

Liferay

Victor Herrero Cazurro

Contenidos

Portlets	1
Ciclo de vida	2
Configuracion	4
Portlet desarrollado con el Estandar	5
Render Mode	6
View State	7
Init Param	8
Preferencias	8
Internacionalizacion	9
Comunicacion entre Fases	11
Eventos	12
Libreria de Etiquetas Estandar	13
AlloyUI	14
Libreria de JSPs AlloyUI	14
Formularios	14
Modelo	15
FieldSet	15
Input	15
Button Row	15
Button	15
Validator	16
Layout	16
Nav-Bar	16
Nav	17
Nav-Item	17
Script	18
Modulos	18
DOM	20
Event	22
Carousel	22
Javascript	24
Libreria Liferay-UI	24
Menu de Navegacion Pop-Up	24
Panel	25
Localizacion	25
Acceso a mensajes internacionalizados desde UI	26

Gestion Excepciones	27
Service Builder	29
Introduccion	29
Acceso a los datos	30
Entidades	31
Service.xml	31
Relaciones	34
1-1	34
1-n	35
n-m	35
Finders	36
Custom SQL	36
Dynamic Query	39
Restriciones	40
Roles	40
API	41
Permisos	41
Permisos para Portlets	43
Permisos para las entidades	44
Habilitacion de permisos sobre los registros	46
Comprobacion de permisos	47
Acceso a la pantalla de configuracion de Permisos de una entidad	48
Selector de permisos	49
Assets	49
Actualizacion/Insercion de un registro de tipo Asset	50
Borrado de un registro de tipo Asset	52
Asset Renderer - Gestion de Asset con herramientas genericas (Asset Publisher Portlet y Asset Renderer portlet)	53
Contenido Relacionado (Asset Related)	57
Añadir / Actualizar	57
Borrado	58
Modificacion desde UI	58
Visualizacion desde UI	58
Categorias y Tags	59
Valoraciones	60
Comentarios	60
Busquedas e Indices	61
API	65

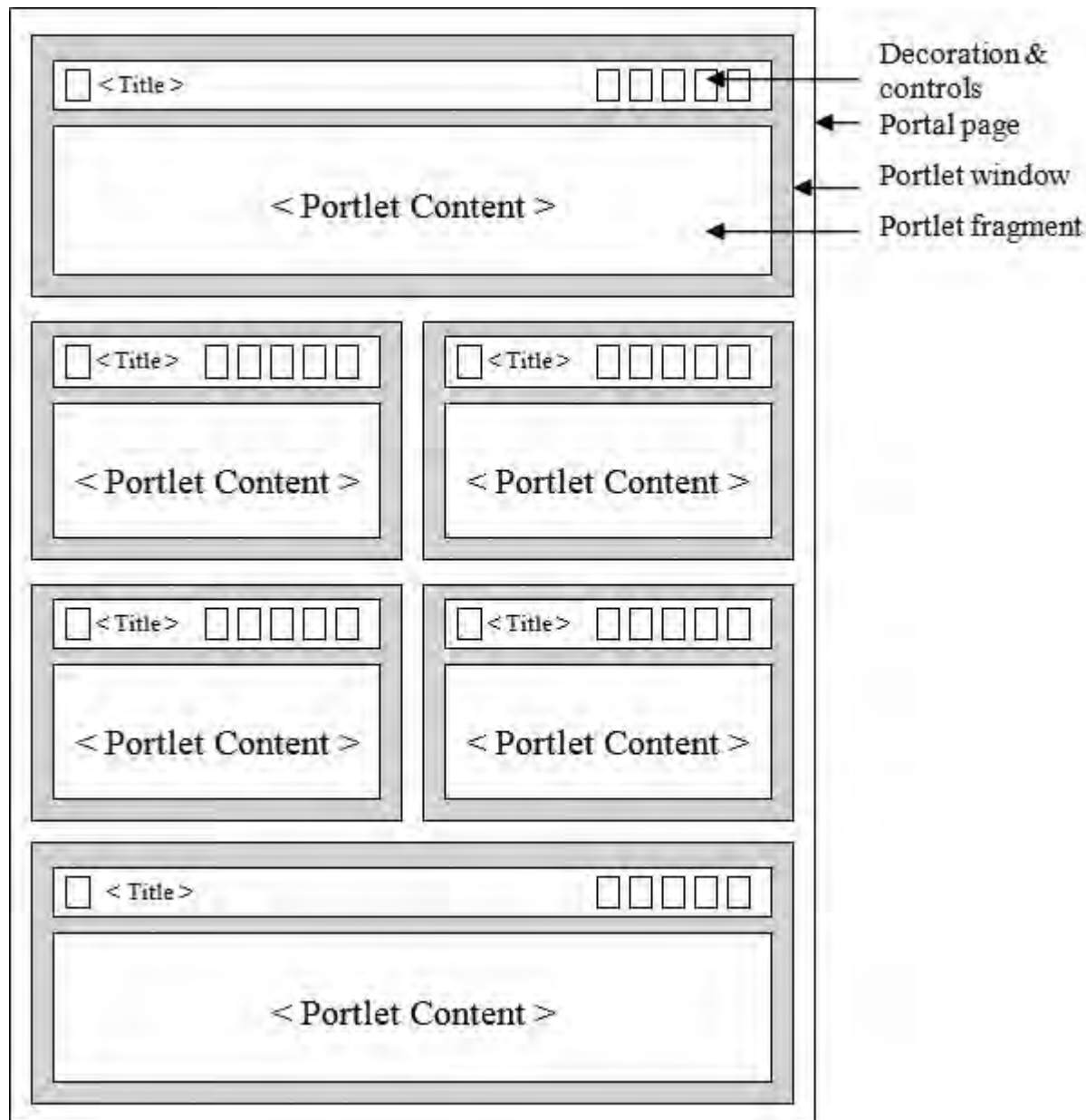
Implementacion de busqueda	65
Web Service	66
SOAP	66
JSON	67
Hook	67
Hook de propiedades	68
Hook de Idiomas	69
Hook de Actions de Struts	69
Hook de JSPs	70
Hook de Servicio	71
Hook de Indexador	71
Temas	72
Proyecto de tipo Tema	73
Parametrizacion del Tema	75
Plantillas	76
Thumbnail y Favicon	76
Estilos	77
Javascript.	77
Esquemas de color	77
Ext	78
Sobreescritura Servicio	78

