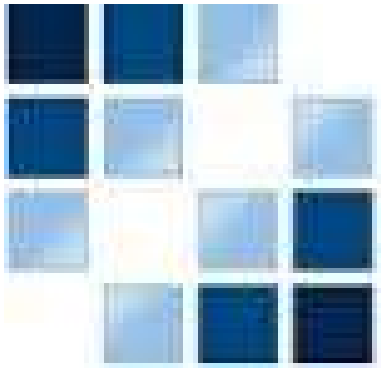
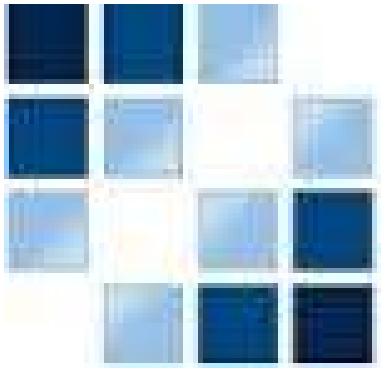


LIFERAY

Enterprise. Open Source. For Life.

Portlets

Contenidos



LIFERAY

Enterprise. Open Source. For Life.

INTRODUCCIÓN

Portal

- Aplicación web, que puede trabajar con Portlets.
- Define sus Páginas con Portlets.
- Define un Contenedor de Portlets.

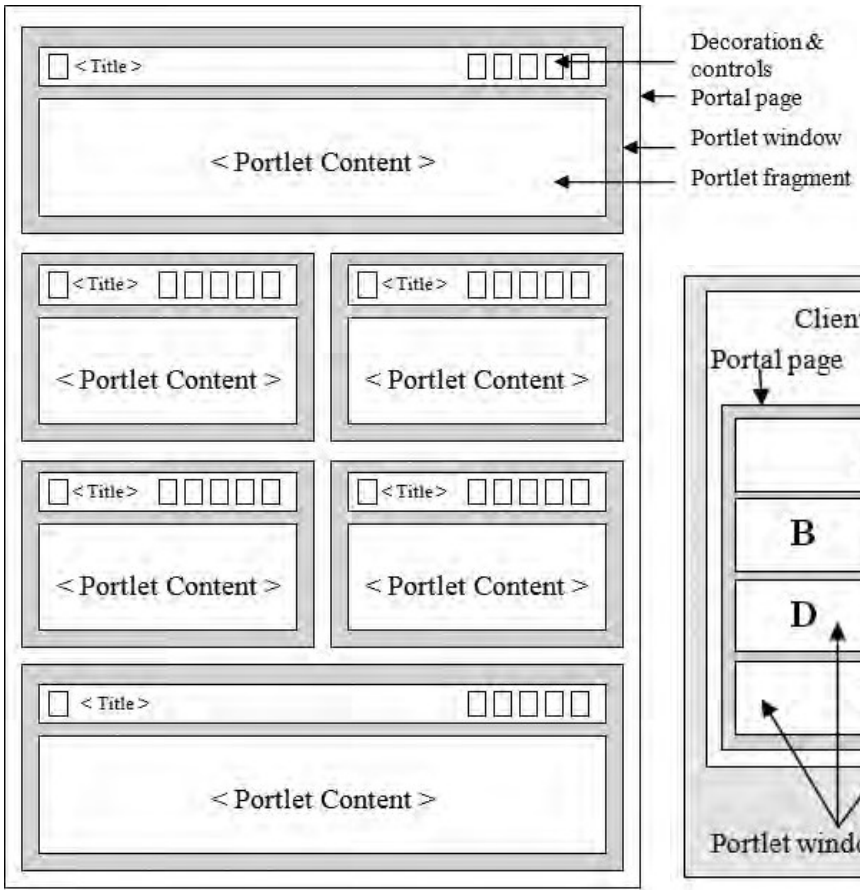
Contenedor de Portlets

- Entorno de ejecución para los Portlets
- Extensión del contenedor de Servlets
- Maneja el ciclo de vida y almacena las preferencias

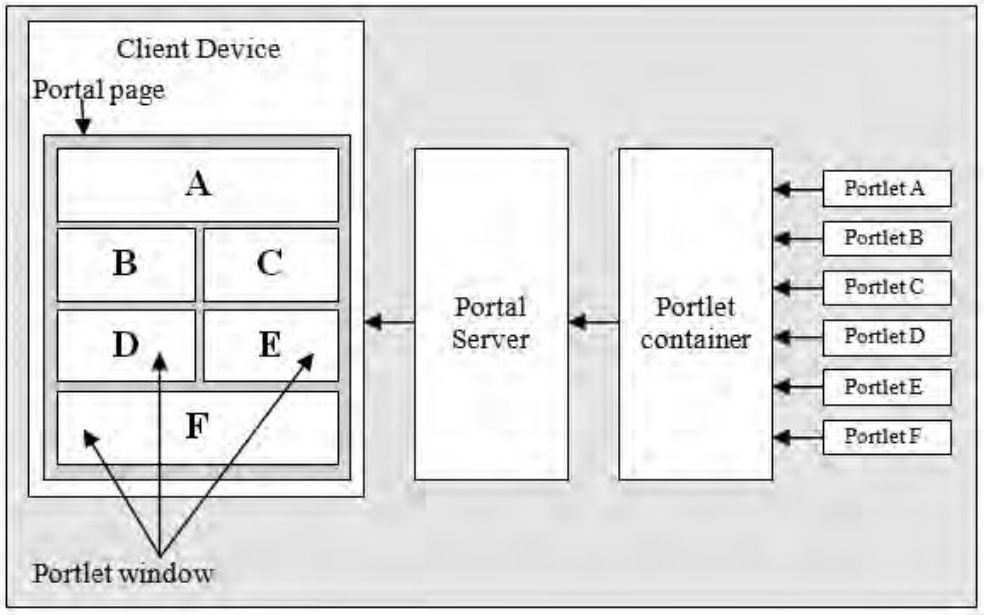
Portlet

- Mini-aplicación que proporciona contenido
- Componente gráfico usado en portales
- Configurados en WEB-INF\portlet.xml
- Generan código según
  - Su modo de renderizado (VIEW, EDIT, HELP)
  - Y su estado (Normal, Maximizado, Minimizado)

