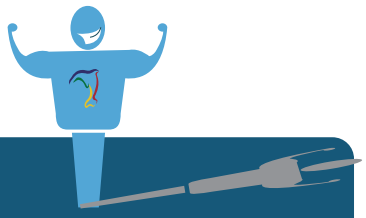
autentia

Avenida de Castilla,1 - Edificio Best Point - Oficina 21B
28830 San Fernando de Henares (Madrid)
tel./fax: +34 91 675 33 06
info@autentia.com - **www.autentia.com**

¿**Qué ofrece** Autentia Real Business Solutions S.L?

Somos su empresa de **Soporte a Desarrollo Informático**.
Ese apoyo que siempre quiso tener...

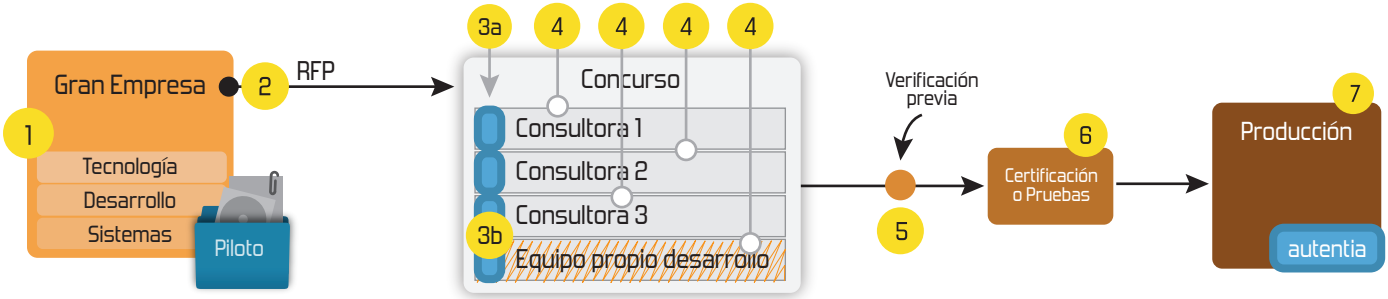
1. Desarrollo de componentes y proyectos a medida



2. Auditoría de código y recomendaciones de mejora

3. Arranque de proyectos basados en nuevas tecnologías

1. Definición de frameworks corporativos.
2. Transferencia de conocimiento de nuevas arquitecturas.
3. Soporte al arranque de proyectos.
4. Auditoría preventiva periódica de calidad.
5. Revisión previa a la certificación de proyectos.
6. Extensión de capacidad de equipos de calidad.
7. Identificación de problemas en producción.



4. Cursos de formación (impartidos por desarrolladores en activo)

Spring MVC, JSF-PrimeFaces /RichFaces, HTML5, CSS3, JavaScript-jQuery

Gestor portales (Liferay)
Gestor de contenidos (Alfresco)
Aplicaciones híbridas

Control de autenticación y acceso (Spring Security)
UDDI
Web Services
Rest Services
Social SSO
SSO (Cas)

JPA-Hibernate, MyBatis
Motor de búsqueda empresarial (Solr)
ETL (Talend)

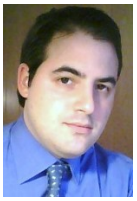
Dirección de Proyectos Informáticos.
Metodologías ágiles
Patrones de diseño
TDD

Tareas programadas (Quartz)
Gestor documental (Alfresco)
Inversión de control (Spring)

BPM (jBPM o Bonita)
Generación de informes (JasperReport)
ESB (Open ESB)

Compartimos nuestro conocimiento en:
www.adictosaltrabajo.com

Para más información visítenos en:
www.autentia.com



DESARROLLADO POR:
Rubén Aguilera Díaz-Heredero

Consultor tecnológico de desarrollo de proyectos informáticos.

Ingeniero en Informática, especialidad en Ingeniería del Software

Puedes encontrarme en Autentia: Ofrecemos servicios de soporte a desarrollo, factoría y formación

Somos expertos en Java/J2EE

Fecha de publicación del tutorial: 2009-02-26

16

Share | Regístrate para votar

Ejemplo básico con Spring MVC Portlet

0. Índice de contenidos.