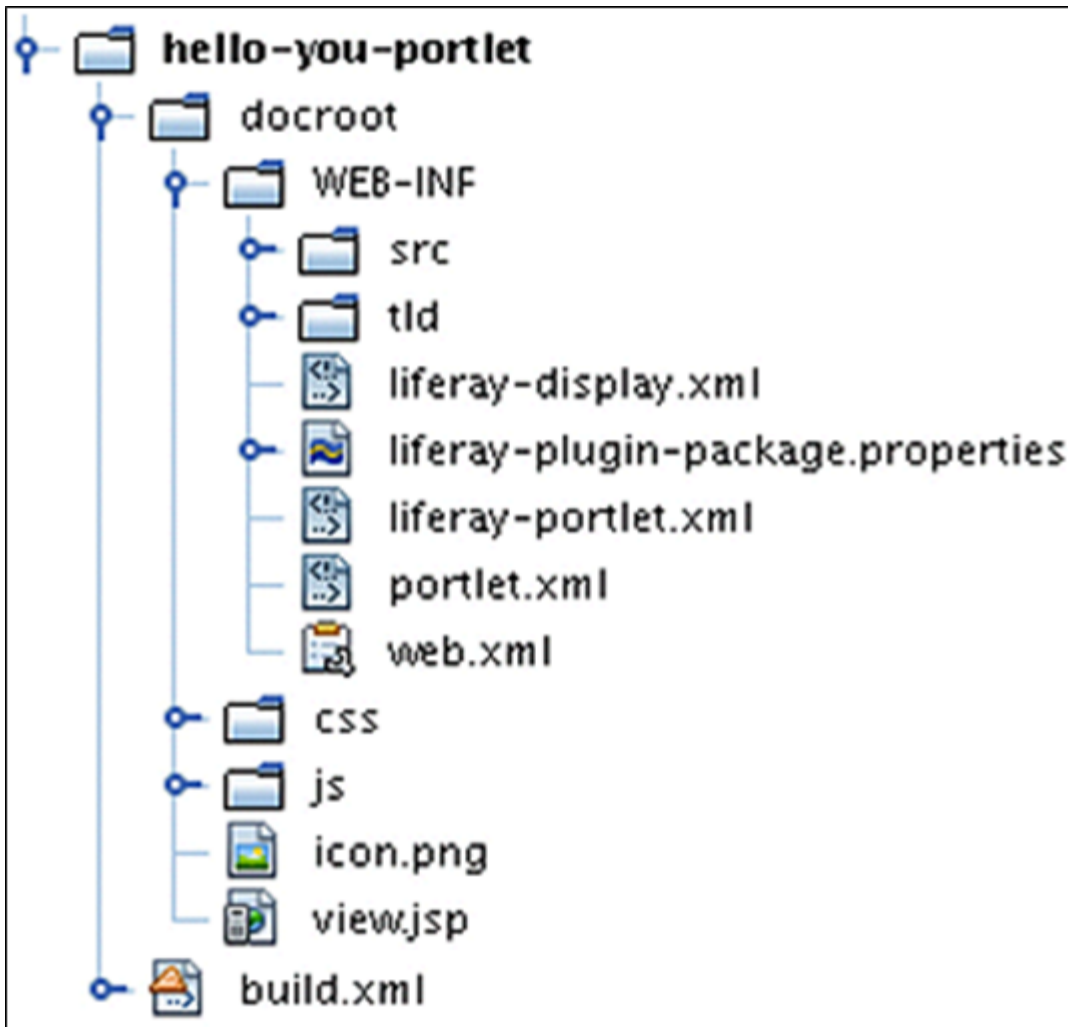
Portlets

Portlet → Componente que define la capa de presentación de los **Portales** (mini-aplicación). Se les asocia un **Modo de renderización** (VIEW, EDIT, HELP) y un **Estado** (Normal, Maximizado, Minimizado).

Portal → Aplicación web, que puede trabajar con Portlets, dispone de un **Contenedor de Portlets** y define sus páginas con Portlets.

Contenedor de Portlets → Entorno de ejecución para los Portlets, es una extensión del contenedor de Servlets que maneja el ciclo de vida y almacena las preferencias de los Portlets.