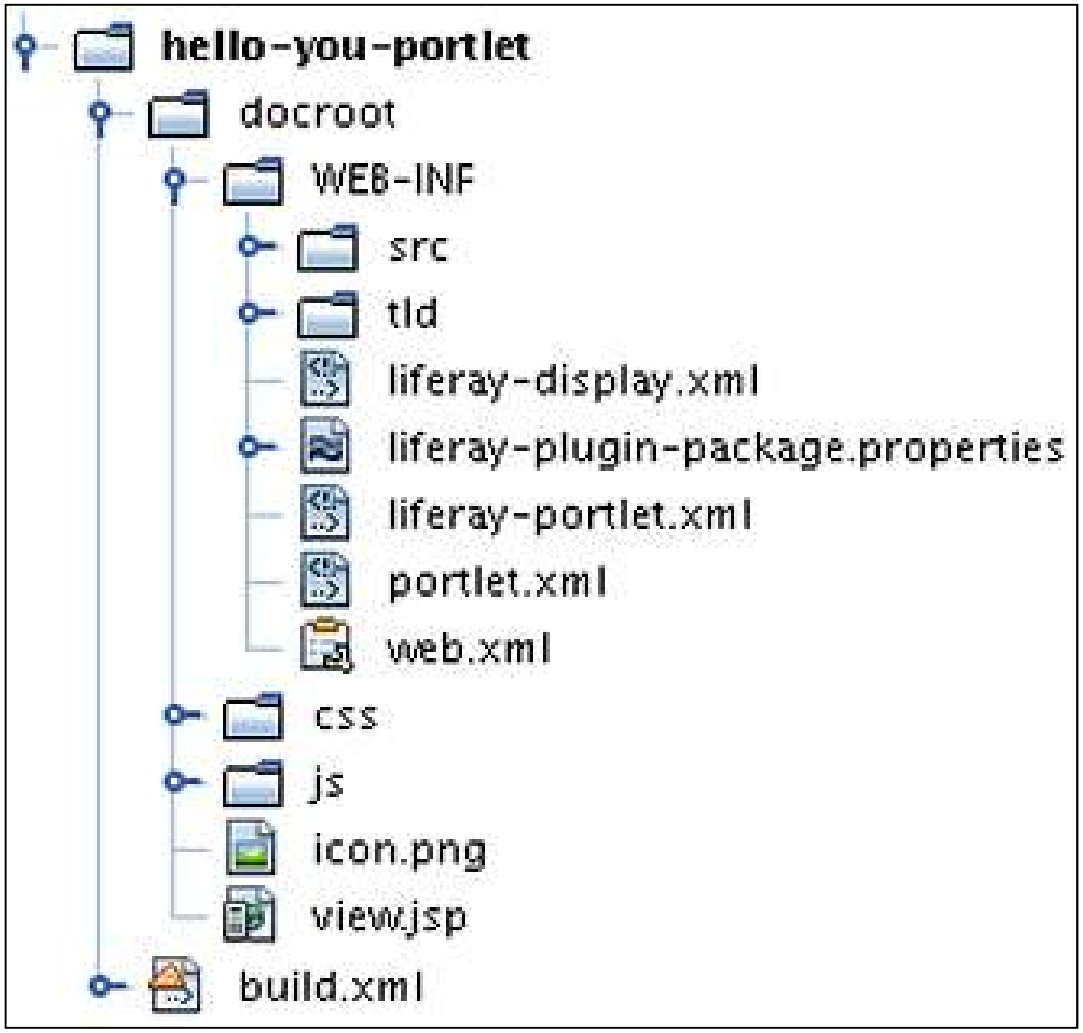
Portlet

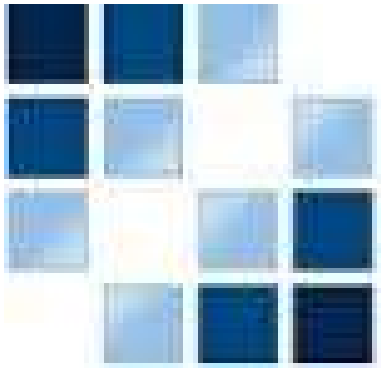


- Portales y Portlets Liferay



Anatomía de un proyecto de Portlet





LIFERAY

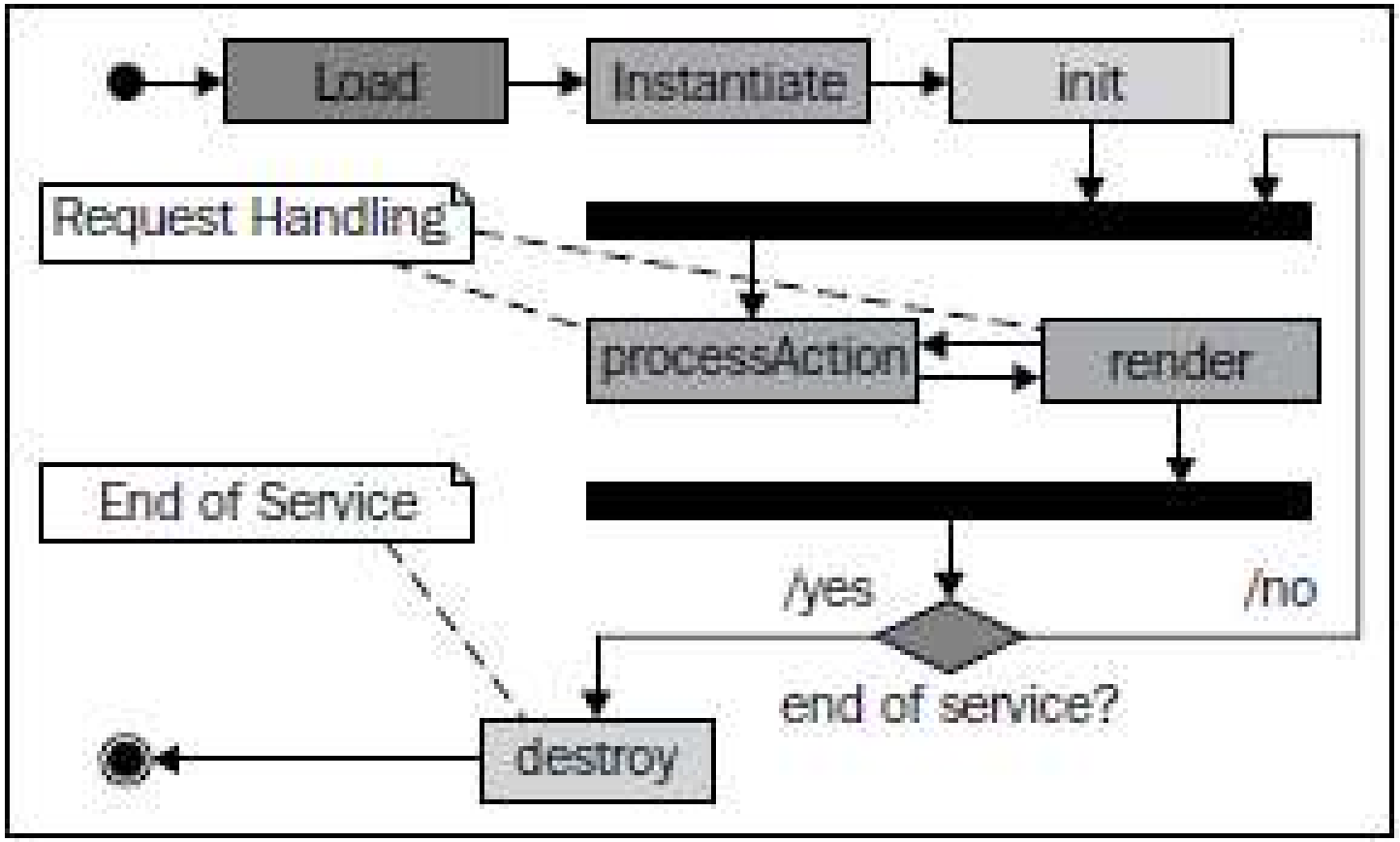
Enterprise. Open Source. For Life.

CICLO DE VIDA

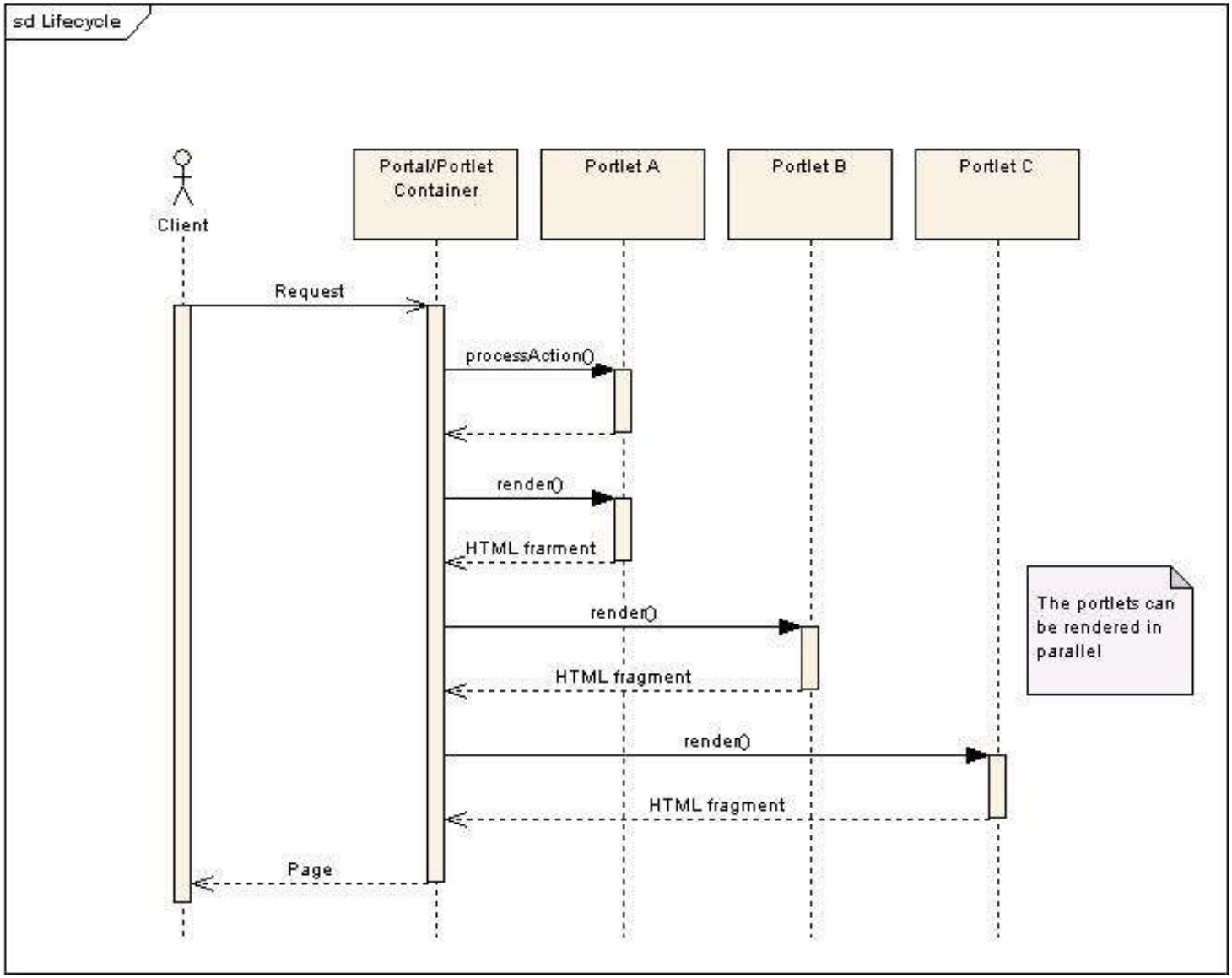
Ciclo de vida de un Portlet

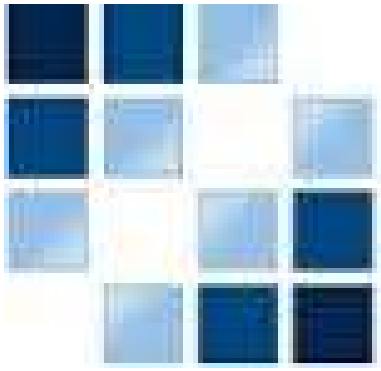
- Carga
- Inicialización
- Procesado de Acciones (1)
- Procesado de Eventos (todos los afectados)
- Renderizado (todos)
- Destrucción

Ciclo de vida de un Portlet



Flujo de una petición a un Portlet





LIFERAY

Enterprise. Open Source. For Life.

