LIFERAY

Enterprise. Open Source. For Life.

Implementaciones de Portlets

Contenidos

1. Liferay MVC
2. Struts 2
3. Spring MVC
4. JSF 2
5. Contexto de Spring en Liferay

PORTLETS CON LIFERAY MVC

Liferay MVC

- Esta implementación de portlet, es similar a **GenericPortlet.**
- Ofrece métodos para los distintos modos de renderizado soportados por Liferay
  - doView
  - doEdit
  - doPrint
  - doConfig
  - doAbout
  - …

Liferay MVC

- Aporta la capacidad de traducir el nombre de una acción enviada al portlet, a un método que se llama igual que la acción.
- Es decir, dado el **ActionURL**, tal que

```
<portlet:actionURL name="addProduct"/>
```

- Se ejecutará el método del Portlet

```
public void addProduct(ActionRequest request, ActionResponse response)
                                                        throws Exception {}
```

Liferay MVC

- Permite definir en el **portlet.xml** a través de **init-param**, las rutas de las vistas a emplear en los distintos modos de renderización

```
<init-param>
		<name>view-template</name>
		<value>/html/portletcompletoliferaymvc/view.jsp</value>
</init-param>
<init-param>
		<name>preview-template</name>
		<value>/html/portletcompletoliferaymvc/preview.jsp</value>
</init-param>
<init-param>
		<name>print-template</name>
		<value>/html/portletcompletoliferaymvc/print.jsp</value>
</init-param>
```

PORTLETS CON STRUTS 2

Struts 2

- Se necesitan las dependencias del proyecto struts2-portlet

```
<dependency>
        <groupId>org.apache.struts</groupId>
        <artifactId>struts2-portlet-plugin</artifactId>
        <version>2.3.20</version>
</dependency>
<dependency>
        <groupId>javassist</groupId>
        <artifactId>javassist</artifactId>
        <version>3.7.1.GA</version>
</dependency>
```

Struts 2

- En el fichero **portlet.xml**, incluir un portlet, cuya clase sea la implementación de portlet que nos proporciona este API.

```
<portlet-class>
        org.apache.struts2.portlet.dispatcher.Jsr286Dispatcher
</portlet-class>
```

- También hay que desactivar el namespace en los formularios, en el fichero **liferay-portlet.xml**

```
<requires-namespaced-parameters>false</requires-namespaced-parameters>
```

Struts 2

- Se han de configurar **init-parameter**, que hacen referencia a los **namespace** de Struts2 que llevaran a cabo el procesamiento de las peticiones de Render y de Action.
  - **viewNamespace.**
  - **defaultViewAction.**
  - **editNamespace**.
  - **defaultEditAction**.
  - **helpNamespace**.
  - **defaultHelpAction**.

Struts 2

- Un posible ejemplo de los **init-param**

```
<init-param>
        <name>viewNamespace</name>
        <value>/view</value>
</init-param>
<init-param>
        <name>defaultViewAction</name>
        <value>index</value>
</init-param>
<init-param>
        <name>editNamespace</name>
        <value>/edit</value>
</init-param>
<init-param>
        <name>defaultEditAction</name>
        <value>index!input</value>
</init-param>
```

Struts 2

- **viewNamespace** -> Namespace de Struts2, que define los Action en modo View.
- **defaultViewAction** -> Action de Struts2 dentro del **viewNamespace**, a ejecutar al renderizar el portlet en modo View.

Struts 2

- Crear las clases de los Action, haciéndolas heredar de

```
org.apache.struts2.dispatcher.DefaultActionSupport
```

- Y sobrescribiendo el método "**execute**", o el indicado en el **struts.xml**.
- Definición del fichero **struts.xml** en la raiz del classpath del proyecto.
- Los paquetes definidos deberán extender de "**struts-portlet-default**".

