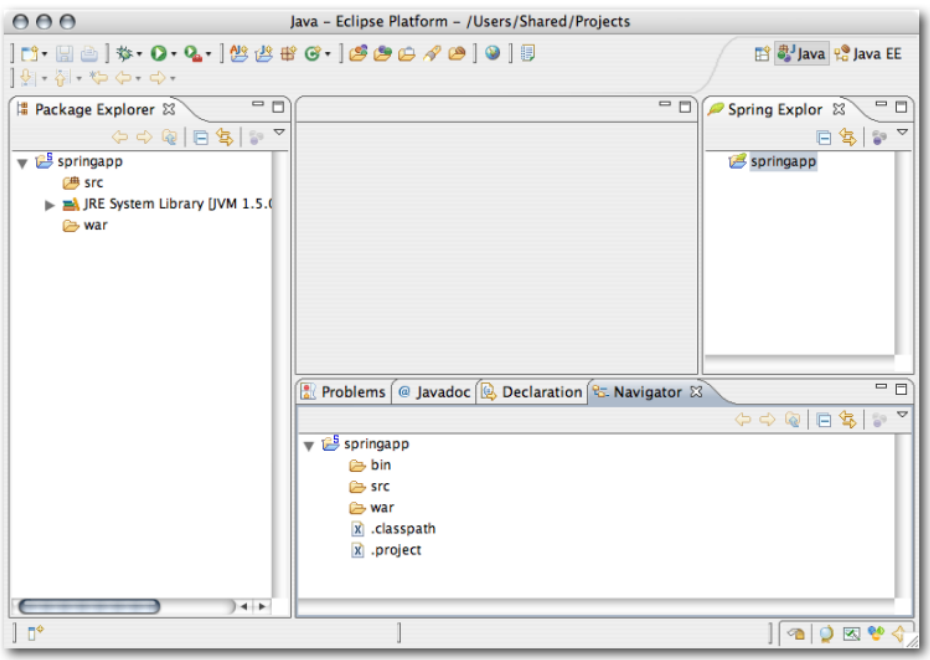
Capitulo 1. Aplicacion Base y Configuracion del Entorno

1.1. Crear la estructura de directorios del proyecto

Necesitamos crear un lugar donde alojar todos los archivos que vayamos creando, asi que comenzaremos creando un directorio llamado "springapp". La decision sobre donde crear este directorio esta en tu mano; nosotros hemos creado el nuestro dentro del directorio "Projects" que anteriormente habiamos creado en "home", por lo que la ruta completa a nuestro directorio del proyecto es "$HOME/Projects/springapp". Dentro de este directorio, vamos a crear un subdirectorio llamado "src" donde guardar todos los archivos de codigo fuente Java. De nuevo en el directorio "$HOME/Projects/springapp" creamos otro subdirectorio llamado "war". Este directorio almacenara todo lo que debe incluir el archivo WAR que usaremos para almacenar y desplegar rapidamente nuestra aplicacion. Todos los archivos que no sean codigo fuente Java, como archivos JSP y de configuracion, se alojaran en el directorio "war".

Debajo puedes ver una captura de pantalla mostrando como quedaria la estructura de directorios si has seguido las instrucciones correctamente. *(La imagen muestra dicha estructura desde el IDE Eclipse: no se necesita usar Eclipse para completar este tutorial, pero usandolo podras hacerlo de manera mucho mas sencilla).*



La estructura de directorios del proyecto

1.2. Crear "index.jsp"

Puesto que estamos creando una aplicacion web, comencemos creando un archivo JSP muy simple llamado "index.jsp" en el directorio "war". El archivo "index.jsp" es el punto de entrada a nuestra aplicacion.

"springapp/war/index.jsp":

```
<html>
  <head><title>Example :: Spring Application</title></head>
  <body>
    <h1>Example - Spring Application</h1>
    <p>This is my test.</p>
  </body>
</html>
```

Para tener una aplicacion web completa vamos a crear un directorio llamado "WEB-INF" dentro del directorio "war", y dentro de este nuevo directorio creamos un archivo llamado "web.xml".