CONFIGURACIÓN

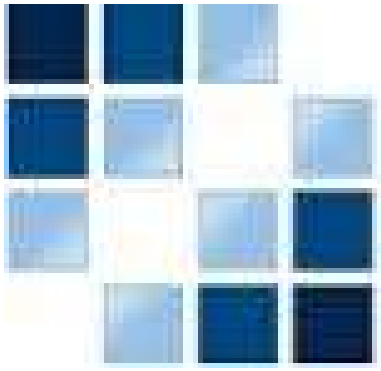
Configuración de un Portlet

- Fichero **portlet.xml**

```
<portlet>
        <portlet-name>NOMBRE_UNICO</portlet-name>
        <display-name>MiPortlet</display-name>
        <portlet-class>com.ext.portlet.MiPortlet</portlet-class>
        <supports>
                <mime-type>text/html</mime-type>
                <portlet-mode>VIEW</portlet-mode>
                <portlet-mode>EDIT</portlet-mode>
                <portlet-mode>HELP</portlet-mode>
                <window-state>NORMAL</window-state>
                <window-state>MINIMIZED</window-state>
                <window-state>MAXIMIZED</window-state>
        </supports>
</portlet>
```

Configuración de un Portlet

- Fichero **portlet.xml**
  - Se definen principalmente un nombre de Portlet, que ha de ser único en el Portal.
  - La clase que implementa la interface **javax.portlet.Portlet**, la más básica **GenericPortlet**.
  - Los modos de render aceptados (VIEW, EDIT y HELP son los por defecto).
  - Los estados de ventana aceptados (NORMAL, MAXIMIZED y MINIMIZED son los por defecto).



LIFERAY

Enterprise. Open Source. For Life.

CLASES DEL API

Clases del API

- Interfaz **Portlet**
  - destroy()
  - init(PortletConfig)
  - processAction(ActionRequest, ActionResponse)
  - render(RenderRequest, RenderResponse)

Clases del API

- Interfaz **PortletConfig**
  - getContainerRuntimeOptions()
  - getDefaultNamespace()
  - getInitParameter(String)
  - getInitParameterNames()
  - getPortletContext()
  - getPortletName()
  - getProcessingEventQNames()
  - getPublicRenderParameterNames()
  - getPublishingEventQNames()
  - getResourceBundle(Locale)
  - getSupportedLocales()

Clases del API

- Interfaz **PortletRequest**
  - ActionForward render(ActionMapping mapping, ActionForm form, PortletConfig PortletConfig, RenderRequest req, RenderResponse res);

Clases del API

- Interfaz **PortletResponse**
  - void addProperty(Cookie cookie);
  - void addProperty(String key, Element element);
  - void addProperty(String key, String value);
  - Element createElement(String tagName);
  - String encodeURL(String path);
  - String getNamespace();
  - void setProperty(String key, String value);

Clases del API

- Interfaz **PortletContext**
  - String getServerInfo();
  - InputStream getResourceAsStream(String path);
  - int getMajorVersion();
  - int getMinorVersion();
  - URL getResource(String path);
  - Object getAttribute(String name);

Clases del API

- Interfaz **PortletPreferences**
  - boolean isReadOnly(String key);
  - String getValue(String key, String def);
  - void setValue(String key, String value);
  - Enumeration<String> getNames();
  - void reset(String key);
  - void store();

Clases del API

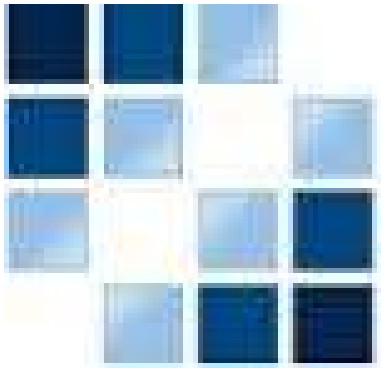
- Interfaz **ResourceServingPortlet**
  - serveResource(ResourceRequest, ResourceResponse)

Clases del API

- Interfaz **EventPortlet**
  - processEvent(EventRequest, EventResponse)

Clases del API

- Clase **GenericPortlet,** implementa **Portlet**, **PortletConfig**, **ResourceServingPortlet** y **EventPortlet**.
  - cacheAnnotatedMethods()
  - doDispatch(RenderRequest, RenderResponse)
  - doEdit(RenderRequest, RenderResponse)
  - doHeaders(RenderRequest, RenderResponse)
  - doHelp(RenderRequest, RenderResponse)
  - doView(RenderRequest, RenderResponse)
  - getNextPossiblePortletModes(RenderRequest)
  - getPortletConfig()
  - getTitle(RenderRequest)
  - init()



LIFERAY

Enterprise. Open Source. For Life.

GENERIC PORTLET

Implementación de un GenericPortlet

- Existen cuatro funcionalidades que se pueden implementar en un Portlet
  - **RenderView**.
  - **ProcessAction**.
  - **ProcessEvent**.
  - **Resource**.

Implementación de un GenericPortlet

- Las funcionalidades de render, se implementan sobrescribiendo los método del API
  - **doView**()
  - **doHelp**()
  - **doEdit**()
- O anotando un método con la misma firma (cambiando el nombre) que alguno de los anteriores con
  - **@RenderMode(name="VIEW")**

Implementación de un GenericPortlet

- Las funcionalidades de acción, se implementan sobrescribiendo el método del API
  - **processAction**()
- O anotando un método con la misma firma que el anterior con
  - **@ProcessAction(name="ejecutar")**
- El acceso a las acciones se debe hacer vía **POST** (Method de HTTP).