- 1. Entorno
- 2. Introducción
- 3. Creando el proyecto con Maven
- 4. Configurando las dependencias del proyecto
- 5. Configurando el web.xml
- 6. Configurando el applicationContext.xml
- 7. Configurando el portlet.xml
- 8. Configurando el spring-mvc-prueba-portlet.xml
- 9. Creando la clase controladora
- 10. Creando la vista por defecto (view.jsp)
- 11. Configurando el portlet para hacerlo funcionar en Liferay
- 12. Desplegando el portlet en Liferay
- 13. Conclusiones

1. Entorno

Este tutorial está escrito usando el siguiente entorno:

- Hardware: Portátil Mac Book Pro 17" (2,6 Ghz Intel Core i7, 8 GB DDR3)
- Sistema Operativo: Mac OS X Snow Leopard 10.6.4
- Spring MVC Portlet 3.0.4
- Maven 2.2.1
- Eclipse 3.6 (Helios) con M2Eclipse
- Liferay 6.0.5

2. Introducción

En este tutorial vamos a ver paso a paso cómo crear un portlet muy básico con la ayuda de Spring MVC Portlet, desde su creación con Maven hasta su despliegue en Liferay. Servirá como base para la creación de portlets más complejos utilizando esta tecnología.

3. Creando el proyecto con Maven

En este tipo de proyectos en los que vamos a utilizar librerías de terceros como Spring, se hace especialmente útil la utililización de una herramienta de gestión de dependencias como Maven. Teniendo Maven ya instalado en nuestra máquina lo único que tenemos que hacer es abrir un terminal y situarnos en el directorio donde vayamos a crear el proyecto de tipo web con el siguiente comando:

```
view plain  print  ?
01.  mvn archetype:create -DgroupId=com.autentia -DartifactId=spring-mvc-portlet-prueba -DarchetypeId=maven-archetype-webapp
```

NOTA: para los usuarios de Liferay, existe un arquetipo que crea el proyecto con el ficheros necesarios de despliegue para Liferay. En este caso el comando sería:

```
view plain  print  ?
01.  mvn archetype:create -DgroupId=com.adictos.portlet -DartifactId=maven-portlet -DarchetypeArtifactId=liferay-portlet-archetype -DarchetypeGroupId=com.liferay.maven.archetypes -DarchetypeVersion=6.0.5
```

Podréis encontrar más detalles en este tutorial: http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=mavenliferay

4. Configurando las dependencias del proyecto

Con el fin de que la edición del proyecto sea más cómoda podemos importarlo a un IDE como Eclipse. Para hacer esto, contamos con el plugin M2Eclipse que es la evolución del conocido EclipseIAM.

Para configurar las dependencias editamos el fichero pom.xml del proyecto, añadiendo las siguientes:

```
view plain  print  ?
01.  <dependency>
02.      <groupId>org.springframework</groupId>
03.      <artifactId>spring-webmvc-portlet</artifactId>
04.      <version>3.0.4.RELEASE</version>
05.  </dependency>
06.  <dependency>
07.      <groupId>javax.servlet</groupId>
08.      <artifactId>jstl</artifactId>
09.      <version>1.1.2</version>
10.  </dependency>
11.  <dependency>
12.      <groupId>javax.portlet</groupId>
13.      <artifactId>portlet-api</artifactId>
14.      <version>2.0</version>
15.      <scope>provided</scope>
16.  </dependency>
```

Últimas Noticias

- XIII Charla Autentia - AOS y TDD
- XII Charla Autentia - LiquiBase - Material
- XI Charla Autentia - Mule - Vídeos y Material
- Reflexiones sobre AOS2010
- Comentando el libro: Nunca comas solo de Keith Ferrazzi y Tahl Raz.

Histórico de NOTICIAS

Últimos Tutoriales

- JCaptcha: Análisis Técnico en aplicativos reales
- Zen-coding: una nueva forma de escribir código HTML
- JBoss autenticación basada en certificados cliente
- Rdiff-backup: Herramienta para realizar backups
- Cómo crear un efecto reflejo con Adobe Photoshop

Últimos Tutoriales del Autor

- Trabajando con los Web Services de Liferay
- Liferay IDE
- CAS: Validador personalizado
- CAS: Personalización de la interfaz
- Introducción a CAS

Síguenos a través de:

Últimas ofertas de empleo

2010-10-11
Comercial - Ventas - SEVILLA.
2010-08-30
Otras - Electricidad - BARCELONA.
2010-08-24
Otras Sin catalogar - LUGO.
2010-06-25
T. Información - Analista / Programador - BARCELONA.

La librería portlet-api tiene el scope a provided porque es necesaria para compilar el proyecto, pero debe ser proporcionada por el gestor de portales donde vayamos a desplegar nuestro proyecto. Utilizamos una versión a partir de la 3.0.x para poder hacer uso del estándar JSR-286 o más conocido como Portlet 2.0.