"springapp/war/WEB-INF/web.xml":

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

<web-app version="2.4"
         xmlns="http://java.sun.com/xml/ns/j2ee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
         http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd" >

  <welcome-file-list>
    <welcome-file>
```

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

</web-app>
```

1.3. Desplegar la aplicacion en Tomcat

Escribamos ahora un script para Ant que utilizaremos a lo largo de todo este tutorial. Este script para Ant contendra la informacion necesaria para compilar, construir y desplegar la aplicacion automaticamente. Ademas, otro script para Ant sera creado y usado para tareas especificas del servidor.

"springapp/build.xml":

```
<?xml version="1.0"?>

<project name="springapp" basedir="." default="usage">
    <property file="build.properties"/>

    <property name="src.dir" value="src"/>
    <property name="web.dir" value="war"/>
    <property name="build.dir" value="${web.dir}/WEB-INF/classes"/>
    <property name="name" value="springapp"/>

    <path id="master-classpath">
        <fileset dir="${web.dir}/WEB-INF/lib">
            <include name="*.jar"/>
        </fileset>
        <!-- We need the servlet API classes: -->
        <!--  * for Tomcat 5/6 use servlet-api.jar -->
        <!--  * for other app servers - check the docs -->
        <fileset dir="${appserver.lib}">
            <include name="servlet*.jar"/>
        </fileset>
        <pathelement path="${build.dir}"/>
    </path>

    <target name="usage">
        <echo message=""/>
        <echo message="${name} build file"/>
        <echo message="-----------------------------------"/>
        <echo message=""/>
        <echo message="Available targets are:"/>
        <echo message=""/>
        <echo message="build     --> Build the application"/>
        <echo message="deploy    --> Deploy application as directory"/>
        <echo message="deploywar --> Deploy application as a WAR file"/>
        <echo message="install   --> Install application in Tomcat"/>
        <echo message="reload    --> Reload application in Tomcat"/>
        <echo message="start     --> Start Tomcat application"/>
        <echo message="stop      --> Stop Tomcat application"/>
        <echo message="list      --> List Tomcat applications"/>
        <echo message=""/>
    </target>

    <target name="build" description="Compile main source tree java files">
        <mkdir dir="${build.dir}"/>
        <javac destdir="${build.dir}" source="1.5" target="1.5" debug="true"
               deprecation="false" optimize="false" failonerror="true">
            <src path="${src.dir}"/>
            <classpath refid="master-classpath"/>
        </javac>
    </target>

    <target name="deploy" depends="build" description="Deploy application">
        <copy todir="${deploy.path}/${name}" preservelastmodified="true">
            <fileset dir="${web.dir}">
                <include name="**/*.*"/>
            </fileset>
        </copy>
    </target>

    <target name="deploywar" depends="build" description="Deploy application as a WAR file">
        <war destfile="${name}.war"
             webxml="${web.dir}/WEB-INF/web.xml">
            <fileset dir="${web.dir}">
                <include name="**/*.*"/>
            </fileset>
        </war>
        <copy todir="${deploy.path}" preservelastmodified="true">
            <fileset dir=".">
                <include name="*.war"/>
            </fileset>
        </copy>
    </target>

<!-- ============================================================== -->
<!-- Tomcat tasks - remove these if you don't have Tomcat installed -->
<!-- ============================================================== -->

    <path id="catalina-ant-classpath">
        <!-- We need the Catalina jars for Tomcat -->
        <!--  * for other app servers - check the docs -->
        <fileset dir="${appserver.lib}">
            <include name="catalina-ant.jar"/>
        </fileset>
    </path>

    <taskdef name="install" classname="org.apache.catalina.ant.InstallTask">
        <classpath refid="catalina-ant-classpath"/>
    </taskdef>
    <taskdef name="reload" classname="org.apache.catalina.ant.ReloadTask">
        <classpath refid="catalina-ant-classpath"/>
```

```
    </taskdef>
    <taskdef name="list" classname="org.apache.catalina.ant.ListTask">
        <classpath refid="catalina-ant-classpath"/>
    </taskdef>
    <taskdef name="start" classname="org.apache.catalina.ant.StartTask">
        <classpath refid="catalina-ant-classpath"/>
    </taskdef>
    <taskdef name="stop" classname="org.apache.catalina.ant.StopTask">
        <classpath refid="catalina-ant-classpath"/>
    </taskdef>

    <target name="install" description="Install application in Tomcat">
        <install url="${tomcat.manager.url}"
                 username="${tomcat.manager.username}"
                 password="${tomcat.manager.password}"
                 path="/${name}"
                 war="${name}"/>
    </target>

    <target name="reload" description="Reload application in Tomcat">
        <reload url="${tomcat.manager.url}"
                 username="${tomcat.manager.username}"
                 password="${tomcat.manager.password}"
                 path="/${name}"/>
    </target>

    <target name="start" description="Start Tomcat application">
        <start url="${tomcat.manager.url}"
                 username="${tomcat.manager.username}"
                 password="${tomcat.manager.password}"
                 path="/${name}"/>
    </target>

    <target name="stop" description="Stop Tomcat application">
        <stop url="${tomcat.manager.url}"
                 username="${tomcat.manager.username}"
                 password="${tomcat.manager.password}"
                 path="/${name}"/>
    </target>

    <target name="list" description="List Tomcat applications">
        <list url="${tomcat.manager.url}"
                 username="${tomcat.manager.username}"
                 password="${tomcat.manager.password}"/>
    </target>

<!-- End Tomcat tasks -->

</project>
```