Acceso al Portlet.

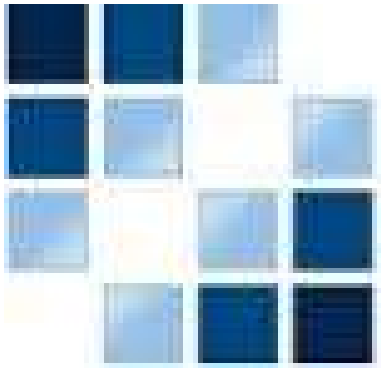
- Dado que pueden existir múltiples **Portlet** en una misma pagina o en paginas distintas, la especificación, incluye una forma sencilla de generar las URLs necesarias para interaccionar con los **Portlets**.
- Se permite la creación de URL

```
response.createActionURL();
actionURL.setParameter(ActionRequest.ACTION_NAME, “saludo”);
```

Acceso al Portlet.

- Ojo, que Liferay por defecto trabaja con **namespace** para los campos de los formularios, y al emplear la etiqueta <**input**> de **html** no se emplea **namespace**, por lo que la recogida del dato en el servidor no se produce, ya que se busca una combinación del **name** del <**input**> con **namepace**, excepto que se defina la siguiente propiedad en el **liferay-portlet.xml**

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



LIFERAY

Enterprise. Open Source. For Life.

RENDER MODE

Render Mode

- Permiten definir como se ha de pintar el Portlet, la vista a emplear.
- En el estándar se definen los siguientes modos
  - VIEW
  - EDIT
  - HELP
- Pudiéndose extender, practica habitual de las distintas implementaciones de Portales, como Liferay.

Render Mode

- Y en liferay se extienden a
  - ABOUT
  - CONFIG
  - PRINT
  - PREVIEW
  - EDIT_DEFAULTS
  - EDIT_GUEST
- Para poder emplear estos modos, se ha de emplear el API de Portlets de Liferay.

Render Mode

- Para activar los distintos modos, se ha de incluir en el fichero **portlet.xml**

```
<supports>
        <portlet-mode>VIEW</portlet-mode>
        <portlet-mode>EDIT</portlet-mode>
        <portlet-mode>CUSTOM</portlet-mode>
        <portlet-mode>ABOUT</portlet-mode>
        <portlet-mode>CONFIG</portlet-mode>
        <portlet-mode>PRINT</portlet-mode>
        <portlet-mode>PREVIEW</portlet-mode>
        <portlet-mode>EDIT_DEFAULTS</portlet-mode>
        <portlet-mode>EDIT_GUEST</portlet-mode>
</supports>
```

Render Mode

- Se pueden definir otros modos de renderización, incluyendo en el **portlet.xml** fuera de cualquier nodo portlet.

```
<custom-portlet-mode>
        <description>Modo de renderizacion CUSTOM</description>
        <portlet-mode>CUSTOM</portlet-mode>
        <portal-managed>false</portal-managed>
</custom-portlet-mode>
```

- Empleándose en el **Portlet** de la misma manera que los existentes

```
<supports>
        <portlet-mode>CUSTOM</portlet-mode>
</supports>
```

Render Mode

- Incluyendo en la clase que implementa el Portlet, un método con las siguientes características

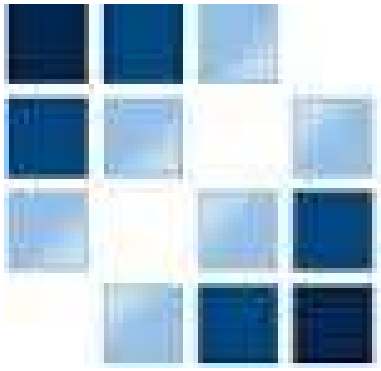
```
@RenderMode(name="CUSTOM")
public void doCustomRenderMode(RenderRequest renderRequest,
                RenderResponse renderResponse) throws IOException, PortletException
{

}
```

Render Mode

- Para acceder a un **Render Mode** concreto desde la vista, se ha de invocar una URL particular, estas URL cambian con cada implementación del Portal, por lo que se proporciona un API de etiquetas JSP, que permiten definirlas fácilmente.
- Se verá más adelante, pero sirva ahora de ejemplo.

```
<portlet:renderURL portletMode="CUSTOM" var="customRenderModeURL"/>
```



LIFERAY

Enterprise. Open Source. For Life.

VIEW STATE

View State

- Permiten definir la forma de pintar la vista del portlet
- En el estándar se definen los siguientes estados
  - NORMAL
  - MAXIMIZED
  - MINIMIZED

View State

- Para activar los distintos estados, se ha de incluir en el fichero **portlet.xml**

```
<supports>
        <window-state>MAXIMIZED</window-state>
        <window-state>MINIMIZED</window-state>
        <window-state>NORMAL</window-state>
</supports>
```

View State

- Se pueden definir otros estados de renderización, incluyendo en el **portlet.xml** fuera de cualquier nodo portlet.

```
<custom-window-state>
        <description>Nuevo Window STATE</description>
        <window-state>CUSTOM</window-state>
</custom-window-state>
```

- Empleándose en el **Portlet** de la misma manera que los existentes

```
<supports>
        <window-state>CUSTOM</window-state>
</supports>
```

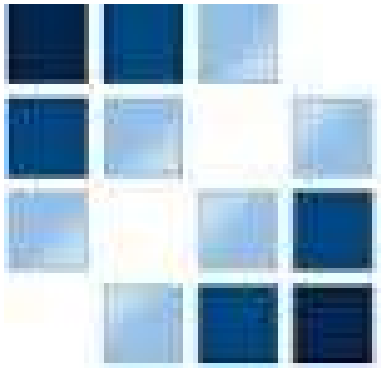
View State

- Para acceder a la característica definida como View State desde el Portlet, se tiene.

```
request.getWindowState();
```

- A la hora de construir las URLs para acceder a los distintos modos de renderización y acciones, se puede definir el View State en el que se quiere representar el Portlet.

```
<portlet:renderURL windowState="MAXIMIZED"
                   portletMode="VIEW“
                   var="viewRenderModeUrl">
</portlet:renderURL>
```



LIFERAY

Enterprise. Open Source. For Life.