5. Configurando el web.xml

Ahora tenemos que editar el fichero web.xml de nuestro proyecto para intregar nuestro desarrollo con Spring MVC Portlet, de tal modo que tenemos que añadir los siguientes elementos:

```
view plain  print  ?
01. <web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
02.     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
03.     xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
04.
05.     <display-name>spring-mvc-prueba-portlet</display-name>
06.
07.     <!-- Define la localizaciÃ³n de los ficheros de configuraciÃ³n de Spring -->
08.     <context-param>
09.         <param-name>contextConfigLocation</param-name>
10.         <param-value>/WEB-INF/context/applicationContext.xml</param-value>
11.     </context-param>
12.
13.     <!-- Listener de Contexto -->
14.     <listener>
15.         <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
16.     </listener>
17.
18.     <!-- Servlet de Spring MVC Portlet -->
19.     <servlet>
20.         <servlet-name>ViewRendererServlet</servlet-name>
21.         <servlet-class>org.springframework.web.servlet.ViewRendererServlet</servlet-class>
22.         <load-on-startup>1</load-on-startup>
23.     </servlet>
24.
25.     <servlet-mapping>
26.         <servlet-name>ViewRendererServlet</servlet-name>
27.         <url-pattern>/WEB-INF/servlet/view</url-pattern>
28.     </servlet-mapping>
29.
30. </web-app>
```

En primer lugar definimos el parámetro contextConfigLocation para establecer donde vamos a almacenar el fichero de configuración de Spring de nuestro proyecto, en este caso será dentro de la carpeta WEB-INF/context con el nombre applicationContext.xml. Luego tenemos que crear el listener y el servlet necesarios para trabajar con Spring MVC.

6. Configurando el applicationContext.xml

Si no lo hemos hecho, ya este es un buen momento para instalar el plugin Spring IDE en nuestro Eclipse, ya que nos va a facilitar la tediosa tarea de tener que generar los encabezados de los ficheros de configuración de Spring.

Creamos el fichero WEB-INF/applicationContext.xml donde vamos a determinar el comportamiento de Spring dentro de nuestro proyecto. Para ello añadimos el siguiente contenido:

```
view plain  print  ?
01. <beans xmlns="http://www.springframework.org/schema/beans"
02.     xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:context="http://www.springframework.org/schema/context"
03.     xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-
beans-3.0.xsd
04.         http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-
3.0.xsd" default-autowire="byName">
05.
06.     <!-- Para que Spring sepa que vamos a usar anotaciones -->
07.     <context:annotation-config />
08.
09.     <!-- Desde donde tiene que escanear -->
10.     <context:component-scan base-package="com.autentia.springmvcpruebaportlet" />
11.
12. </beans>
```

Estamos determinando que vamos a utilizar anotaciones y que Spring tiene que comenzar a escanear cuando tenga que localizar un bean, a partir de la ruta del paquete que establezcamos en el valor base-package.

7. Configurando el portlet.xml

La gente que ya está acostumbrada al desarrollo de portlet ya sabe que en este fichero se definen las características de nuestro portlet: cuál es su ID, que modos de trabajo va a tener, cuál va a ser el fichero de internacionalización, ... y sobre todo que clase se va a encargar de implementar la lógica del portlet.

Pues bien, en este caso, todo va a ser igual, sólo que la clase encargada de implementar el portlet siempre va a ser la misma org.springframework.web.portlet.DispatcherPortlet que requiere la inicialización del parámetro contextConfigurationLocation con la ruta del fichero de Spring que se va a encargar de manejar nuestro portlet. Este es un posible código de ejemplo:

```
view plain  print  ?
01. <portlet-app
02.     version="2.0"
03.     xmlns="http://java.sun.com/xml/ns/portlet/portlet-app_2_0.xsd"
04.     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
05.     xsi:schemaLocation="http://java.sun.com/xml/ns/portlet/portlet-
app_2_0.xsd http://java.sun.com/xml/ns/portlet/portlet-app_2_0.xsd">
06.     <portlet>
07.         <portlet-name>spring-mvc-prueba-portlet</portlet-name>
08.         <display-name>spring-mvc-prueba-portlet</display-name>
09.         <portlet-class>org.springframework.web.portlet.DispatcherPortlet</portlet-class>
10.         <init-param>
11.             <name>contextConfigLocation</name>
12.             <value>/WEB-INF/context/portlet/spring-mvc-prueba-portlet.xml</value>
13.         </init-param>
14.         <expiration-cache>0</expiration-cache>
15.         <supports>
16.             <mime-type>text/html</mime-type>
17.         </supports>
18.         <portlet-info>
19.             <title>spring-mvc-prueba-portlet</title>
20.             <short-title>spring-mvc-prueba-portlet</short-title>
21.             <keywords>spring-mvc-prueba-portlet</keywords>
22.         </portlet-info>
23.     </portlet>
24. </portlet-app>
```

8. Configurando el spring-mvc-prueba-portlet.xml

Ahora tenemos que crear el fichero de configuración de Spring para nuestro portlet. La información más importante de este fichero es qué controlador se va a encargar de manejar las peticiones y cómo se va a mostrar la vista. En este caso vamos a tener un controlador para todas las peticiones que se realicen en el estado de la vista del portlet y vamos a mostrar el contenido a través de páginas JSP.