Si estas usando un servidor de aplicaciones web distinto, puedes eliminar las tareas especificas de Tomcat que hay al final del script. Tendras que confiar en que tu servidor soporte despliegue en caliente, o de lo contrario tendras que parar e iniciar tu aplicacion manualmente.

Si estas usando un IDE, es posible que encuentre errores en el archivo `"build.xml"`, como los objetivos de Tomcat. Ignoralos, el archivo listado arriba es correcto.

El archivo de script para Ant listado arriba contiene toda la configuracion de objetivos que vamos a necesitar para hacer nuestro esfuerzo de desarrollo mas facil. No vamos a explicar este script en detalle, puesto que la mayor parte de el es sobre todo configuracion estandard para Ant y Tomcat. Selecciona todo el script, copialo, y pegalo en un nuevo archivo de texto llamado **`"build.xml"`** que debes guardar en tu directorio principal de desarrollo. Tambien necesitamos un archivo **`"build.properties"`** que modificaremos para que coincida con nuestra instalacion del servidor. Este archivo permanecera en el mismo directorio donde hemos guardado el archivo **`"build.xml"`**.

`"springapp/build.properties"`:

```
# Ant properties for building the springapp

appserver.home=${user.home}/apache-tomcat-6.0.14
# for Tomcat 5 use $appserver.home}/server/lib
# for Tomcat 6 use $appserver.home}/lib
appserver.lib=${appserver.home}/lib

deploy.path=${appserver.home}/webapps

tomcat.manager.url=http://localhost:8080/manager
tomcat.manager.username=tomcat
tomcat.manager.password=s3cret
```

*Si estas en un equipo donde no eres el usuario propietario de la instalacion de Tomcat, entonces dicho usuario debe garantizarte el acceso sin restriccciones al directorio `"webapps"`, o el propietario debe crear un nuevo directorio llamado `"springapp"` en el directorio `"webapps"` de Tomcat, y crear los permisos necesarios para que puedas desplegar la aplicacion en este directorio. En Linux, escribe el comando **`'chmod a+rwx springapp'`** para dar todos los permisos a este directorio.*

Para crear un usuario de Tomcat llamado 'tomcat' con contraseña 's3cret', edita el archivo de usuarios de Tomcat `"appserver.home/conf/tomcat-users.xml"` y añade una nueva entrada de usuario.

```
<?xml version='1.0' encoding='utf-8'?>
<tomcat-users>
  <role rolename="manager"/>
  <user username="tomcat" password="s3cret" roles="manager"/>
</tomcat-users>
```

Ahora ejecutaremos Ant para estar seguros que todo funciona bien. Debes estar situado en el directorio **`"springapp"`**.