Struts 2

- Un ejemplo del documento

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC …..”http://struts.apache.org/dtds/struts-2.3.dtd">
<struts>
        <package name=“view" extends="struts-portlet-default"
                                                namespace="/view">
                <action name="index" class="com.ejemplo.HelloAction">
                        <result>/WEB-INF/jsp/view/index.jsp</result>
                </action>
                <action name="index“ class="com.ejemplo.action.OtraAction"
                                                        method="execute">
                        <result  name="input">/html/edit/index.jsp</result>
                </package>
</struts>
```

Struts 2

- Desde la clase del Action, se puede tener acceso a
  - **PortletPreferences**
  - **PortletConfig**
  - **PortletContext**
  - **PortletRequest**
  - **PortletResponse**

Struts 2

- Para ello, hay que implementar las interface “Aware” respectivas.
  - org.apache.struts2.portlet.interceptor.PortletPreferencesAware
  - org.apache.struts2.portlet.interceptor.PortletConfigAware
  - org.apache.struts2.portlet.interceptor.PortletContextAware
  - org.apache.struts2.portlet.interceptor.PortletRequestAware
  - org.apache.struts2.portlet.interceptor.PortletResponseAware

Struts 2

- Un ejemplo de clase de Action

```
public class EditHelloWorldAction extends DefaultActionSupport
                                        implements PortletPreferencesAware {
        private PortletPreferences portletPreferences;
        private String sufijo;
        @Override
        public String execute() throws Exception {}
        @Override
        public void setPortletPreferences(PortletPreferences arg0) {}
        //Getter y Setter de nombre
}
```

Struts 2

- Por ultimo quedaría definir las JSP que representan la salida de los portlets, e ellas, emplearemos las etiquetas de struts

```
<%@ taglib prefix="s" uri="/struts-tags" %>

<s:form action="index">
        <s:textfield name="prefijo" label="Prefijo"/>
        <s:submit value="Enviar"/>
</s:form>

<s:property value="saludo" />

<s:url action='index'/>
<s:url action='index!input'/>
```

Struts 2

- La forma en que se intercambia la información desde el Action y las JSP, es a través de atributos de clase encapsulados, de la misma forma que una aplicación puramente Struts 2.
- Tanto para la entrada de datos al Action, como para la salida.

PORTLETS CON SPRING MVC

Spring MVC

- El API de Spring para Portlets es **spring-portlet-MVC**.
- Para trabajar con **Spring MVC** en **Liferay**, no es necesario incluir las librerías con **Maven**, ya que **Liferay** las emplea, únicamente, hay que hacerles referencia.

Spring MVC

- Para ello, editar el fichero

liferay-plugin-package.properties

- Y en la sección **Portal dependency jars**, añadir los siguientes jar.
  - org.springframework.asm-3.0.0.RELEASE
  - org.springframework.beans-3.0.0.RELEASE
  - org.springframework.context-3.0.0.RELEASE
  - org.springframework.core-3.0.0.RELEASE
  - org.springframework.expression-3.0.0.RELEASE
  - org.springframework.web-3.0.0.RELEASE
  - org.springframework.web.portlet-3.0.0.RELEASE
  - org.springframework.web.servlet-3.0.0.RELEASE

Spring MVC

- Esto se ha de traducir en el **properties** de la siguiente forma:

```
portal-dependency-jars=\
   spring-asm.jar,\
   spring-beans.jar,\
   spring-context.jar,\
   spring-core.jar,\
   spring-expression.jar,\
   spring-web-portlet.jar,\
   spring-web-servlet.jar,\
   spring-web.jar
```

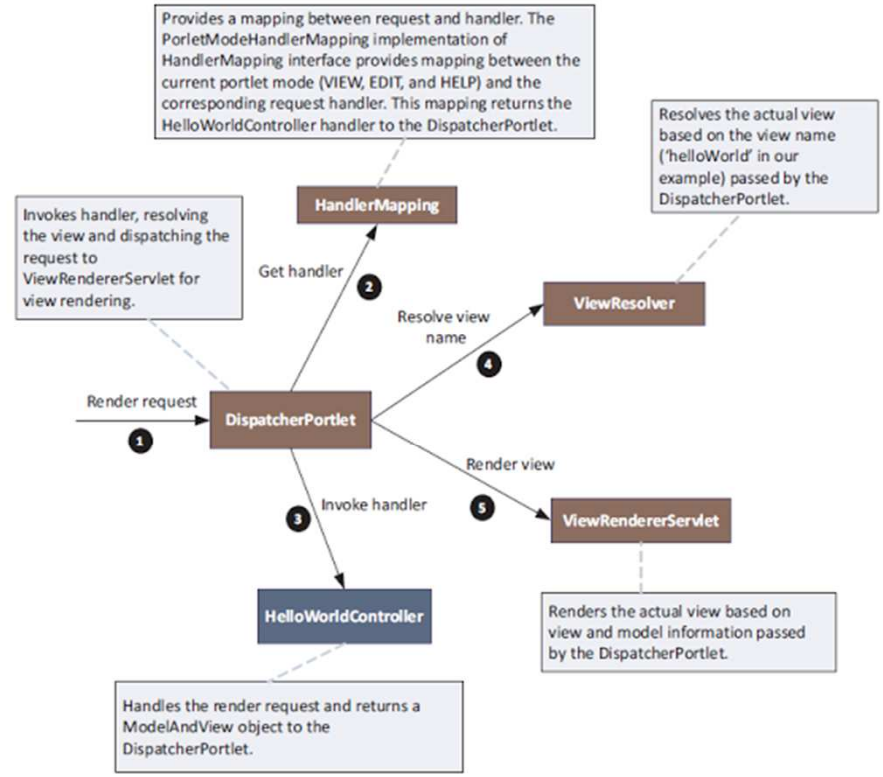
Spring MVC

- Para hacerlo con Maven, se ha de añadir la dependencia

```
<dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-webmvc-portlet</artifactId>
        <version>3.0.0.RELEASE</version>
        <scope>provided</scope>
</dependency>
```

Spring MVC

- Participantes



Request processing in Spring Portlet MVC. The `DispatcherPortlet` acts as the front controller for each portlet and finds the handler mapped to the request using `HandlerMapping`. The result of executing a handler, `ModelAndView`, is used to resolve the actual view to be rendered, and the request is dispatched to `ViewRendererServlet` for rendering.

Spring MVC

- Se ha de definir en el **web.xml**, el **RendererServlet** de Spring.

```
<servlet>
        <servlet-name>ViewRendererServlet</servlet-name>
        <servlet-class>
                org.springframework.web.servlet.ViewRendererServlet
        </servlet-class>
        <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
        <servlet-name>ViewRendererServlet</servlet-name>
        <url-pattern>/WEB-INF/servlet/view</url-pattern>
</servlet-mapping>
```

Spring MVC

- En el fichero **portlet.xml**, se han de definir los portlets como

```
<portlet-class>org.springframework.web.portlet.DispatcherPortlet</portlet-class>
```

- Siendo el fichero de contexto de spring

```
<NombreDeMiPortlet>-portlet.xml
```

- También se puede definir como se llamara el fichero de contexto de spring con el **init-param**

```
<init-param>
        <name>contextConfigLocation</name>
        <value>/WEB-INF/HolaMundoSpringMVC-portlet.xml</value>
 </init-param>
```

Spring MVC

- Se han de declarar los Bean de Spring que realizaran las labores de controlador, para ello se puede hacer en
  - El fichero de contexto de spring en **WEB-INF**

```
<bean id="MiPortletSpringMVCController“
                                    class="com.ejemplo.MiPortletSpringMVCController"/>
```

  - O con anotaciones, para lo que el en fichero de contexto se ha de añadir.