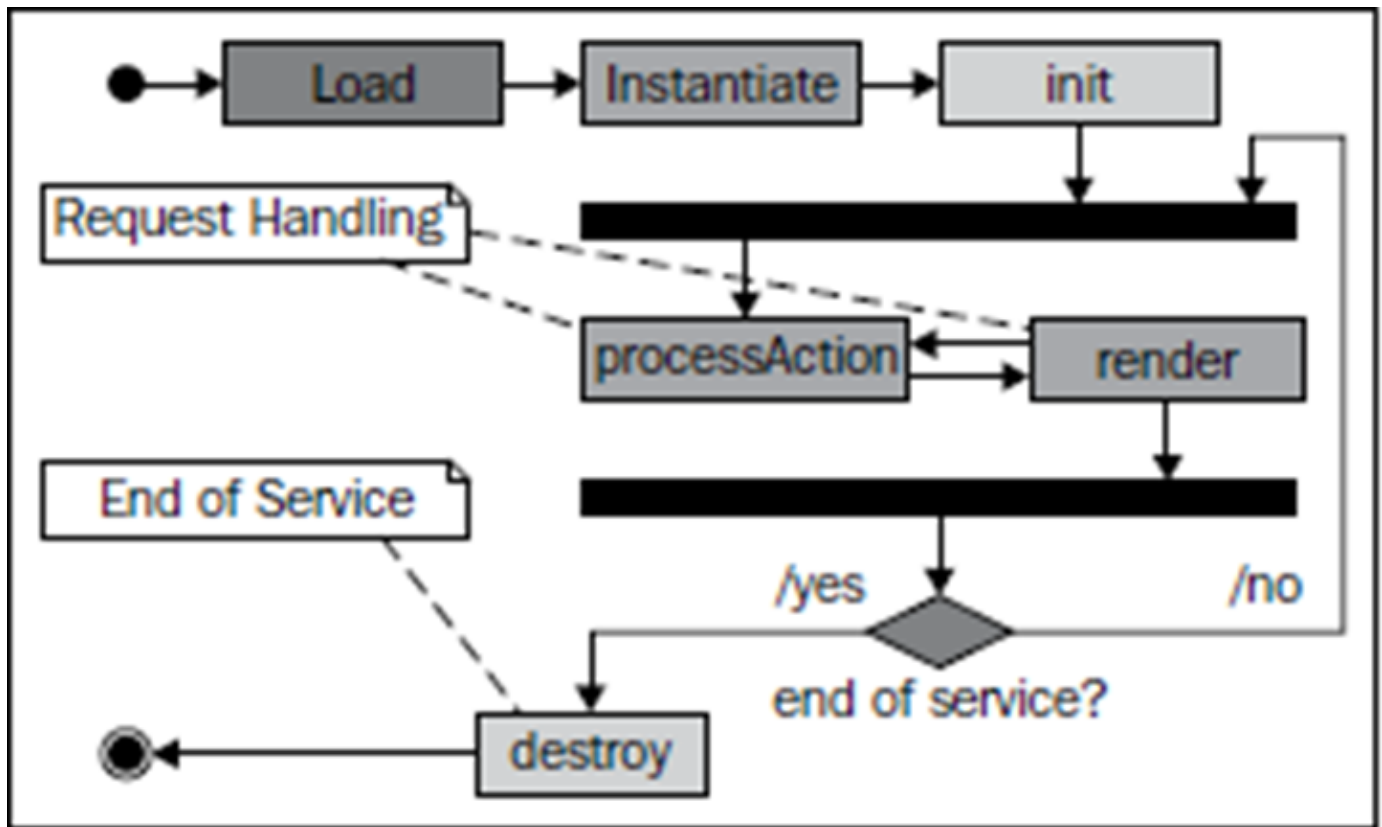


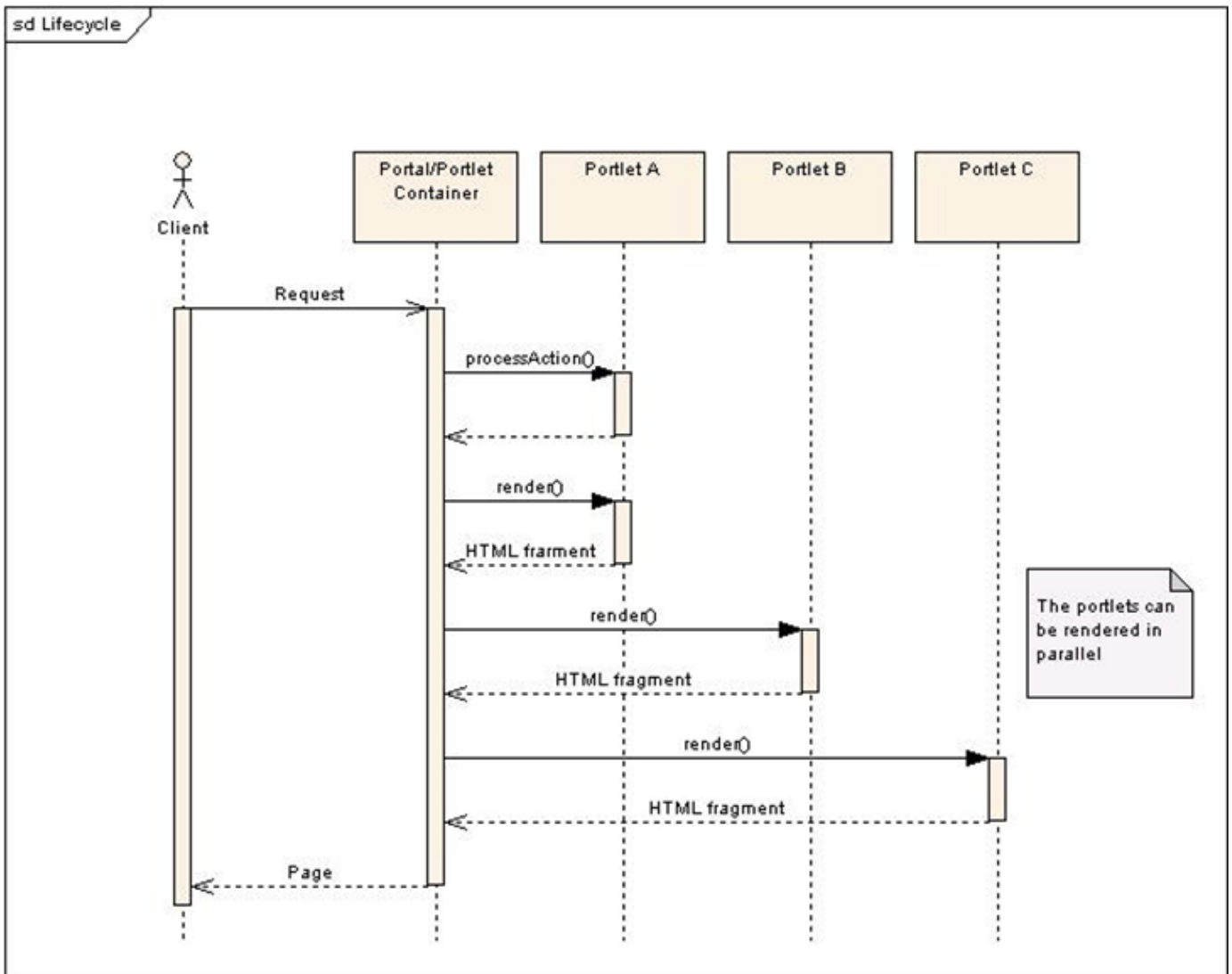


Ciclo de vida

El ciclo de vida de un Portlet, se divide en las siguientes fases

- Inicialización → Dado que los Portlets se instancian desde el contenedor, se permite inicializarlos con el método **init**
- Procesado de Acciones → Es la fase en la que el Portlet responde ante la interacción del usuario. Solo se ejecutará en el Portlet afectado.
- Procesado de Eventos → Es la fase en la que el Portlet responde ante un evento provocado por otro Portlet. Se ejecutará en todos los Portlets que hayan definido la escucha del evento lanzado.
- Renderizado → Fase en la que se realiza el pintado del Portlet. Se ejecutará en todos los Portlets de la página visualizada.
- Destrucción → Cuando el Portlet ya no es empleado.





Configuracion

La configuracion de un Portlet se realiza en el fichero **Portlet.xml**, donde se han de configurar:

- 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).


```

<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>

```

Portlet desarrollado con el Estandar

El desarrollo de un Portlet estandar se basa en la herencia de la clase **GenericPortlet** y la implementacion de los siguientes métodos

- Para la fase de Render
 - doView()
 - doHelp()
 - doEdit()
 - O anotando un método con la misma firma (cambiando el nombre) que alguno de los anteriores con la anotación `@RenderMode(name="VIEW")`

```

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

}

```

- Para la fase de Action
 - processAction()
 - O anotando un método con la misma firma que el anterior con la anotación `@ProcessAction(name="ejecutar")`

NOTE | El acceso a las acciones se debe hacer vía POST (Method de HTTP).

- Para la fase de Events

- processEvent()
- O anotando un método con la misma firma que el anterior con **@ProcessEvent**
- Para la fase de Resource
 - serveResource()

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, donde se dispone de

- ABOUT
- CONFIG
- PRINT
- PREVIEW
- EDIT_DEFAULTS
- EDIT_GUEST

Para poder emplear estos modos, se ha de emplear el API de Portlets de Liferay.

Para activar los distintos modos, se ha de incluir en la configuracion de Portlet

```
<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>
```

Se pueden definir nuevos, definiendo de forma global un nuevo **custom-portlet-mode**

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

Para acceder a un **Render Mode** de un Portlet concreto, se ha de invocar una URL particular, dado que los Portlet pueden estar en distintas paginas, e incluso haber mas de una instancia del mismo en la pagina, la generacion de estas URL, es dinamica, por lo que se precisa de ayuda para obtenerlas.

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

View State

Permiten definir la forma de pintar la vista del portlet. En el estándar se definen los siguientes estados

- NORMAL
- MAXIMIZED
- MINIMIZED

Para activar los distintos estados, se ha de incluir en la configuracion de Portlet

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

Se pueden definir nuevos, definiendo de forma global un nuevo **custom-window-state**

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

Para acceder a esta característica desde el Portlet, se dispone del método **request.getWindowState()**.

Para indicar este parametro en la renderizacion

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

Init Param

Se pueden configurar parametros de inicio asociados al Portlet (constantes), habitualmente se emplean para definir las JSP que se emplearan como Vista.

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

Para recogerlos en la implementacion.

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

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>
```

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.

```
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);
        }
    }
}
```

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);
```

Internacionalizacion

Se ha de definir un fichero de properties para cada idioma, que ha de colocarse dentro del classpath y declararse 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>
```

Se definirán los ficheros properties de cada idioma en la ruta indicada siguiendo el formato **<resource-bundle-name>_<language>_<country>.properties**.

El uso de las propiedades internacionalizadas, se realiza a partir del API de ResourceBundle de Java, empleando en API de Liferay, en concreto la clase **LanguageUtil**

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

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

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

O haciendo uso del 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");
```

En las JSPs para acceder a los mensajes, se importa el taglib de **liferay-ui**

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

Empleando la etiqueta

```
<liferay-ui:message key="message-key" />
```

Y si el mensaje es parametrizado

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

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

Comunicacion 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.

Existen tres mecanismos para intercambiar la información entre las fases

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

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

- En el render

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

- PortletSession → Opción para JSR168. En versiones posteriores esta desaconsejado.
- RequestAttribute → 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");
```

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.

Se ha de declarar en el Portlet 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>
```

Y en el 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>
```

NOTE El tipo de objeto a compartir, ha de ser Serializable.

Para provocar el evento

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

La escucha del 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);
}

```

Libreria de Etiquetas Estandar

El estandar define una libreria de etiquetas para ayudar a generar los JSP, para emplearla se ha de declarar su uso en las cabeceras del JSP

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

Ofrece las siguientes etiquetas

- defineObjects → define las siguientes variables en el ambito de la JSP:
 - request
 - response
 - portletConfig
 - portletSession
 - portletSessionScope (atributos de sesión)
 - portletPreferences
 - portletPreferencesValues (map de preferencias)

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

- 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.

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

- actionURL

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

- renderURL

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

- param → Para definir un parámetro en una URL.

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

- resourceURL

AlloyUI

Librería javascript, que define componentes avanzados.

Liferay proporciona librerías de etiquetas JSP que se basan en dichos componentes.

El desarrollo se basa en la carga por parte del **Theme** de las librerías javascript.

NOTE

Eclipse en los proyectos de tipo Maven, no encuentra los TLDs en los jar, así que para poder emplear los TLD ya sean de Liferay o incluso de JSTL, hay que copiar los ficheros en la carpeta **/WEB-INF/tld**

Librería de JSPs AlloyUI

Para emplearla, hay que añadir el taglib a la JSP

```
<%@ taglib uri="http://alloy.liferay.com/tld/au" prefix="au" %>
```

Formularios

```
<au:form action="<%= addEntryURL %>" name="<portlet:namespace />fm">  
</au:form>
```

Modelo

Permite definir un Bean a emplear para rellenar los campos del formulario. Se puede emplear para editar con un formulario un dato existente.