Abre una consola de shell (o prompt) y ejecuta el comando **'ant'** .

```
$ ant
Buildfile: build.xml

usage:
     [echo]
     [echo] springapp build file
     [echo] -----------------------------------
     [echo]
     [echo] Available targets are:
     [echo]
     [echo] build     --> Build the application
     [echo] deploy    --> Deploy application as directory
     [echo] deploywar --> Deploy application as a WAR file
     [echo] install   --> Install application in Tomcat
     [echo] reload    --> Reload application in Tomcat
     [echo] start     --> Start Tomcat application
     [echo] stop      --> Stop Tomcat application
     [echo] list      --> List Tomcat applications
     [echo]

BUILD SUCCESSFUL
Total time: 2 seconds
```

La ultima cosa que debemos hacer es construir y desplegar la aplicacion. Para ello, simplemente ejecuta Ant y especifica `'deploy'` o `'deploywar'` como objetivo.

```
$ ant deploy
Buildfile: build.xml

build:
    [mkdir] Created dir: /Users/trisberg/Projects/springapp/war/WEB-INF/classes

deploy:
     [copy] Copying 2 files to /Users/trisberg/apache-tomcat-5.5.17/webapps/springapp

BUILD SUCCESSFUL
Total time: 4 seconds
```

1.4. Comprobar que la aplicacion funciona

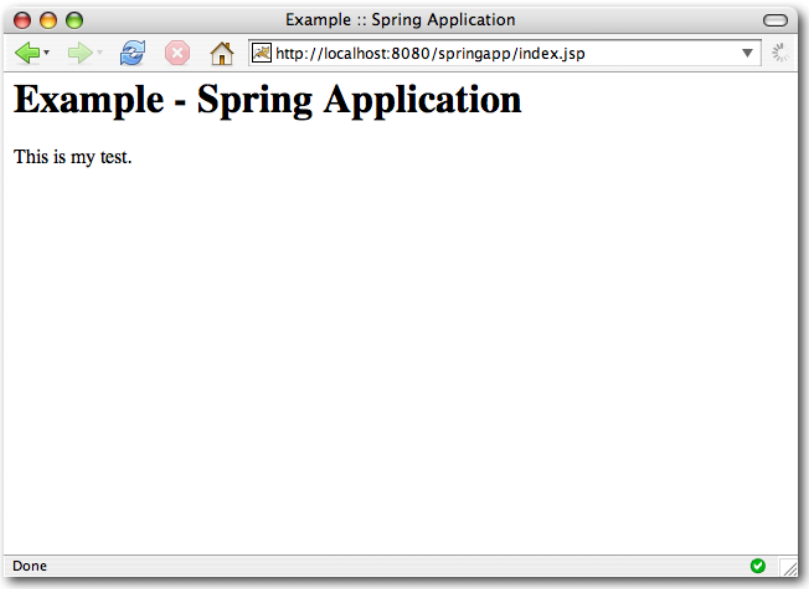
Inicia tomcat ejecutando el archivo `"${appserver.home}/bin/startup.bat"`. Para asegurarnos que podemos acceder a la aplicacion, ejecuta la tarea `'list'` desde nuestro archivo de contruccion Ant para ver si Tomcat puede encontrar la aplicacion.

```
$ ant list
Buildfile: build.xml

list:
     [list] OK - Listed applications for virtual host localhost
     [list] /springapp:running:0:springapp
     [list] /manager:running:0:manager
     [list] /:running:0:ROOT
     [list] /docs:running:0:docs
     [list] /examples:running:0:examples
     [list] /host-manager:running:0:host-manager

BUILD SUCCESSFUL
Total time: 3 seconds
```

Ahora puedes abrir un navegador y acceder a la pagina de inicio de la aplicacion en la siguiente URL: http://localhost:8080/springapp/index.jsp.



