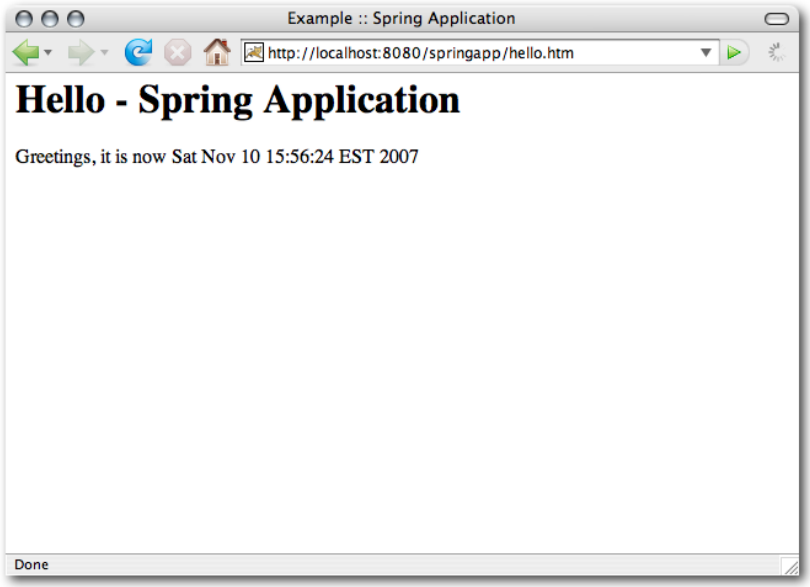
Ahora actualizamos `HelloController` configurando la referencia a la vista con su nueva localizacion, `"WEB-INF/jsp/hello.jsp"`, asi como la pareja clave/valor con la fecha y hora actual con la clave "now" y el valor: `'now'`.

`"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);
    }

}
```

Ejecutamos de nuevo Ant con la opcion `'tests'` y el test pasa.

Recuerda que el `Controller` ha sido previamente configurado en el archivo `"springapp-servlet.xml"` , por lo que estamos listos para probar nuestras mejoras despues de construir y desplegar el nuevo codigo. Ejecuta las tareas `'deploy reload'` en Ant, como hicimos previamente. Cuando introduzcamos la direccion http://localhost:8080/springapp/ en un navegador, deberia ejecutarse la pagina de bienvenida `"index.jsp"`, la cual deberia redireccionarnos a `"hello.htm"` que es manejada por `DispatcherServlet`, quien a su vez delega nuestra solicitud al controlador de pagina, que inserta la fecha y hora en el modelo y lo pone a disposicion de la vista `"hello.jsp"`.



La aplicacion actualizada

2.3. Separar la vista del controlador

Ahora el controlador especifica la ruta completa a la vista, lo cual crea una dependencia innecesaria entre el controlador y la vista. Idealmente queremos referirnos a la vista usando un nombre logico, permitiendonos intercambiar la vista sin tener que cambiar el controlador. Puedes crear este mapeo en un archivo de propiedades si estas usando `ResourceBundleViewResolver` y `SimpleUrlHandlerMapping`. Para el mapeo basico entre una vista y una localizacion, simplemente configura un prefijo y sufijo en `InternalResourceViewResolver`. Esta solucion es la que vamos a implantar ahora, por lo que modificamos "`springapp-servlet.xml`" y declaramos una entrada `'viewResolver'`. Eligiendo `JstlView` tendremos la oportunidad de usar JSTL en combinacion con paquetes de mensajes de idioma, los cuales nos ofreceran soporte para la internacionalizacion de la aplicacion.

"`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>
```

Actualizamos el nombre de la vista en la clase de pruebas del controlador `HelloControllerTests` por `'hello'` y relanzamos el test para comprobar que falla.

"`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);
    }
}
```

Ahora eliminamos el prefijo y sufijo del nombre de la vista en el controlador, dejando al controlador referirse a la vista por su nombre logico `"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);
    }

}
```

Relanzamos el test y ahora debe pasar.

Compilamos y desplegamos la aplicacion, y verificamos que todavia funciona.

2.4. Resumen

Echamos un vistazo rapido a lo que hemos creado en la Parte 2.

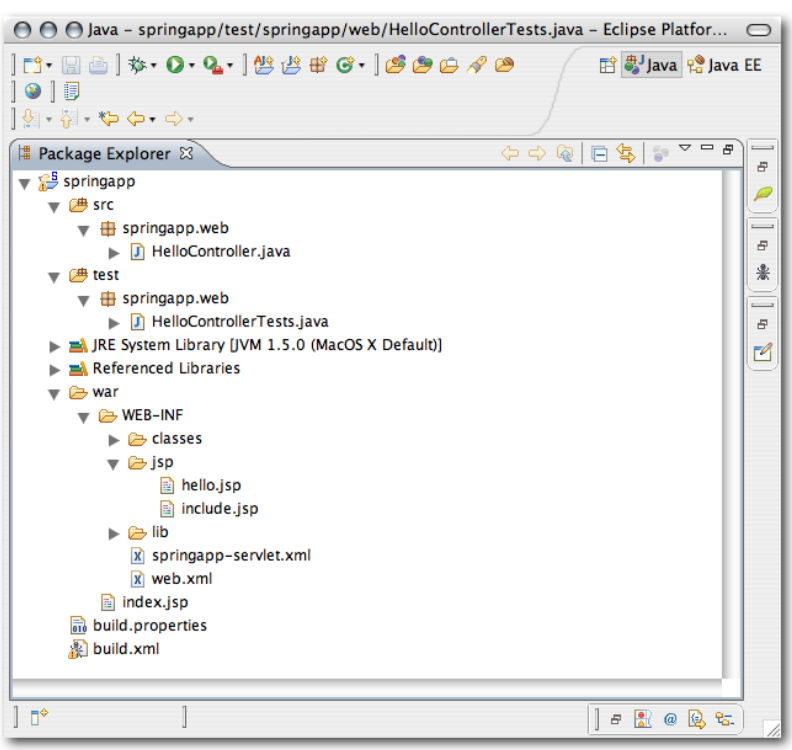
- Un archivo de cabecera "`include.jsp`", el archivo JSP que contiene la directiva taglib que usaremos en todos nuestros archivos JSPs.

Estos son los componentes de la aplicacion que hemos cambiado en la Parte 2.

- `HelloControllerTests` ha sido actualizado repetidamente para hacer al controlador referirse al nombre logico de la vista en lugar de a su localizacion y nombre completo.

  El controlador de pagina, `HelloController`, ahora hace referencia a la vista por su nombre logico mediante el uso del '`InternalResourceViewResolver`' definido en "`springapp-servlet.xml`".

Debajo puedes ver una captura de pantalla mostrando como debe aparecer tu estructura de directorios despues de seguir todas las instrucciones anteriores.



La estructura de directorios del proyecto al final de la parte 2