*Ejemplo de uso, donde al renderizar el formulario, el input **name** tendra como valor el campo **name** del bean **entry***

```
<au:form action="<%= addEntryURL %>" name="<portlet:namespace />fm">
  <au:model-context bean="<%= entry %>" model="<%= Entry.class %>" />
  <au:input name="name"></au:input>
</au:form>
```

FieldSet

Agrupacion de campos de formulario

```
<au:fieldset>
</au:fieldset>
```

Input

Define un campo del formulario, puede ser de varios tipos: text, textarea, hidden, ... para lo cual se configura el atributo **type**

```
<au:input name="name">
</au:input>
```

Button Row

Permite definir una fila de botones

```
<au:button-row>
</au:button-row>
```

Button

Permite definir un boton de formulario

```
<au:button-row>
  <au:button id="useEmailButton" value="Use My Email Address"></au:button>
</au:button-row>
```

Validator

Permite establecer una validacion sobre un campo de entrada, existen varios validadores: email, required, ...

```
<au:input name="email" >
  <au:validator name="email" />
  <au:validator name="required" />
</au:input>
```

Layout

Permite definir una distribucion de la pantalla, basada en columnas, indicando dentro de cada columna los elementos que compondrán dicha columna

```
<au:layout>
  <au:column>
    <div id="message1-div"></div>
  </au:column>
</au:layout>
```

- **columnWidth:** Indica el ancho de la columna
- **first:** Indica que es la primera columna
- **last:** Indica que es la última columna

Nav-Bar

```

<au:nav-bar>
  <au:nav>
    <au:nav-item href="/..." label="view-all" selected='<%= "view-
all".equals(toolbarItem) %>' />
    <au:nav-item dropdown="true" iconCssClass="icon-plus"
      label="add" selected='<%= "add".equals(toolbarItem) %>'>
      <au:nav-item href="/..." label="user" />
      <au:nav-item href="/..." label="organization" />
      <au:nav-item href="/..." label="group" />
    </au:nav-item>
    <au:nav-item href="/..." iconCssClass="icon-download"
      label="export" selected='<%= "export".equals(toolbarItem) %>' />
  </au:nav>
</au:nav-bar>

```

Nav

Permite la definicion de un menu de navegacion.

Se puede representar de varias formas: pestañas, ...

*Representacion en forma de pestañas, empleando la clase CSS **nav-tabs***

```

<au:nav cssClass="nav-tabs">
</au:nav>

```

Nav-Item

Cada una de los elementos que componen el menu de navegacion. Estan compuestos por

- **cssClass** → Estilo de pintado, para la representacion en pestañas, se emplea **active**
- **href** → URL que se ha de cargar al seleccionar la opción, es habitual que se cargue siempre la misma ventana, cambiando los datos cargados a traves de parametros asociados al enlace y en caso de representacion con pestañas, el estilo de la pestaña seleccionada.
- **label** → Texto que se asocia a la opcion del menu

```
<au:nav cssClass="nav-tabs">

    <portlet:renderURL var="viewPageURL">
        <portlet:param name="mvcPath" value="/html/guestbook/view.jsp" />
        <portlet:param name="guestbookId"
            value="<%=String.valueOf(curGuestbook.getGuestbookId())%>" />
    </portlet:renderURL>

    <au:nav-item cssClass="<%=cssClass%>" href="<%=viewPageURL%>" au:nav>
```

Script

Permite emplear apis de la libreria JS de AlloyUI.

Se han de definir los modulos a emplear en la propiedad **use**, pudiendo tomar como valores: node, event, au-char-counter, ...

Dentro del script existirá una variable **A** que permite el acceso a los apis de los modulos cargados.

```
<au:script use="node, event">
    var useNameButton = A.one('#useNameButton');
</au:script>
```

Los modulos disponibles en Liferay se pueden encontrar en **<Liferay portal bundle directory>\tomcat-7.0.62\webapps\ROOT\html\js\au**

De definir las funcionalidades javascript en un fichero a parte, se ha de definir el bloque

```
AUI().use('au-base', function (A){
});
```

Que es equivalente al uso de la etiqueta **<au:script>** en las jsp.

Modulos

Para emplear un modulo, hay que declararlo sobre el objeto **AUI()**

```

<au:script use="au-char-counter">
AUI().use(
    function(A) {
    }
);
</au:script>

```

Existen multiples modulos que se pueden emplear en el API de AlloyUI, algunos de ellos son:

- **au-char-counter** → Permite contar los caracteres que restan por introducir en un input, definiendo un máximo. Se necesita definir:
 - Un campo de entrada de texto, de emplear el API de AUI, tener en cuenta que se introduce de forma automatica el **Namespace**.
 - Un contenedor para incluir el numero de caracteres calculado
 - El Javascript que reliza el calculo, empleando el modulo **au-char-counter**.

```

<au:input id="message" type="text" name="message">
    <au:validator name="required" errorMessage="Please enter a message." />
</au:input>

<div id="counterContainer"><p><span id="counter"></span> character(s) remaining</p></div>

<au:script use="au-char-counter">
AUI().use(
    function(A) {
        new A.CharCounter(
            {
                counter: '#counter',
                input: '#<portlet:namespace />message',
                maxLength: 140
            }
        );
    }
);
</au:script>

```

- **node**
- **event**
- **Carousel**

DOM

El modulo **node** permite el manejo del arbol DOM de componentes HTML que se genera en los navegadores.

Permite realizar principalmente tareas de selección, como

- one
- get
- all
- select
- after
- next

```
<c:if test="<%= themeDisplay.isSignedIn() %>">
  <au:script use="node">

    var name = A.one('#<portlet:namespace/>name');

    name.val('<%= user.getFullName() %>');

    var email = A.one('#<portlet:namespace/>email');

    email.val('<%= user.getEmailAddress() %>');

  </au:script>
</c:if>
```

Y tambien de validación y edicion del arbol DOM

- hasChildNodes
- append

```
<au:button-row>
  <au:button
    id="generateMessagesButton"
    value="Generate Sample Messages">
  </au:button>
</au:button-row>

<div id="messages">
  <au:layout>
    <au:column>
```



```

        <div id="message1-div"></div>
    </aui:column>
</aui:layout>
</div>

<aui:script use="node, event">

    var generateMessagesButton = A.one('#generateMessagesButton');

    var message1Div = A.one('#message1-div');

    generateMessagesButton.on('click', function(event) {

        var entryMessages = [
            'Amazing!',
            'Be careful!',
            'Best wishes!',
            'Bravo!',
            'Congratulations!',
            'Great job!',
            'Have fun!',
            "How's it going?",
            'You did it!',
            "Wow!"
        ];

        if (message1Div.hasChildNodes()) {
            message1Div.get('children').remove(true);
        }

        var rand1 = Math.floor(Math.random() * entryMessages.length);

        message1Div.append(
            '<p class="message" id="message1">' + entryMessages[rand1] +
            '</p><p id="use-message1">' +
            '<input class="btn" onclick="useMessage1();" type="button" value="Use
Message" /></p>');

        entryMessages.splice(rand1, 1);
    });
</aui:script>

```

Tareas de lectura de atributos de los nodos

- html
- val

```
option.val();
```

- attr

```
option.attr('value');
```

Event

El modulo **Event** permite definir escuchadores de eventos que se produzcan sobre los componentes del arbol DOM

```
<c:if test="<%= themeDisplay.isSignedIn() %>">

    <au:script use="node, event">

        var useNameButton = A.one('#useNameButton');

        useNameButton.on('click', function(event) {
            var name = A.one('#<portlet:namespace/>name');

            name.val('<%= user.getFullName() %>');
        });

        var useEmailButton = A.one('#useEmailButton');

        useEmailButton.on('click', function(event) {
            var email = A.one('#<portlet:namespace/>email');

            email.val('<%= user.getEmailAddress() %>');
        });
    </au:script>
</c:if>
```

Carousel

Permite visualizar un conjunto de imagenes, que se muestran de forma dinamica.

