Capítulo 2. Desarrollando y Configurando la Vista y el Controlador

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

Esto es lo que hemos implementado hasta ahora:

- Una página de inicio, **'index.jsp'**, la página de bienvenida de nuestra aplicación. Fue usada para comprobar que nuestra configuración era correcta. Más tarde la cambiaremos para proveer un enlance a nuestra aplicación.
- Un controlador frontal, `DispatcherServlet`, con el correspondiente archivo de configuración **'app-config.xml'**.
- Un controlador de página, `HelloController`, con funcionalidad limitada – simplemente devuelve un objeto `ModelAndView`. Actualmente tenemos un modelo vacío, más tarde proveeremos un modelo completo.
- Una unidad de test para la página 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 aplicación funciona y que estamos listos para añadir más funcionalidad.

2.1. Configurar JSTL y añadir un archivo de cabecera JSP

Vamos a usar la Librería Estandar JSP (JSP Standard Tag Library - JSTL), así que comencemos definiendo la dependencia que necesitamos en el fichero **'pom.xml'**.

Group Id	Artifact Id	Version
jstl	jstl	1.2
taglibs	standard	1.1.2

Vamos a crear un archivo de 'cabecera' que será embebido en todas las paginas JSP que escribamos después. Así estaremos seguros de que las mismas definiciones son incluidas en todos nuestros JSP al insertar el archivo de cabecera. También vamos a poner todos nuestros archivos JSP en un directorio llamado **'views'** bajo el directorio **'src/main/webapp/WEB-INF'**. Esto asegurará que nuestras vistas sólo puedan ser accedidas a través del controlador, por lo que no será posible acceder a estas páginas a través de una dirección URL. Esta estrategia podría no funcionar en algunos servidores de aplicaciones; si ése es tu caso, mueve el directorio **'views'** un nivel hacia arriba y usa **'src/main/webapp/views'** como el directorio de JSP en todos los ejemplos que encontrarás a lo largo del tutorial, en lugar de **'src/main/webapp/WEB-INF/views'**.

Primero creamos un archivo de cabecera para incluir en todos los archivos JSP que crearemos con posterioridad.

'springapp/src/main/webapp/WEB-INF/views/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 redireccionar hacia nuestro controlador frontal: `Controller`. Esto significa que todas nuestras solicitudes a **'index.jsp'** se resolverán a través de dicho controlador. Elimina los contenidos actuales de **'index.jsp'** y reemplázalos con los siguientes:

'springapp/src/main/webapp/index.jsp':

```
<%@ include file="/WEB-INF/views/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 **'src/main/webapp/WEB-INF/views'**. Añade la misma directiva include que hemos añadido en **'index.jsp'** a **'hello.jsp'**. Vamos a añadir también la fecha y hora actual, que serán leídas desde el modelo que pasaremos a la vista, y que mostraremos usando la etiqueta JSTL **<c:out/>**.

'springapp/src/main/webapp/WEB-INF/views/hello.jsp':

```
<%@ include file="/WEB-INF/views/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 localización del JSP en nuestro controlador, actualicemos nuestra unidad de test. Sabemos que necesitamos actualizar la referencia a la vista con su nueva localización, **'WEB-INF/views/hello.jsp'**. También sabemos que debería haber un objeto en el modelo mapeado a la clave "now".

'springapp/src/test/java/com/companyname/springapp/web/HelloControllerTests.java':

```
package com.companyname.springapp.web;

import static org.junit.Assert.*;

import org.junit.Test;
import org.springframework.web.servlet.ModelAndView;


public class HelloControllerTests {

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

    }

}
```

A continuación, ejecutamos el test y debería fallar.

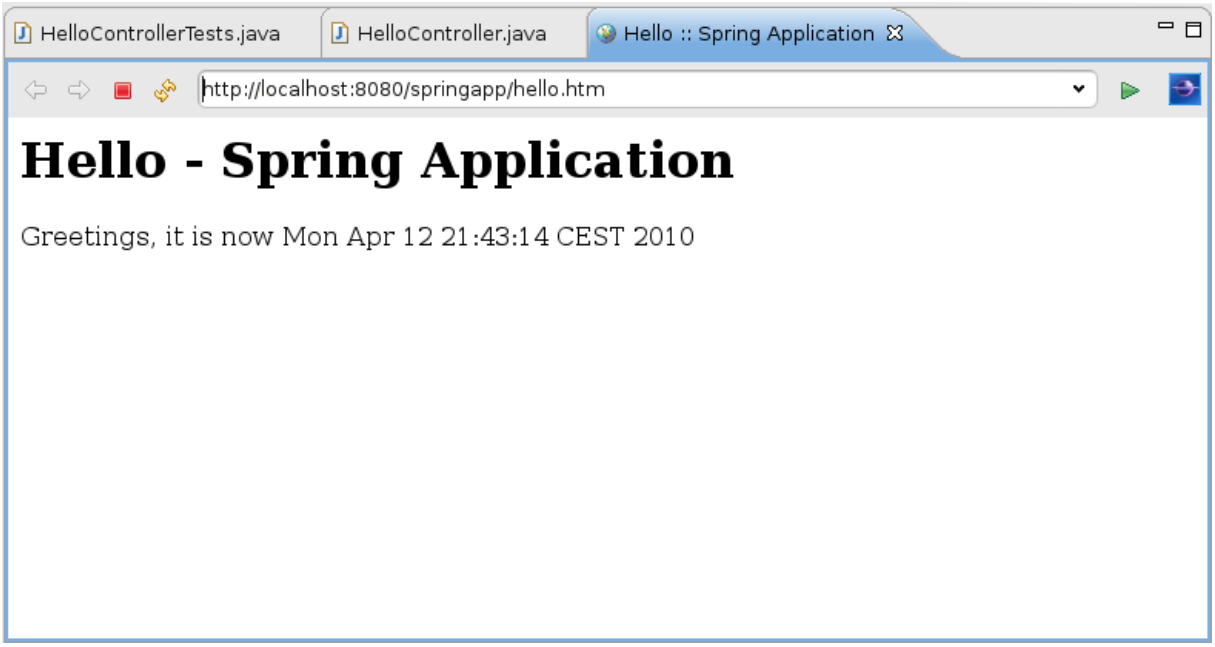
Ahora actualizamos `HelloController` configurando la referencia a la vista con su nueva localización, **'WEB-INF/views/hello.jsp'**, así como la pareja clave/valor con la fecha y hora actual con la clave "now" y el valor: 'now'.