La Pagina de inicio de la aplicacion

1.5. Descargar Spring Framework

Si aun no has descargado Spring Framework, ahora es el momento de hacerlo. Nosotros usamos actualmente la version 2.5 de Spring Framework, que puede ser descargada desde http://www.springframework.org/download. Descomprime el archivo que has descargado en una carpeta, ya que mas tarde vamos a usar varios archivos que estan en su interior.

Esto completa la configuracion de entorno necesaria, asi que ya podemos comenzar a desarrollar nuestra aplicacion Spring Framework MVC.

1.6. Modificar "web.xml" en el directorio "WEB-INF"

Situate en el directorio "springapp/war/WEB-INF". Modifica el archivo "web.xml" que hemos creado anteriormente. Vamos a definir un DispatcherServlet (tambien llamado 'Controlador Frontal' (Crupi et al)). Su mision sera controlar hacia donde seran enrutadas todas nuestras solicitudes basandose en informacion que introduciremos posteriormente. La definicion del servlet tendra como acompañante una entrada <servlet-mapping/> que mapeara las URL que queremos que apunten a nuestro servlet. Hemos decidido permitir que cualquier URL con una extension de tipo '.htm' sea enrutada hacia el servlet 'springapp' (DispatcherServlet).

"springapp/war/WEB-INF/web.xml":

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

<web-app version="2.4"
         xmlns="http://java.sun.com/xml/ns/j2ee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
         http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd" >

  <servlet>
    <servlet-name>springapp</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>

  <servlet-mapping>
    <servlet-name>springapp</servlet-name>
    <url-pattern>*.htm</url-pattern>
  </servlet-mapping>

  <welcome-file-list>
    <welcome-file>
      index.jsp
    </welcome-file>
  </welcome-file-list>

</web-app>
```

A continuacion, crea un archivo llamado "springapp-servlet.xml" en el directorio "springapp/war/WEB-INF". Este archivo contiene las definiciones de beans (POJO's) usados por DispatcherServlet. Este archivo es el WebApplicationContext donde situaremos todos los componentes web. El nombre de este archivo esta determinado por el valor del elemento <servlet-name/> en "web.xml", con la palabra '-servlet' agregada al final (por lo tanto "springapp-servlet.xml"). Esta es la convencion estardar para nombrar archivos del framework Spring MVC. Ahora añade una definicion de bean llamada '/hello.htm' y especifica su clase como springapp.web.HelloController. Esto define el controlador que nuestra aplicacion usara para dar servicio a solicitudes con el correspondiente mapeo de URL de '/hello.htm'. El framework Spring MVC usa una clase que implementa la interface HandlerMapping para definir el mapeo entre la URL solicitada y el objeto que va a manejar la solicitud (manejador). Al contrario que

DispatcherServlet, HelloController es el encargado de manejar las solicitudes para una pagina concreta del sitio web, y es conocido como el 'Controlador de Pagina' (Fowler). El HandlerMapping por defecto que DispatcherServlet usa es BeanNameUrlHandlerMapping; esta clase usa el nombre del bean para mapear la URL de la solicitud, por lo que DispatcherServlet conocera que controlador debe ser invocado para manejar diferentes URLs.

"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-2.5.xsd">

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

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

</beans>
```

1.7. Copiar librerias a "WEB-INF/lib"

Primero crea un directorio llamado **"lib"** dentro del directorio **"war/WEB-INF"** . Ahora, desde la distribucion de Spring, copia los archivos **spring.jar** (desde **spring-framework-2.5/dist**) y **spring-webmvc.jar** (desde **spring-framework-2.5/dist/modules**) al recien creado directorio **"war/WEB-INF/lib"**. Ademas, copia el archivo **commons-logging.jar** (desde **spring-framework-2.5/lib/jakarta-commons**) tambien al directorio **"war/WEB-INF/lib"**. Estos archivos jar seran desplegados y usados en el servidor durante el proceso de construccion.

1.8. Crear el Controlador

Vamos a crear una instancia de la clase Controller – a la que llamaremos HelloController, y que estara definida dentro del paquete 'springapp.web'. Primero, crea los directorios necesarios para que coincidan con el arbol del paquete. A continuacion, crea el archivo **"HelloController.java"** y situalo en el recien citado directorio **"src/springapp/web"** .