Para emplearlo, se han de seguir una serie de pasos:

- Incluir las imagenes en el proyecto, tipicamente en el directorio **img** del contenido web, para proyectos Maven **src/main/webapp/img**
- Definir en el HTML una estructura similar a la siguiente, con tantos bloques **div** como imagenes

compongan el **Carousel**

```
<div id="myCarousel">
  <div id="image1"></div>
  <div id="image2"></div>
</div>
```

- Añadir el script que controla el **Carousel**

```
<au:script>
  AUI().use('au-carousel',function(Y) {
    new Y.Carousel(
      {
        contentBox: '#myCarousel',
        height: 250,
        width: 700
      }
    ).render();
  });
</au:script>
```

- Definir estilos CSS que optimicen la visualización de las imágenes, así como la referencia a las imágenes a mostrar.

```
div.carousel-item
{
  width: 700px;
  height: 250px;
  opacity: 100;
}

#image1
{
  background-image: url("../img/YourImageFile1.jpg");
}

#image2
{
  background-image: url("../img/YourImageFile2.jpg");
}
```

Javascript

Liferay proporciona un API para poder obtener información del portal también desde el Javascript, para ello ofrece el objeto **Liferay**.

A partir de este, se puede obtener acceso a otros como **ThemeDisplay**, que ofrece la siguiente información

- `getCompanyId`
- `getLanguageId`
- `getScopeGroupId`
- `getUserId`
- `getUserName`
- `getPathImage`
- `getPathJavaScript`: Path relativo al directorio que contiene los ficheros javascript del portlet.
- `getPathMain`: Path del directorio principal de la instancia del portal.
- `getPathThemeImages`
- `getPathThemeRoot`
- `isImpersonated`: Indica si el usuario actual está siendo suplantado por un administrador.
- `isSignedIn`

```
if(Liferay.ThemeDisplay.isSignedIn()){
    alert('Hello ' + Liferay.ThemeDisplay.getUserName() + '. Welcome Back.')
}
else {
    alert('Hello Guest.')
}
```

Libreria Liferay-UI

Para emplearla, hay que añadir el taglib a la JSP

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

Menu de Navegacion Pop-Up

La etiqueta `<liferay-ui:icon-menu>`, permite definir un listado de etiquetas hijas `<liferay-ui:icon>` que

definen el menú.

```
<liferay-ui:icon-menu>
  <liferay-ui:icon iconCssClass="icon-user" message="Use" url="www.liferay.com" />
  <liferay-ui:icon iconCssClass="icon-film" message="Film" url="www.liferay.com" />
  <liferay-ui:icon iconCssClass="icon-edit" message="Edit" url="www.liferay.com" />
  <liferay-ui:icon iconCssClass="icon-trash" message="Trash" url="www.liferay.com" />
</liferay-ui:icon-menu>
```

La etiqueta **<liferay-ui:icon-menu>** dispone entre otros de los siguientes atributos configurables.

- **direction:** Posibles direccion en la que se despliega el menu. Puede valer left, right, up, o down. El valor por defecto es left.
- **maxDisplayItems:** Numero de iconos a mostrar en el area visible, antes de usar croll, el valor por defecto es 15.
- **message:** Texto a mostra en el boton que muestra el menú. El valor por defecto es actions.
- **showArrow:** Establece cuando se muestra una flecha dentro del boton de accion, por defecto es true.
- **useIconCaret:** Establece si cambia la direccion del icono al abrir el menu, por defecto es false.

Panel

Permite definir paneles colapsables que agrupan componentes visuales.

```
<liferay-ui:panel defaultState="closed" extended="<%= false %>" id="populatePanel"
persistState="<%= true %>" title="populate">
</liferay-ui:panel>
```

- **defaultState:** Indica como aparece el panel por defecto
- **persistState:** Indica si se mantendrá el estado último del panel ante recargas de la pagina

Localizacion

Liferay permite la internacionalizacion de los mensajes mostrados a los usuarios.

Se definen un gran numero de mensajes ya internacionalizados en el fichero **portal-impl.jar/src/content/Language.properties**

Se pueden añadir nuevos mensajes configurando un nuevo Bundle para la internacionalizacion. Para el Bundle se define el paquete donde se encuentra y el nombre del fichero, sin codigo idiomático y sin extension.

Configuración en portlet.xml del Bundle

```
<portlet>
  ...
  <supported-locale>es</supported-locale>
  <supported-locale>en</supported-locale>
  <resource-bundle>content.Language</resource-bundle>
  ...
</portlet>
```

La ubicación del Bundle es relativo al Classpath **src/main/java**

Es buena práctica si existen más de un portlet en el proyecto, que el path del bundle incluya el nombre del proyecto.

Configuración en portlet.xml del Bundle

```
<resource-bundle>content.miportlet.Language</resource-bundle>
```

Existen dos características que todo portlet debería tener internacionalizadas

Propiedades internacionalizadas

```
javax.portlet.title=Mi Portlet
javax.portlet.description=Este es un portlet customizado
```

NOTE

El SDK proporciona una herramienta para la traducción de los properties de forma automática, basada en **Microsoft Translator**, para lo cual hay que dar de alta el servicio en la plataforma **Azure**.

Acceso a mensajes internacionalizados desde UI

Para emplear los mensajes internacionalizados, se pueden utilizar varias vías

- Librería de etiquetas de Liferay **liferay-ui**

```
<liferay-ui:message key="titulo" />

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

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

<liferay-ui:error key="MiErrorEnProperties"/>
```

Para el anterior ejemplo, deberán existir en el properties los siguientes pares clave-valor

```
titulo=Esto es español  
saludo= Hola {0}
```

Y se tendrá que haber añadido al objeto **SessionErrors** un error con clave **MiErrorEnProperties**

```
SessionErrors.add(request, "MiErrorEnProperties", "SE ha producido un error en el Action  
");
```

- Librería de etiquetas JSTL de formateo **fmt**

```
<fmt:message bundle="%=new  
LocalizationContext(portletConfig.getResourceBundle(request.getLocale()))%"  
key="saludo">  
    <fmt:param value="Antonio"></fmt:param>  
</fmt:message>
```

Gestion Excepciones

La gestión de excepciones se basa en la gestión de mensajes al usuario, para ello se proporcionan dos APIs

- SessionMessages → Mensajes afirmativos al usuario.
- SessionErrors → Mensajes negativos al usuario.

```

@Override
public void processAction(
    ActionRequest actionRequest, ActionResponse actionResponse)
    throws IOException, PortletException {

    PortletPreferences prefs = actionRequest.getPreferences();
    String greeting = actionRequest.getParameter("greeting");

    if (greeting != null) {
        try {
            prefs.setValue("greeting", greeting);
            prefs.store();
        }
        catch (Exception e) {
            SessionErrors.add(actionRequest, "error");
        }
    }

    SessionMessages.add(actionRequest, "success");
    super.processAction(actionRequest, actionResponse);
}

```

Y en la vista se proporcionan las etiquetas

- liferay-ui:success
- liferay-ui:error


```

<%@ taglib uri="http://java.sun.com/portlet_2_0" prefix="portlet" %>
<%@ taglib uri="http://liferay.com/tld/ui" prefix="liferay-ui" %>
<%@ page import="javax.portlet.PortletPreferences" %>

<portlet:defineObjects />

<liferay-ui:success key="success" message="Greeting saved successfully!" />

<liferay-ui:error key="error" message="Sorry, an error prevented saving your greeting" />

<%
PortletPreferences prefs = renderRequest.getPreferences();
String greeting = (String)prefs.getValue("greeting", "Hello! Welcome to our portal.");
%>

<p><%= greeting %></p>

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

<p>
    <a href="<%= editGreetingURL %>">Edit greeting</a>
</p>

```

Service Builder

Introduccion

Permite definir una capa de Persistencia y de Servicio de forma automatica, considerando las funcionalidades mas basicas.