```
<context:component-scan base-package="com.mvc"/>
```

  - Y en las clases anotar con

```
@Controller
```

Spring MVC

- Las clases de Controlador, que no sean anotadas, deberán heredar de la jerarquía de

```
org.springframework.web.portlet.mvc.Controller
```

- Implementando los métodos
  - **handleActionRequest**
  - **handleRenderRequest**

Spring MVC

- Las anotadas, deberán anotar la clase con
  - **@Controller**
  - **@RequestMapping**("VIEW"): Define el RenderMode que procesara el Controller.
- Y cada uno de los métodos con
  - **@RenderMapping**: Define el método que procesara la fase de **Render**. Es parametrizable.
  - **@ActionMapping**: Define el método de procesará la fase de **Action**. Es parametrizable.

Spring MVC

- Y cada uno de los métodos con
  - @**ResourceMapping**: Define el método que procesara la fase de **Resource**. Define un Texto que será igual al ID de la etiqueta < <**portlet:resourceURL**>.
  - @**EventMapping**: Define el método que procesara la fase de **Event**. Define un parámetro **value**, que ha de definir el tipo de evento que trata.

Spring MVC

- Para que una petición de
  - **Render**
  - **Action**
  - **Resource**
  - **Event**
- Llegue al **Controller** de Spring, se ha de definir una estrategia de Manejo, definiendo un Bean que herede la jerarquía de

```
org.springframework.web.portlet.HandlerMapping
```

Spring MVC

- Existen disponibles
  - **DefaultAnnotationHandlerMapping**: Permite interpretar las anotaciones **@RequestMapping, @RenderMapping**, **@ActionMapping**, **@ResourceMapping** y **@EventMapping**.
  - **PortletModeHandlerMapping**: Permite configurar la relación entre los modos de renderización (view, edit, help) con los controladores con una propiedad del Bean.

Spring MVC

- Existen disponibles
  - **PortletModeParameterHandlerMapping**: Permite definir la relación entre los modos de renderización y los controladores, con un fichero de properties.
  - **ParameterHandlerMapping**. Permite mapear el controller a ejecutar a través de un parámetro de la URL, fuera del API de Portlets y RenderMode.

Spring MVC

- El mas habitual es

```
org.springframework.web.portlet.mvc.annotation.DefaultAnnotationHandler
Mapping
```

- Pudiéndose definir

```
<bean
class="org.springframework.web.portlet.mvc.annotation.DefaultAnnotation
HandlerMapping" />
```

- No olvidar en este caso las etiquetas de anotaciones básicas

```
<context:annotation-config />

<context:component-scan base-package="com.ejemplo.controller" />
```

Spring MVC

- Una configuración del **PortletModeHandlerMapping**, podría ser

```
<bean id="portletModeHandlerMapping"
class="org.springframework.web.portlet.handler.PortletModeHandlerMapping">
        <property name="portletModeMap">
                <map>
                        <entry key="view" value-ref="viewController"/>
                        <entry key="edit" value-ref="editController"/>
                        <entry key="help" value-ref="helpController"/>
                </map>
        </property>
</bean>

<bean id="viewController" class="ViewController"/>
<bean id="editController" class="EditController"/>
<bean id="helpController" class="HelpController"/>
```

Spring MVC

- También hay que definir en el contexto de Spring un **ViewResolver**, que se encargue de resolver el resultado de los **Controller** para seleccionar la Vista adecuada.
- Se ha de definir un Bean que herede la jerarquía de

```
org.springframework.web.servlet.ViewResolver
```

Spring MVC

- Existen varios en el API
  - ResourceBoundleViewResolver
  - FreeMakerViewResolver
  - VelocityViewResolver
  - InternalResorceViewResolver
  - JasperReportsViewResolver
  - TilesViewResolver
  - XsltViewResolver
  - BeanNameViewResolver
  - ContentNegottiationViewResolver

Spring MVC

- El mas habitual es

```
org.springframework.web.servlet.view.InternalResourceViewResolver
```

- Pudiéndose definir

```
<bean id="jspViewResolver"
class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="viewClass"
        value="org.springframework.web.servlet.view.InternalResourceView" />
        <property name="prefix" value="/html/" />
        <property name="suffix" value=".jsp" />
        <property name="order" value="1" />
</bean>
```

PORTLETS CON JAVA SERVER FACES 2

JSF 2

- Hay que incluir las librerías de JSF, así como el **portlet-bridge**.
- Con maven