"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;

public class HelloController implements Controller {

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

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

        logger.info("Returning hello view");

        return new ModelAndView("hello.jsp");
    }

}
```

Esta implementacion del controlador Controller es muy basica. Mas tarde la iremos expandiendo, asi como extendiendo parte de la implementacion base provista por Spring. En Spring Web MVC, Controller *maneja* las solicitudes y devuelve un objeto ModelAndView - en este caso, uno llamado **"hello.jsp"** el cual es ademas el nombre del archivo JSP que vamos a crear a continuacion. El modelo que esta clase devuelve es resuelto via ViewResolver. Puesto que no hemos definido explicitamente un ViewResolver, vamos a obtener uno por defecto de Spring que simplemente redigira a una direccion URL que coincida con el nombre de la vista especificada. Mas tarde modificaremos esto. Ademas, hemos especificado un logger de manera que podemos verificar que pasamos por el manejador en cada momento. Usando Tomcat, estos mensajes de log deben mostrarse en el arhivo de log **"catalina.out"** que puede ser encontrado en el directorio **"${appserver.home}/log"** de tu instalacion de Tomcat.

*Si estas usando un IDE, configura las dependencias del proyecto añadiendo los archivos jar desde el directorio **"lib"**. Añade ademas el archivo **servlet-api.jar** desde el directorio **"lib"** de tu contenedor de servlets ('${appserver.lib}'). Añadiendo estos archivos a las dependencias del proyecto, deberian funcionar todas las sentencias import del archivo **"HelloController.java"**.*

1.9. Escribir un test para el Controlador

Los tests son una parte vital del desarrollo de software. Es ademas una de las practicas fundamentales en Desarrollo Agil. El mejor momento para escribir tests es durante el desarrollo, no despues, de manera que aunque nuestro controlador no contiene logica demasiado compleja vamos a escribir un test para probarlo. Esto nos permitira hacer cambios en el futuro con total seguridad. Vamos a crear un nuevo directorio bajo **"springapp"** llamado **"test"**. Aqui es donde alojaremos todos nuestros tests, en una estructura de paquetes que sera identica a la estructura de paquetes que tenemos en **"springapp/src"**.

Crea una clase de test llamada **"HelloControllerTests"** y haz que extienda la clase de JUnit TestCase. Esta es una

unidad de test que verifica que el nombre de vista devuelto por `handleRequest()` coincide con el nombre de la vista que esperamos: **"hello.jsp"**.

"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.jsp", modelAndView.getViewName());
    }
}
```

Para ejecutar el test (y todos los tests que escribamos en el futuro), necesitamos añadir una tarea de test a nuestro script de Ant **"build.xml"**. Primero, copiamos **junit-3.8.2.jar** desde **"spring-framework-2.5/lib/junit"** al directorio **"war/WEB-INF/lib"**. En lugar de crear una tarea unica para compilar los tests y a continuacion ejecutarlos, vamos a separar el proceso en dos tareas diferentes: `'buildtests'` y `'tests'`, el cual depende de `'buildtests'`.

*Si estas usando un IDE, tal vez prefieras ejecutar los tests desde el IDE. Configura las dependencias de tu proyecto añadiendo **junit-3.8.2.jar**.*

"springapp/build.xml":

```
    <property name="test.dir" value="test"/>

    <target name="buildtests" description="Compile test tree java files">
        <mkdir dir="${build.dir}"/>
        <javac destdir="${build.dir}" source="1.5" target="1.5" debug="true"
            deprecation="false" optimize="false" failonerror="true">
            <src path="${test.dir}"/>
            <classpath refid="master-classpath"/>
        </javac>
    </target>

    <target name="tests" depends="build, buildtests" description="Run tests">
        <junit printsummary="on"
            fork="false"
            haltonfailure="false"
            failureproperty="tests.failed"
            showoutput="true">
            <classpath refid="master-classpath"/>
            <formatter type="brief" usefile="false"/>

            <batchtest>
                <fileset dir="${build.dir}">
                    <include name="**/*Tests.*"/>
                </fileset>
            </batchtest>

        </junit>

        <fail if="tests.failed">
            tests.failed=${tests.failed}
            ***********************************************************
            ***********************************************************
            ****  One or more tests failed!  Check the output ...  ****
            ***********************************************************
            ***********************************************************
        </fail>
    </target>
```