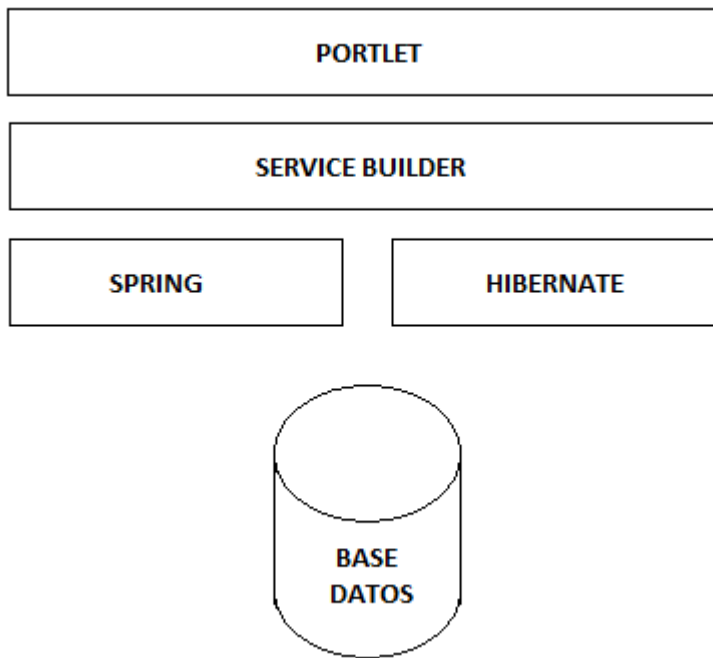
La idea de Service Builder, es homogeneizar la capa de servicios y de persistencia de las aplicaciones de Liferay, para ello proporciona un API y una herramienta (dentro del plugin de maven de Liferay) que permite la creación de las clases de estas capas de forma automática siguiendo el patrón model-driven.

El propio Liferay esta creado empleando esta herramienta y la arquitectura que propone.

La herramienta, creará la capa de modelo y la de servicio de forma independiente, incluyendo las funcionalidades básicas de CRUD, permitiendo expandir dichas funcionalidades definiendo relaciones, Finders, Custom SQL, ...

Esta tecnología, emplea por debajo **Hibernate** y **Spring**, aunque en principio no se tiene porque tocar

las configuraciones de estos frameworks, siendo su uso transparente para el desarrollador de **service-builder**.



No hay un tipo de proyecto propio para Service Builder, sino que se creará a partir de un proyecto de Portlets. Esto tiene su sentido ya que cualquier modelo de datos que se quiera incluir en el Portal, tendrá que ser consumido por algún Portlet.

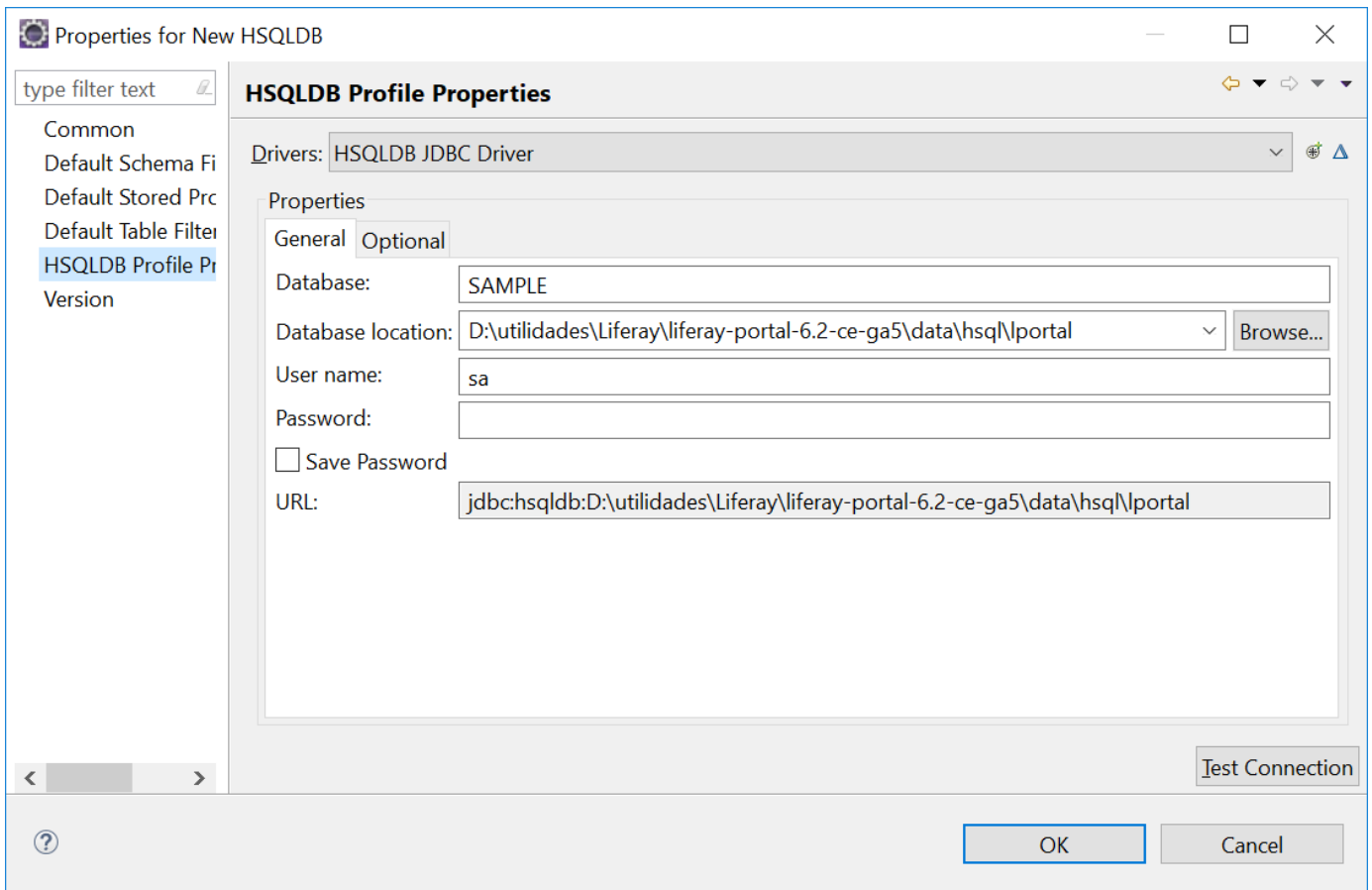
Los proyectos Maven, crean la siguiente estructura de proyectos

- **XX-Service-Builder:** Proyecto padre Maven, que aglutina configuraciones comunes a los otros dos proyectos.
- **XX-Service-Builder-portlet:** Proyecto destinado a tener los Portlets, la implementación del servicio y donde se configurará el propio servicio con el fichero `service.xml`
- **XX-Service-Builder-portlet-service:** Proyecto con las interfaces y clases autogeneradas del servicio, de la cual depende el proyecto **XX-Service-Builder-portlet**.

Acceso a los datos

Una vez generado el modelo de datos, se puede consultar accediendo a la BD, en el caso de la HSQL que viene configurada por defecto, la conexión se haría con estos datos

- Cadena de conexión → `jdbc:hsqldb:<directorio instalacion liferay>\data\hsql\lportal`
- user → SA
- password → <vacío>



Entidades

Se definen las entidades en el fichero **WEB-INF/service.xml**. Que empleará el siguiente esquema

```
<!DOCTYPE service-builder PUBLIC "-//Liferay//DTD Service Builder 6.2.0//EN"
"http://www.liferay.com/dtd/liferay-service-builder_6_2_0.dtd">
```

Una vez definidas las entidades, se ha de lanzar la tarea **build-service** de Maven para que se autogeneren las clases necesarias.