PARAMETROS DE INICIO

Parámetros de Inicio

- Dentro del fichero **portlet.xml**, se pueden configurar asociado al Portlet parámetros de inicio (constantes).

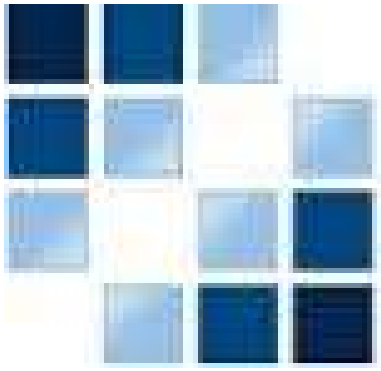
```
<init-param>
        <name>init-view-template</name>
        <value>/html/view.jsp</value>
</init-param>
```

- Habitualmente se emplean para definir las JSP que se emplearan como Vista.

Parámetros de Inicio

- Para recogerlos dentro del Portlet

```
String initView = getInitParameter("init-view-template");
```



LIFERAY

Enterprise. Open Source. For Life.

PREFERENCIAS

Preferencias

- Las preferencias de los Portlet, permiten definir unas variables persistentes asociadas al Portlet.
- Estas variables, pueden tener valores por defecto definidos en el **portlet.xml**

```
<portlet-preferences>
        <preference>
                <name>prefijo</name>
                <value>Hello </value>
        </preference>
</portlet-preferences>
```

Preferencias

- Los parámetros a definir son
  - **name**: Indica la clave de la preferencia.
  - **value**: Indica el valor o valores por defecto (puede haber mas de uno).
  - **read-only**: Indica que no puede ser escrito.
  - **preferences-validator**: Indica un clase que implementa **PreferencesValidator**, que sirve para validar el valor asignado a la preferencia al llamar a **store**(), lanzando un **ValidatorException** si no cumple la validación.

Preferencias

- Para recuperar las preferencias del **portlet.xml**, hay que ejecutar la siguiente sentencia

```
PortletPreferences preferences = request.getPreferences();
```

- El acceso se tiene tanto desde **RenderRequest**, como de **ActionRequest**.
- Para establecer un nuevo valor distinto al por defecto:

```
preferences.setValue("prefijo", prefijo);
preferences.store();
```

- O para leer su valor

```
preferences.getValue("prefijo", null);
```

Preferencias

- Ejemplo de clase **PreferencesValidator**.

```
public class SamplePreferencesValidator implements PreferencesValidator{
@Override
public void validate(PortletPreferences portletPreferences) throws ValidatorException {
        String miPreferencia = portletPreferences.getValue("MiPreferencia", "");
        List<String> valoresInvalidos = new ArrayList<String>();
         valoresInvalidos.add(miPreferencia);
        if(miPreferencia.equalsIgnoreCase("No Valido")){
                throw new ValidatorException(
                        "Se ha introducido un valor invalido",
                        valoresInvalidos);
        }
}
}
```

INTERNACIONALIZACIÓN

Internacionalización

- Se ha de definir un fichero de **properties** para cada idioma, que ha de colocarse dentro del **classpath**.
- El properties se ha de declarar en el fichero **portlet.xml**

```
<resource-bundle>com.my.portlets.Resource</resource-bundle>
```

- Además se han de declarar los idiomas soportados por el portlet.

```
<supported-locale>es</supported-locale>
<supported-locale>en_GB</supported-locale>
<supported-locale>en_US</supported-locale>
```

Internacionalización

- Se definirán los ficheros properties de cada idioma en la ruta indicada

```
<resource-bundle-name>_<language>_<country>.properties
```

- El uso de las propiedades internacionalizadas, se realiza a partir del API de **ResourceBoundle** de Java.

```
Locale locale = LanguageUtil.getLanguageId(request);
locale = request.getLocale();

ResourceBundle res = ResourceBundle.getBundle("Language", locale);

res.getString("message-key");
```

Internacionalización

- Aunque también se puede hacer uso del propio API de portlets.

```
ResourceBundle res = getResourceBundle(locale);
res = getPortletConfig().getResourceBundle(locale);

res.getString("message-key")
```

- Si el mensaje es parametrizado

```
String mensaje = MessageFormat.format(resourceBundle.getString("saludo"), “Juan");
```

Internacionalización

- En una JSP

```
<%@ taglib uri="http://liferay.com/tld/ui" prefix="liferay-ui" %>
<liferay-ui:message key="message-key" />
```

- Y con mensaje parametrizado

```
<%Object[] argumentos = new Object[]{"Juan"}; %>

<liferay-ui:message key="saludo" arguments="<%=argumentos%>" />
```

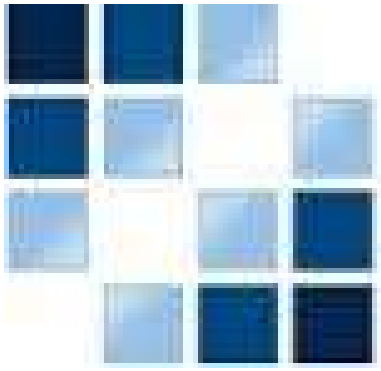
Internacionalización

- Para cambiar el idioma en el que se muestra el portal para un usuario concreto, se accede a

```
MyAccount --> Display Settings
```

- Para cambiar el idioma del portal por defecto se haría en

```
Control Panel --> Portal --> Settings ---> Display settings
```



LIFERAY

Enterprise. Open Source. For Life.

RECURSOS

Recursos asociados al Portlet

- Asociado a un Portlet pueden existir recursos, los cuales serán provistos por el método **serveResource().**
- Esta funcionalidad se asocia habitualmente al uso de AJAX.
- No hay anotación para esta funcionalidad.