Ahora ejecuta la tarea de Ant `'tests'` y el test debe pasar.

```
$ ant tests
Buildfile: build.xml

build:

buildtests:
    [javac] Compiling 1 source file to /Users/Shared/Projects/springapp/war/WEB-INF/classes

tests:
    [junit] Running springapp.web.HelloWorldControllerTests
    [junit] Oct 30, 2007 11:31:43 PM springapp.web.HelloController handleRequest
    [junit] INFO: Returning hello view
    [junit] Tests run: 1, Failures: 0, Errors: 0, Time elapsed: 0.03 sec
    [junit] Testsuite: springapp.web.HelloWorldControllerTests
    [junit] Tests run: 1, Failures: 0, Errors: 0, Time elapsed: 0.03 sec

    [junit] ------------- Standard Error -----------------
    [junit] Oct 30, 2007 11:31:43 PM springapp.web.HelloController handleRequest
    [junit] INFO: Returning hello view
    [junit] ------------- ---------------- ---------------

BUILD SUCCESSFUL
Total time: 2 seconds
```

Otra de las mejores practicas dentro del Desarrollo Agil es *Integracion Continua*. Es una muy buena idea asegurar que tus tests se ejecutan con cada construccion (idealmente como construccion automatica del proyecto) de manera que sepas que la logica de tu aplicacion se comporta como se espera de ella a traves de la evolucion del codigo.

1.10. Crear la Vista

Ahora es el momento de crear nuestra primera vista. Como hemos mencionado antes, estamos redirigiendo hacia una pagina JSP llamada "hello.jsp". Para empezar, crearemos este fichero en el directorio "war".

"springapp/war/hello.jsp":

```
<html>
  <head><title>Hello :: Spring Application</title></head>
  <body>
    <h1>Hello - Spring Application</h1>
    <p>Greetings.</p>
  </body>
</html>
```

1.11. Compilar y desplegar la aplicacion

Ejecuta la tarea 'deploy' en Ant (la cual invoca la tarea 'build'), y a continuacion 'reload' del archivo "build.xml" . Esto forzara la construccion y recarga de la aplicacion en Tomcat. Tenemos que comprobar la salida de Ant y los logs de Tomcat para verificar cualquier posible error de despliegue – como errores tipograficos en los archivos de arriba y clases o archivos jar no encontrados.

Aqui tienes un ejemplo de salida del script de construccion Ant:

```
$ ant deploy reload
Buildfile: build.xml

build:
    [mkdir] Created dir: /Users/trisberg/Projects/springapp/war/WEB-INF/classes
    [javac] Compiling 1 source file to /Users/trisberg/Projects/springapp/war/WEB-INF/classes

deploy:
     [copy] Copying 7 files to /Users/trisberg/apache-tomcat-5.5.17/webapps/springapp

BUILD SUCCESSFUL
Total time: 3 seconds
$ ant reload
Buildfile: build.xml

reload:
   [reload] OK - Reloaded application at context path /springapp

BUILD SUCCESSFUL
Total time: 2 seconds
```