Las clases **-impl** generadas, tanto para **Model**, **Service** y **Persistence**, son modificables, pudiendo cambiar los comportamientos definidos por la herramienta de generacion de código.

Cualquier modificación sobre estas clases, que añada un nuevo método o cambie la firma de uno existente, exigirá que se vuelva a lanzar la tarea de Maven **build-service**.

Service.xml

El nodo principal de este fichero será **<service-builder>**, que tendrá como atributos

- **package-path**: Paquete java donde se encontrarán las entidades definidas.

- **auto-namespace-tables:** Crear el namespace de forma automática para las tablas.

Y como nodos hijos

- **author:** Autor del modelo.
- **namespace:** Prefijo que se incluye en el nombre de la tabla con el formato.
- **exceptions:** Excepciones propias que se pueden lanzar al trabajar con el modelo.
- **service-builder-import:** Referencia a otro fichero que defina entidades, para partir en varios ficheros la definición de la capa de servicios.
- **entity:** Etiqueta principal, que permite definir las entidades que define el modelo.

El nodo `<entity>` tiene como atributos

- **name:** Nombre de la entidad.
- **uuid:** (booleano) Indica si se ha de generar este valor de forma automática.
- **local-service:** Si se permite acceder con la interface local.
- **remote-service:** Si se permite acceder con la interface remota, creando el servicio web.
- **table:** Nombre de la tabla a emplear para diferenciarla de la clase que representa la entidad.
- **cache-enabled:** Permite activar el cacheo de las entidades.
- **deprecated:** Si se considera la entidad en desuso.
- **human-name:** Nombre legible de la entidad empleado a la hora de generar la documentación.
- **json-enabled:** Si se permite el acceso a través del servicio web con JSON.
- **persistence-class:** Nombre de la clase creada hija de `XXPersistenceImpl`.
- **session-factory:** Bean de Spring que gestiona las sesiones de hibernate.
- **data-source:** Bean de Spring que gestiona las conexiones a la base de datos.
- **tx-manager:** Bean de Spring que gestiona las transacciones.
- **trash-enabled:** Si se habilita la posibilidad de enviar a la papelera estas entidades.
- **uuid-accessor:** (booleano) indica si se genera el Getter para el uuid

Y como nodos hijos

- **column:** Definición de un campo de la entidad.
- **finder:** Nueva funcionalidad de búsqueda relacionada con la entidad, dará lugar a unos nuevos métodos en la clase de implementación de la persistencia `XXPersistenceImpl`.
- **order:** Establece como se ordenan los registros retornados por las consultas por defecto.
- **reference:** Indica referencias a servicios ya existentes en Liferay o definidos en otro `service.xml` pero que carguen en el mismo classpath, para que estén accesibles directamente en la clase de implementación de servicio.

- **tx-required:** Patrón de nombre para todos aquellos métodos que necesiten transacciones, por defecto los siguientes patrones ya están incluidos*: add*, check*, clear*, delete*, set*, and update*, el resto serán de solo lectura.

El nodo **<column>**, puede definir relaciones con otras entidades del portal, así como unos campos particulares de la arquitectura de Liferay

Portal and site scope columns

Name	Type	Primary
companyId	long	no
groupId	long	no

User column

Name	Type	Primary
userId	long	no

Audit columns

Name	Type	Primary
userId	long	no
createDate	Date	no
modifiedDate	Date	no

Los campos

- **companyId:** hace referencia a la instancia del Portal
- **groupId:** hace referencia al Sitio

La referencia a estos campos permite que los distintos Sitios e instancias de Portal tengan sus propios datos de estas tipologías.

El nodo column tiene como atributos

- **name:** Nombre del campo empleado en los Get y Set.
- **type:** Tipo Java del campo.
- **primary:** Si es clave primaria, pueden existir varias columnas formado la clave primaria.
- **entity:** Cuando el type es Collection, indica el tipo de objetos en la colección.
- **accessor:** (booleano) Si se accede por Get y Set o por propiedad.
- **convert-null:** (booleano) Convertir a null al insertar en BD.

- **db-name:** Nombre del campo en BD.
- **filter-primary:** (booleano) Permite indicar una clave primaria para los finder, sino se indica será la clave primaria por defecto.
- **id-type:** Se emplea para definir el método de generación de la clave primaria, puede ser*: class, increment, sequence o identity.
- **id-param:** Se necesita para definir la clase o la secuencia que genera la clave primaria en el tipo de generación class o sequence. En caso de emplear la secuencia, esta ha de definirse en el fichero
- **json-enabled:** (booleano) Si está habilitada la conversión a JSON
- **lazy:** (booleano) Si la carga es perezosa.
- **localized:** (booleano) Si acepta traducciones.
- **mapping-table:** Indica el nombre de la tabla intermedia en relaciones m-n.

El nodo **<order>** tiene como atributos

- **by:** Indica si la ordenación es asc o desc.

El nodo **<order>** como sub-nodos puede tener

- **order-column:** Que indica las columnas que participan en la ordenación, pudiendo haber una o más.

El nodo **<order-column>** tiene como atributos

- **name:** Nombre de la columna.
- **case-sensitive:** (boolean) Si la ordenación contempla mayúsculas y minúsculas.
- **order-by:** En lugar del atributo by del nodo order, permite indicar un tipo de ordenación distinto para cada columna.

Relaciones

Se pueden definir relaciones

- 1-1
- 1-n
- n-m

1-1

Basta con añadir una columna a la entidad que refleje la FK con la otra entidad.

```
<entity name="Subasta">
    <column name="subastaId" type="long" primary="true" />
    <column name="productoId" type="long"/>
</entity>
```

Con esto se consigue que en el Servicio se genere un método **getProducto**, que habrá que implementar

```
public Producto getProducto() throws PortalException, SystemException{
    return ProductoLocalServiceUtil.getProducto(getProductoId());
}
```

1-n

Similar a la anterior, pero en este caso habrá que definir el tipo de la entidad a la que se referencia y como **type** de la columna **Collection**

```
<entity name="Puja">
    <column name="pujaId" type="long" primary="true" />
</entity>
<entity name="Subasta">
    <column name="subastaId" type="long" primary="true" />
    <column name="pujas" type="Collection" entity="Puja" mappingKey="pujaId"/>
</entity>
```

n-m

Similar a la anterior, pero en este caso se ha de definir en las dos entidades la relacion, haciendo referencia a la tabla que mantiene la relacion

```
<entity name="Asignatura">
    <column name="asignaturaId" type="long" primary="true" />
    <column name="alumnos" type="Collection" entity="Alumno" mappingKey="alumnoId"
mapping-table="Asignaturas_Alumnos"/>
</entity>
<entity name="Alumno">
    <column name="alumnoId" type="long" primary="true" />
    <column name="asignaturas" type="Collection" entity="Asignatura" mappingKey=
"asignaturaId" mapping-table="Asignaturas_Alumnos"/>
</entity>
```

Finders

Métodos que se generan en la capa de persistencia, que definen consultas sencillas, con clausulas de **where**.

Se emplea el nodo **finder**.

```
<finder name="producto" return-type="Collection">
  <finder-column name="producto" />
</finder>
```

El nodo **finder-column** define los campos por los que se realiza la búsqueda, definiendo los atributos que recibirán los métodos generados, si se aplican múltiples **finder-column** al **finder**, se aplica por defecto la cláusula **AND**.