Estas definiciones se corresponden con el siguiente contenido:

```
view plain  print  ?
01. <beans xmlns="http://www.springframework.org/schema/beans"
02.     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
03.     xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-
    beans-3.0.xsd">
04.
05.     <bean id="portletModeHandlerMapping"
06.         class="org.springframework.web.portlet.handler.PortletModeHandlerMapping">
07.         <property name="portletModeMap">
08.             <map>
09.                 <entry key="view" value-ref="springMVCPruebaPortletViewController" />
10.             </map>
11.         </property>
12.     </bean>
13.
14.     <!-- Default View Resolver -->
15.     <bean id="viewResolver"
16.         class="org.springframework.web.servlet.view.InternalResourceViewResolver">
17.         <property name="cache" value="false" />
18.         <property name="viewClass"
19.             value="org.springframework.web.servlet.view.JstlView" />
20.         <property name="prefix" value="/WEB-INF/jsp/spring-mvc-prueba-portlet/" />
21.         <property name="suffix" value=".jsp" />
22.     </bean>
23.
24.
25. </beans>
```

9. Creando la clase controladora

Para la creación del controlador utilizamos la convención frente a la configuración por lo que vamos a crear una clase controladora del mismo nombre (salvo empezando por mayúsculas) que el id del bean referenciado en la entrada key="view".

Por lo tanto vamos a crear la clase SpringMVCPruebaPortletViewController dentro de un paquete que este en el alcance del escaneado de Spring, por ejemplo, com.autentia.springmvcpruebaportlet.controllers con el siguiente contenido:

```
view plain  print  ?
01. package com.autentia.springmvcpruebaportlet.controllers;
02.
03. import org.springframework.stereotype.Controller;
04. import org.springframework.web.bind.annotation.RequestMapping;
05.
06. @Controller
07. @RequestMapping("VIEW")
08. public class SpringMVCPruebaPortletViewController {
09.
10.     public SpringMVCPruebaPortletViewController() {
11.     }
12.
13.     @RequestMapping
14.     protected String defaultView() {
15.         return "view";
16.     }
17.
18. }
```

Este es el contenido mínimo que tiene que tener nuestro controlador. Con la anotación @Controller estamos definiendo que es un controlador manejado por Spring y por tanto implícitamente estamos creando el bean asociado dentro de nuestro fichero de configuración. Con la anotación @RequestMapping("VIEW") estamos estableciendo que este controlador se va a encargar de la vista del portlet y con la anotación @RequestMapping sin ningún parámetro adicional estamos estableciendo que ese método es el que se va a ejecutar por defecto.

En Spring MVC Portlet, tenemos que dintiguir dos tipos de métodos: lo que devuelven String y los que no devuelven nada (void). Los del primer caso se corresponden con métodos Render donde el String que devuelven le indica a Spring que tiene que buscar un .jsp dentro de la carpeta establecida con ese nombre, para mostrarlo en la vista del portlet. Los del segundo caso se corresponden con métodos Action que reciben las peticiones del usuarios, ejecutan la lógica asociada y establecen que método Render va a ser el siguiente a ejecutar, en caso de no establecer ninguno, se ejecutaría el de por defecto, anotado con @RequesMapping sin parámetros adicionales.

10. Creando la vista por defecto (view.jsp)

Como se puede ver en el código anterior estamos definiendo que la vista por defecto muestre el fichero view.jsp. Por tanto tenemos que crear este fichero dentro de la ruta configurado en el ViewResolver de nuestro portlet con el siguiente contenido, por ejemplo:

```
view plain  print  ?
01. <%@ taglib uri="http://java.sun.com/portlet_2_0" prefix="portlet" %>
02.
03. <portlet:defineObjects />
04.
05. This is the <b>spring-mvc-prueba-portlet</b>.
```

11. Configurando el portlet para hacerlo funcionar en Liferay

Hasta este punto nuestro desarrollo es perfectamente portable a cualquier gestor de portales que queramos utilizar (a no ser que se haya utilizado el arquetipo de Liferay definido en el punto 3, en cuyo caso podemos obviar este punto).