Recursos asociados al Portlet

- Dado que actualmente Liferay esta integrado con Alloy, para realizar peticiones AJAX, lo mas recomendable es emplear este framework javascript.

```
http://alloyui.com/
```

- Para ello, se ha de emplear el objeto **AUI()**. Que es el eje central de este framework, de forma análoga a **$()** lo es en JQuery.

Recursos asociados al Portlet

- Su uso se basará principalmente en los método **ready** y **use**, de uso similar, el primero actuará cuando termine de cargarse el DOM, el segundo en el momento que se ejecute.
- Un ejemplo de uso de estos métodos

```
AUI().ready('aui-io-request', function(A){
        A.io.request('<%=updaContentURL%>', {
                dataType: 'json',
                        on: { success: function() {
                                alert(this.get('responseData').data);
                        }}
                });
        });
```

Ejercicio

- Crear un **Portlet** que proporcione un recurso, que sea un objeto **JSONArray**, con todos los usuarios registrados en el portal Liferay.
  - Emplear las clases **JSONFactoryUtil**, **JSONArray** y **JSONObject**, para generar el resultado.
- En la vista lanzar una petición **AJAX** sobre la URL del Recurso para obtener el **JSON**, e imprimir en la misma vista.

COMUNICACIÓN ENTRE FASES

Comunicación entre Fases

- Dada la naturaleza del ciclo de vida de los **Portlets**, se necesita un procedimiento por el cual intercambiar información entre las fases, dado que en una **petición de Acción**, en la fase de **Acción** se procesa el negocio y se obtiene el resultado, y en la fase de **Render** se pintara dicho resultado.

Comunicación entre Fases

- Existen tres mecanismos para intercambiar la información entre las fases
  - **RenderParameter**
  - **PortletSession**
  - **RequestAttribute**

Comunicación entre Fases

- **RenderParameter**.
  - Solo permite enviar String.
    - En el **action**

```
response.setRenderParameter("saludo", saludo);
```

    - En el **render**

```
request.getParameter("saludo");
```

  - Es persistente ante renderizaciones provocadas por otro **Portlet** de la pagina, por ejemplo una petición de acción de otro **Portlet**.

Comunicación entre Fases

- **PortletSession**.
  - Opción para JSR168.
  - Está desaconsejado.
  - Permite enviar objetos.
    - En el **action**

```
request.getPortletSession().setAttribute("saludo",  saludo,
                                                PortletSession.PORTLET_SCOPE);
```

    - En el **render**

```
request.getPortletSession().getAttribute("saludo",
                                                PortletSession.PORTLET_SCOPE)
```

Comunicación entre Fases

- **PortletSession**.
  - Se pueden definir dos ámbitos para la sesión
    - **Portlet_Scope**: Solo el mismo Portlet en esa sesión Web puede ver los objetos añadidos en ese ámbito.
    - **Application_Scope**: Cualquier Portlet, dentro de la misma aplicación web, puede ver los objetos de este ámbito, siempre que estén en la misma sesión web.

Comunicación entre Fases

- **RequestAttribute**.
  - Opción recomendada para JSR286.
  - Permite enviar objetos
    - En el **portlet.xml** (versiones anteriores a 6.2)

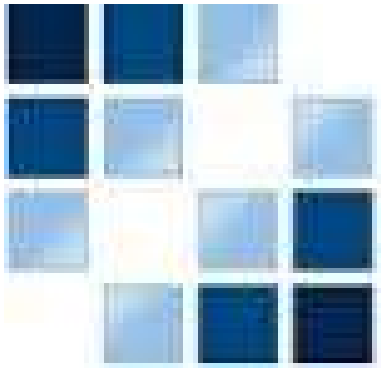
```
<container-runtime-option>
        <name>javax.portlet.actionScopedRequestAttributes</name>
        <value>true</value>
</container-runtime-option>
```

    - En el **action**

```
request.setAttribute("saludo", saludo);
```

    - En el **render**

```
request.getAttribute("saludo");
```



LIFERAY

Enterprise. Open Source. For Life.

IPC – COMUNICACIÓN ENTRE PORTLETS

Comunicación entre Portlets

- La comunicación entre Portlets en Liferay, se puede realizar de varias formas
  - En servidor
    - Portlet Session (Propio de Liferay)
    - Public Render Parameter
    - Eventos
  - En cliente
    - Eventos Javascript(Propio de Liferay)
    - Cookies

Comunicación entre Portlets

- **Portlet Session**:
  - Por defecto cada War es propietario de su propia sesión web, y no la comparte con otros War.
  - Liferay permite cambiar esto y que se pueda compartir.
  - Para ello en el fichero **liferay-portlet.xml**

```
<portlet>
        <private-session-attributes>false</private-session-attributes>
</portlet>
```

Comunicación entre Portlets

- **Portlet Session**:
  - Una vez se permite compartir la sesión web, se procede a trabajar con el ámbito **application_scope** del objeto **Session**.
  - En el portlet emisor

```
PortletSession session = renderRequest.getPortletSession();
session.setAttribute("sessionValue", "dato compartido desde el Portlet Emisor" ,
                                        PortletSession.APPLICATION_SCOPE);
```

  - En el portlet receptor

```
PortletSession ps = renderRequest.getPortletSession();
String dato = (String)ps.getAttribute("sessionValue",
                                        PortletSession.APPLICATION_SCOPE);
```

Comunicación entre Portlets

- **Public Render Parameter**:
  - Permite emplear los parámetros de Render en varios Portlet.

Comunicación entre Portlets

- **Public Render Parameter**:
  - Y en el fichero **portlet.xml**

```
<portlet>
        <supported-public-render-parameter>
                parametro
        </supported-public-render-parameter>
</portlet>
<public-render-parameter>
        <identifier>parametro</identifier>
        <qname xmlns:x="http://miDominio.com">
                x:parametro
        </qname>
</public-render-parameter>
```

Comunicación entre Portlets

- **Public Render Parameter**:
  - Y en el código, se accede de la misma forma que a los Render Parameter.
    - En el **processAction**

```
response.setRenderParameter(“parametro", “Dato a enviar");
```

    - En el **Render**

```
request.getParameter(“parametro”);
```

    - Se podría borrar el parámetro con

```
response.removePublicRenderParameter("parametro");
```

Comunicación entre Portlets

- **Eventos**:
  - Permite la comunicación entre Portlets, a base de eventos, introduciendo un nuevo componente en el ciclo de vida de los Portlets, similar al **ProcessAction**, denominado **ProcessEvent**, que se coloca en el flujo entre el **ProcessAction** y el **Render**.