```
<dependency>
		<groupId>com.liferay.faces</groupId>
		<artifactId>liferay-faces-bridge-impl</artifactId>
		<version>3.1.3-ga4</version>
</dependency>
<dependency>
		<groupId>com.sun.faces</groupId>
		<artifactId>jsf-api</artifactId>
		<version>2.1.7</version>
</dependency>
<dependency>
		<groupId>com.sun.faces</groupId>
		<artifactId>jsf-impl</artifactId>
		<version>2.1.7</version>
</dependency>
```

JSF 2

- En el **web.xml**, hay que definir el **FacesServlet**

```
<servlet>
        <servlet-name>Faces Servlet</servlet-name>
        <servlet-class>
                javax.faces.webapp.FacesServlet
        </servlet-class>
        <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
        <servlet-name>Faces Servlet</servlet-name>
        <url-pattern>*.xhtml</url-pattern>
</servlet-mapping>
```

JSF 2

- En el **portlet.xml**, se ha de definir los portlets, que serán del tipo

```
<portlet-class>javax.portlet.faces.GenericFacesPortlet</portlet-class>
```

- Este portlet, exige que se le pasen unos parámetros iniciales, que indican los xhtml de las fases de render.
  - **javax.portlet.faces.defaultViewId.view**
  - **javax.portlet.faces.defaultViewId.edit**
  - **javax.portlet.faces.defaultViewId.help**

JSF 2

- Se han de definir los **Managed Bean** de JSF
- Para ello se puede emplear
  - El fichero **faces-config.xml**.
  - Anotaciones @**ManagedBean**
- Donde las propiedades de los bean, serán la vía de intercambio entre los xhtml (vistas) y los métodos de los managed bean (acciones).

JSF 2

- Con **faces-config.xml**.

```
<managed-bean>
        <managed-bean-name>
                editModeManagedBean
        </managed-bean-name>
        <managed-bean-class>
                com.ejemplo.MiBean
        </managed-bean-class>
        <managed-bean-scope>
                request
        </managed-bean-scope>
</managed-bean>
```

JSF 2

- Con anotaciones

```
@ManagedBean(name="miBean")
@RequestScoped
public class MiBean {}
```

JSF 2

- Los **ManagedBean**, han de tener el constructor por defecto.
- Las propiedades de los **ManagedBean**, han de estar encapsuladas.
- Los métodos de acción, han de ser públicos, sin parámetros, que retornen un String, que representa la vista a mostrar como resultado de la operación.

JSF 2

- En las vistas se pueden emplear los tld de faces.

```
<f:view
        xmlns="http://www.w3.org/1999/xhtml"
        xmlns:c="http://java.sun.com/jsp/jstl/core"
        xmlns:f="http://java.sun.com/jsf/core"
        xmlns:h="http://java.sun.com/jsf/html"
        xmlns:ui="http://java.sun.com/jsf/facelets">
```

JSF 2

- Para acceder a los objetos **PortletRequest** y **PortletResponse** desde un **ManagedBean**, se emplea **ExternalContext**

```
FacesContext facesContext = FacesContext.getCurrentInstance();

ExternalContext externalContext = facesContext.getExternalContext();

PortletRequest portletRequest = (PortletRequest) externalContext.getRequest();

PortletResponse portletResponse = (PortletResponse)
                                                    externalContext.getResponse();
```

CONTEXTO DE SPRING

Contexto de Spring

- En cualquier tipo de proyecto, se puede crear un contexto de Spring, para que sea Spring quien maneje los Bean de Negocio y Persistencia.
- Para ellos se ha de añadir Spring al Proyecto como se vio en el apartado de **Portlets con Spring MVC**.

Contexto de Spring

- Una vez se tienen las dependencias, habrá que definir un **Listener** en el contexto Web, que cree el contexto de Spring

```
<listener>
        <listener-class>
        org.springframework.web.context.ContextLoaderListener
        </listener-class>
</listener>
<context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/applicationContext.xml</param-value>
</context-param>
```

Contexto de Spring

- Una vez definido el contexto de Spring, se puede acceder a el de forma programática desde cualquier clase con

```
ApplicationContext context = PortletApplicationContextUtils
                                        .getWebApplicationContext(portletContext);

Negocio negocio = (Negocio) context.getBean("negocio");
```