Para hacer funcionar nuestro desarrollo en Liferay tenemos que crear los siguientes ficheros de configuración dentro de la carpeta WEB-INF de nuestro proyecto:

liferay-display.xml: que define la categoría del árbol de portlets donde vamos a encontrarlo a la hora de instanciarlo en una página. Este es un contenido de ejemplo:

```
view plain  print  ?
01. <!DOCTYPE display PUBLIC "-//Liferay//DTD Display 6.0.0//EN" "http://www.liferay.com/dtd/liferay-
    display_6_0_0.dtd">
02.
03. <display>
04.     <category name="category.sample">
05.         <portlet id="spring-mvc-prueba-portlet" />
06.     </category>
07. </display>
```

liferay-portlet.xml que le da información extra de características sólo aplicables a Liferay como si va a llevar un icono, si va a ser instanciable, que ficheros CSS y Javascript tiene que cargar, ...

```
view plain  print  ?
01. <!DOCTYPE liferay-portlet-app PUBLIC "-
    //Liferay//DTD Portlet Application 6.0.0//EN" "http://www.liferay.com/dtd/liferay-portlet-app_6_0_0.dtd">
02.
03. <liferay-portlet-app>
04.     <portlet>
05.         <portlet-name>spring-mvc-prueba-portlet</portlet-name>
06.         <icon>/icon.png</icon>
07.         <instanceable>true</instanceable>
08.         <header-portlet-css>/css/main.css</header-portlet-css>
```

```
        <footer-portlet-javascript>/js/main.js</footer-portlet-javascript>
    </portlet>
</liferay-portlet-app>
```

liferay-plugin-package.properties: que determina información del tipo autoría del plugin, versión, compatibilidad con otras versiones, etc ...

```
view plain  print  ?
name=spring-mvc-prueba-portlet
module-group-id=liferay
module-incremental-version=1
tags=
short-description=
change-log=
page-url=http://www.liferay.com
author=Liferay, Inc.
licenses=LGPL
```

12. Desplegando el portlet en Liferay

Para probar el resultado en Liferay basta con empaquetar nuestro desarrollo a través del plugin de Eclipse o a través del terminal, ejecutando dentro de la carpeta del proyecto el comando:

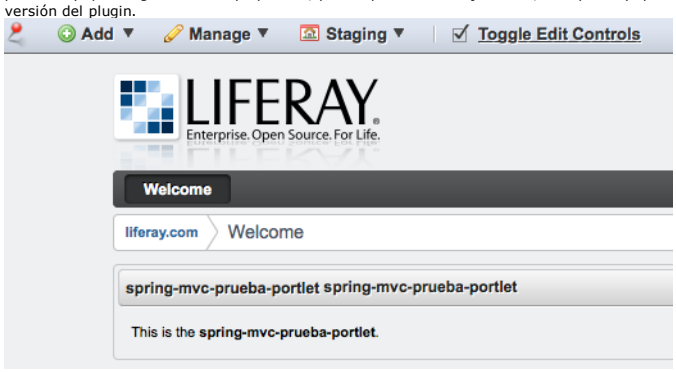
```
view plain  print  ?
mvn clean package
```

Y ahora trasladar el .war resultante a la carpeta deploy de nuestro Liferay. En caso de que esté en ejecución el despliegue se realizará en caliente y ya podremos instanciar nuestro con Spring MVC dentro de la página que queramos.

Los que estén utilizando el arquetipo de Maven para Liferay, se pueden ahorrar este paso ejecutando el siguiente comando por el terminal:

```
view plain  print  ?
mvn liferay:deploy
```

Pero es muy importante indicar que esto sólo hace el traslado del .war a la carpeta que le indicamos en la propiedad <liferay.auto.deploy.dir> del pom.xml y que no generá el empaquetado, por lo que antes de ejecutarlo, siempre hay que hacer, un empaquetado para que se despliegue la última versión del plugin.



13. Conclusiones

Cómo véis no es díficil crear un portlet con Spring MVC y las ventajas que nos reporta van a ser muchas. Por ejemplo, nos facilita la navegación cuando nuestro portlet tiene muchas pantallas, nos permite hacer pruebas unitarias de nuestros controladores a través de Mocks, implementación de validadores y buena forma de desacoplar nuestros desarrollos del gestor de portales. Pero esto lo iremos viendo en sucesivos tutoriales, donde crearemos un portlet para la gestión CRUD de una entidad utilizando Spring MVC Portlet e iremos viendo otras anotaciones fundamentales y técnicas necesarias.

Saludos.

COMENTARIOS