Comunicación entre Portlets

- **Eventos**:
  - Se ha de definir en el **portlet.xml** del emisor

```
<portlet>
        <supported-publishing-event xmlns:x='http://liferay.com'>
                <qname>x:miEvento</qname>
        </supported-publishing-event>>
</portlet>
<event-definition xmlns:x='http://liferay.com'>
        <qname>x:miEvento</qname>
        <value-type>java.lang.String</value-type>
</event-definition>
```

  - La tipología del objeto a enviar deberá ser **Serializable**.

Comunicación entre Portlets

- **Eventos**:
  - Para enviar un evento desde el portlet emisor, se ha de codificar en un **processAction**

```
javax.xml.namespace.QName qName = new QName("http://liferay.com",
                                            "miEvento", "x");
response.setEvent(qName, “Dato a enviar");
```

Comunicación entre Portlets

- **Eventos**:
  - Y en el **portlet.xml** del receptor

```
<portlet>
        <supported-processing-event xmlns:x='http://liferay.com'>
                <qname>x:miEvento</qname>
        </supported-processing-event>
</portlet>
<event-definition xmlns:x='http://liferay.com'>
        <qname>x:miEvento</qname>
        <value-type>java.lang.String</value-type>
</event-definition>
```

Comunicación entre Portlets

- **Eventos**
  - Las funcionalidades de escucha de eventos, se implementan sobrescribiendo el método del API
    - **processEvent**()
  - O anotando un método con la misma firma que el anterior con
    - **@ProcessEvent**

Comunicación entre Portlets

- **Eventos**:
  - Definiendo un método en la clase del portlet que será la encargada de procesar el evento

```
@javax.portlet.ProcessEvent(qname = "{http://liferay.com}miEvento")
public void handleProcessempinfoEvent(javax.portlet.EventRequest request,
                                      javax.portlet.EventResponse response)
                                      throws javax.portlet.PortletException, java.io.IOException {

        javax.portlet.Event event = request.getEvent();
        String value = (String) event.getValue();
        System.out.print("Dato recibido en el evento>>>>>>>>>>" + value);
        response.setRenderParameter("miEvento", value);
}
```

Comunicación entre Portlets

- **Eventos Javascript**:
  - Liferay, proporciona un API javascript, para que se puedan lanzar y manejar eventos del lado del cliente.
  - Para ello en el Portlet Emisor, se ha de lanzar el evento con

```
Liferay.fire(
        'miEvento',              //Nombre del evento
        {                        //Objeto JSON
                dato: 'Este valor sera procesado por otro portlet'
        }
);
```

Comunicación entre Portlets

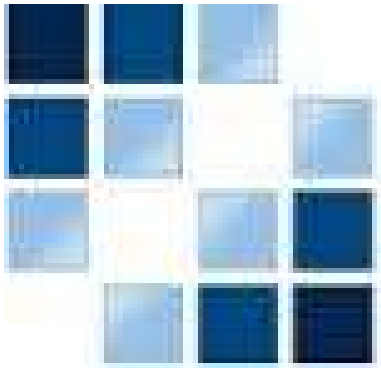
- **Eventos Javascript**:
  - Y en el Portlet Receptor, se ha de manejar el evento con

```
Liferay.on(
        'miEvento',             //Nombre del evento
        function(event) {       //Funcion manejadora que recibe el objeto JSON
                alert(event.dato);
        }
);
```

Comunicación entre Portlets

- **Cookies**:
  - Siempre se puede recurrir a que el Emisor deje en una cookie un valor y que Receptor vaya a buscar dicha cookie.
  - Para ello se puede emplear el API básico de javascript

```
document.cookie;
```



LIFERAY

Enterprise. Open Source. For Life.

LIBRERÍA DE ETIQUETAS ESTANDAR

Librería de etiquetas

- Existe una librería básica de potlets

```
<%@ taglib uri="http://java.sun.com/portlet_2_0" prefix="portlet" %>
```

- Ofrece las siguientes etiquetas
  - **defineObjects**
  - **namespace**
  - **actionURL**
  - **renderURL**
  - **resourceURL**
  - **param**
  - **Property (No funciona en Liferay)**

Librería de etiquetas

- Para definir una URL que cargue el Portlet en un modo de renderización concreto se emplea **renderURL**

```
<portlet:renderURL
        portletMode="EDIT"
        var="editRenderModeUrl">
</portlet:renderURL>
```

Librería de etiquetas

- Para definir una URL que interactue con el Portlet pidiendo que realice una acción se emplea **actionURL**

```
<portlet:actionURL
        name="accion"
        portletMode="VIEW"
        var="actionUrl">
</portlet:actionURL>
```

Librería de etiquetas

- La etiqueta **defineObjects**

```
<portlet:defineObjects />
```

- Crea en el ámbito de la JSP una serie de variables
  - **request**
  - **response**
  - **portletConfig**
  - **portletSession**
  - **portletSessionScope (atributos de sesión)**
  - **portletPreferences**
  - **portletPreferencesValues (map de preferencias)**

Librería de etiquetas

- La etiqueta **namespace**

```
<portlet:namespace/>
```

- Incluye un literal en la vista renderizada, que permite distinguir los componentes estáticos de la vista (formularios, javascript y demás elementos de HTML), entre las distintas instancias de Portlets existentes en una página.

Librería de etiquetas

- Para definir un parámetro dentro de una URL, se emplea la etiqueta **param**.

```
<portlet:renderURL var="editGreetingURL">
        <portlet:param name="mvcPath" value="/edit.jsp" />
</portlet:renderURL>
```

- Se puede definir una url a una jsp de forma directa

```
<%
PortletURL listBooksURL = renderResponse.createRenderURL();
listBooksURL.setParameter("jspPage", "/html/library/list.jsp");
%>
```