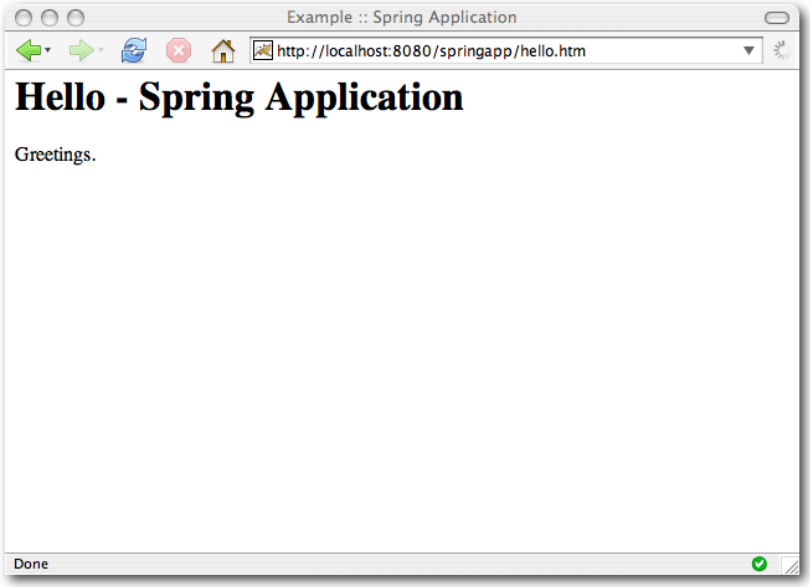
Y aqui un extracto del archivo de log "catalina.out".

```
Oct 30, 2007 11:43:09 PM org.springframework.web.servlet.FrameworkServlet initServletBean
INFO: FrameworkServlet 'springapp': initialization started
Oct 30, 2007 11:43:09 PM org.springframework.context.support.AbstractApplicationContext prepareRefresh
INFO: Refreshing org.springframework.web.context.support.XmlWebApplicationContext@6576d5: display name
[WebApplicationContext for namespace 'springapp-servlet']; startup date [Tue Oct 30 23:43:09 GMT 2007];
...
...
Oct 30, 2007 11:43:09 PM org.springframework.web.servlet.FrameworkServlet initServletBean
INFO: FrameworkServlet 'springapp': initialization completed in 150 ms
```

1.12. Probar la aplicacion

Probemos esta nueva version de la aplicacion.

Abre un navegador y navega hasta http://localhost:8080/springapp/hello.htm.



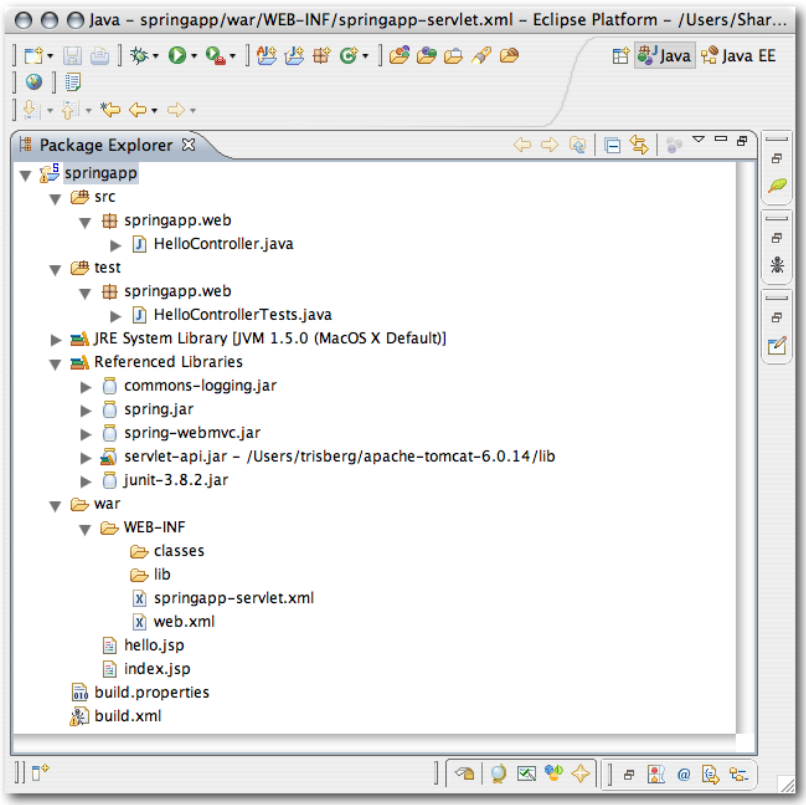
La aplicacion actualizada

1.13. Resumen

Echemos un vistazo rapido a las partes de nuestra aplicacion que hemos creado 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.

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 1