'springapp/src/main/java/com/companyname/springapp/web/HelloController.java':

```
package com.companyname.springapp.web;

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

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 org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.servlet.ModelAndView;

@Controller
public class HelloController {

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

    @RequestMapping(value="/hello.htm")
    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/views/hello.jsp", "now", now);

    }
}
```

Ejecutamos de el test y ahora debería pasar.

Estamos ahora listos para probar nuestras mejoras después sobre el servidor. Cuando introduzcamos la dirección http://localhost:8080/springapp/ en un navegador, debería ejecutarse la página de bienvenida **'index.jsp'**, la cual debería redireccionarnos a **'hello.htm'** que es manejada por el `DispatcherServlet`, quien a su vez delega nuestra solicitud al controlador `HelloController`, que inserta la fecha y hora en el modelo y las pone a disposición de la vista **'hello.jsp'**.



La aplicación 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 lógico, permitiéndonos intercambiar la vista sin tener que cambiar el controlador. Puedes crear este mapeo en un archivo de propiedades si estas usando `ResourceBundleViewResolver` y `SimpleUrlHandlerMapping`. Otra opción para el mapeo básico entre una vista y una localización, consiste en simplemente configurar un prefijo y sufijo en `InternalResourceViewResolver`. Esta solución es la que vamos a implantar ahora, por lo que modificamos **'app-config.xml'** y declaramos una entrada `'viewResolver'`. Eligiendo `JstlView` tendremos la oportunidad de usar JSTL en combinación con paquetes de mensajes de idioma, los cuales nos ofreceran soporte para la internacionalización de la aplicación.

'springapp/src/main/webapp/WEB-INF/spring/app-config.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"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:mvc="http://www.springframework.org/schema/mvc"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
       http://www.springframework.org/schema/context
       http://www.springframework.org/schema/context/spring-context-3.0.xsd
       http://www.springframework.org/schema/mvc
       http://www.springframework.org/schema/mvc/spring-mvc-3.0.xsd">

       <!-- Scans the classpath of this application for @Components to deploy as beans -->
       <context:component-scan base-package="com.companyname.springapp.web" />

       <!-- Configures the @Controller programming model -->
       <mvc:annotation-driven/>

       <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/views/"></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/src/test/java/com/companyname/springapp/web/HelloControllerTests.java':

```
package com.companyname.springapp.web;

import static org.junit.Assert.*;

import org.junit.Test;
```

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


public class HelloControllerTests {

    @Test
    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 que el controlador se refiera a la vista por su nombre lógico `"hello"`.

'springapp/src/main/java/com/companyname/springapp/web/HelloController.java':

```
package com.companyname.springapp.web;

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

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 org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.servlet.ModelAndView;

@Controller
public class HelloController {

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

    @RequestMapping(value="/hello.htm")
    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 aplicación, y verificamos que todavía funciona.

2.4. Resumen

Echemos un vistazo rápido a lo que hemos creado en la Parte 2.

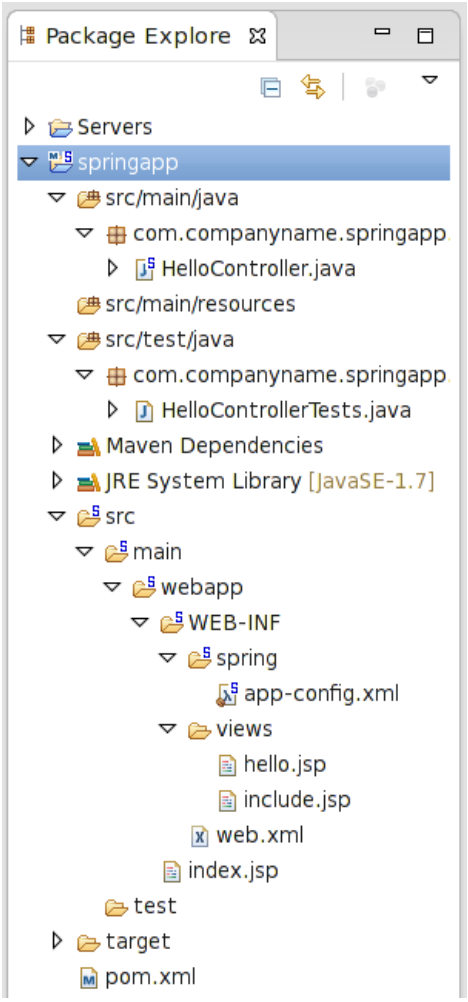
- Un archivo de cabecera **'include.jsp'**, el archivo JSP que contiene la directiva taglib que usaremos en todos nuestros archivos JSPs.

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

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

  El controlador de página, `HelloController`, ahora hace referencia a la vista por su nombre lógico mediante el uso del `'InternalResourceViewResolver'` definido en **'app-config.xml'**.

A continuación puedes ver una captura de pantalla que muestra el aspecto que debería tener la estructura de directorios del proyecto después de seguir todas las instrucciones anteriores.



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