El nodo **finder** tiene los siguientes atributos

- **db-index**: (boolean) Si es true, se genera automáticamente un índice de BD.
- **name**: Nombre del método generado en la capa de persistencia.
- **return-type**: Tipo retornado, puede ser Collection o una Entidad.
- **unique**: Indica que se retorna solo una entidad.
- **where**: Permite definir una clausula de where estatica, que no se ve afectada por los parametros de la consulta **id != 1**

El nodo **finder-column** tiene como atributos:

- **arrayable-operator**: Permite definir el valor AND o OR, y generara un nuevo método de finder, que aceptará un array por parámetro y creara una clausula concatenando todos los valores del array con AND o OR.
- **case-sensitive**: Solo si la columna es de tipo String.
- **comparator**: Indica el tipo de comparación implementada por el método finder, puede tomar como valores: =, !=, <, <=, >, >=, o LIKE.
- **name**: Nombre de la columna.

Custom SQL

Permiten realizar consultas SQL nativas, por lo que se pueden lanzar consultas mas complejas que con los **finder**.

Se tendrán que definir las consultas en un fichero **src/main/resources/custom-sql/default.xml** siguiendo la siguiente estructura


```

<custom-sql>
  <sql id="[nombre canonico de la clase + metodo]">
    Consulta envuelta con <![CDATA[...]]>
    No terminar con punto y coma
  </sql>
</custom-sql>

```

Donde las consultas por convenio tendran como **id**, el nombre de la clase **Finder** mas el nombre del método que ejecutará la consulta, los cuales habrá que definir.

Ejemplo de definicion de consulta

```

<?xml version="1.0" encoding="UTF-8"?>
<custom-sql>
  <sql id=
"com.liferay.docs.guestbook.service.persistence.GuestbookFinder.buscarPorNombre">
    <![CDATA[
      SELECT GB_Guestbook.*
      FROM GB_Guestbook
      WHERE
        GB_Guestbook.name LIKE ?
    ]]>
  </sql>
</custom-sql>

```

NOTE

Ojo con el identificador, si se emplea el nombre de la clase, que en la implementación del **Finder**, no llevará **Impl**.

Para el caso anterior, se debería crear la clase **GuestbookFinderImpl** y dentro el método **buscarPorNombre**, que en este caso retornará un **List<Guestbook>** y recibirá un **String name**.

En un principio hay que hacer extender esta clase de **BasePersistenceImpl<>** e implementar la lógica del método, para lo cual el API proporciona los siguientes servicios.

- **CustomSQLUtil** → Permite obtener la consulta definida en el fichero xml.

```
String sql = CustomSQLUtil.get(GuestbookFinder.class.getName() + ".buscarPorNombre");
```

- **QueryPos** → Permite sustituir las ? por los valores concretos.

```

SQLQuery q = session.createSQLQuery(sql);
QueryPos qPos = QueryPos.getInstance(q);
qPos.add(name);

```

- **QueryUtil** → Permite realizar la ejecución de una sentencia SQLQuery indicando paginación

```
(List<Guestbook>) QueryUtil.list(q, getDialect(), begin, end);
```

Una vez implementado el método se ha de lanzar la tarea de **build-service**, para que genere una nueva interface, en este caso **GuestbookFinder**, que habrá que implementar.

```
public class GuestbookFinderImpl extends BasePersistenceImpl<Guestbook> implements
GuestbookFinder {

    public static final String FIND_BY_NOMBRE = GuestbookFinder.class.getName()
        + ".buscarPorNombre";

    public List<Guestbook> buscarPorNombre(String name, int begin, int end) {

        Session session = null;
        try {
            session = openSession();

            String sql = CustomSQLUtil.get(FIND_BY_NOMBRE);

            SQLQuery q = session.createSQLQuery(sql);
            q.setCacheable(false);
            q.addEntity("GB_Guestbook", GuestbookImpl.class);

            QueryPos qPos = QueryPos.getInstance(q);
            qPos.add(name);

            return (List<Guestbook>) QueryUtil.list(q, getDialect(), begin, end);
        } catch (Exception e) {
            try {
                throw new SystemException(e);
            } catch (SystemException se) {
                se.printStackTrace();
            }
        } finally {
            closeSession(session);
        }

        return null;
    }
}
```

Una vez generada la clase que implementa la consulta, el siguiente paso es consumirlo desde el servicio, para ello se ha de crear el método pertinente en **-ServiceImpl**, que tendrá acceso a la clase

GuestbookFinderUtil.

```
public List<Guestbook> buscarPorNombre(String name, int begin, int end) {  
    return GuestbookFinderUtil.buscarPorNombre(name, begin, end);  
}
```

Despues de añadir el método al Servicio, recordar regenerar las clases con la tarea **build-service**.

Dynamic Query

API que permite definir consultas de forma programatica.

Se basa en las clase **DynamicQuery** y **DynamicQueryFactoryUtil**.

El API de **ServiceBuilder**, define métodos en la capa de servicios que permiten ejecutar **DynamicQuery**

- List dynamicQuery(DynamicQuery dynamicQuery)
- List dynamicQuery(DynamicQuery dynamicQuery, int start, int end)
- List dynamicQuery(DynamicQuery dynamicQuery, int start, int end, OrderByComparator orderByComparator)
- long dynamicQueryCount(DynamicQuery dynamicQuery)
- long dynamicQueryCount(DynamicQuery dynamicQuery, Projection projection)

Algunos ejemplos de uso del API

```
DynamicQuery guestbookQuery = DynamicQueryFactoryUtil.forClass(Guestbook.class)  
    .add(RestrictionsFactoryUtil.eq("name", guestbookName))  
    .setProjection(ProjectionFactoryUtil.property("guestbookId"));  
  
List<Guestbook> guestbooks = GuestbookLocalServiceUtil.dynamicQuery(guestbookQuery);
```

```
DynamicQuery entryQuery = DynamicQueryFactoryUtil.forClass(Entry.class)  
    .add(RestrictionsFactoryUtil.eq("name", entryName))  
    .add(PropertyFactoryUtil.forName("guestbookId").in(guestbookQuery))  
    .addOrder(OrderFactoryUtil.desc("modifiedDate"));  
  
List<Entry> entries = EntryLocalServiceUtil.dynamicQuery(entryQuery);
```

Restricciones

Para las restricciones se dispone de **RestrictionsFactoryUtil**, que permite definir restricciones de tipo

- and
- between
- gt
- lt
- eq
- like
- in
- isEmpty
- isNull

Para definir restricciones respecto a un campo, se tiene **PropertyFactoryUtil** que permite referencias a los campos.

Roles

En el fichero portlet.xml, se definen Roles a nivel del desarrollo del Portlet.

Contenido del fichero portlet.xml

```
<portlet>
  ....
  <security-role-ref>
    <role-name>administrator</role-name>
  </security-role-ref>
  <security-role-ref>
    <role-name>guest</role-name>
  </security-role-ref>
  <security-role-ref>
    <role-name>power-user</role-name>
  </security-role-ref>
  <security-role-ref>
    <role-name>user</role-name>
  </security-role-ref>
</portlet>
```

Pero estos no tienen por que existir en Liferay, por lo que se proporciona la posibilidad de establecer un mapeo entre los del Portlet y los de Liferay.

```
<role-mapper>
  <role-name>administrator</role-name>
  <role-link>Administrator</role-link>
</role-mapper>
<role-mapper>
  <role-name>guest</role-name>
  <role-link>Guest</role-link>
</role-mapper>
<role-mapper>
  <role-name>power-user</role-name>
  <role-link>Power User</role-link>
</role-mapper>
<role-mapper>
  <role-name>user</role-name>
  <role-link>User</role-link>
</role-mapper>
```

Donde los roles **Administrator**, **Guest**, **Power User** y **User**, deben existir en Liferay.

API

Una vez definidos los mapeos entre roles, desde el código del Portlet, se puede acceder a la información de la seguridad, con los métodos:

- **getRemoteUser()**
- **isUserInRole()** → Permite conocer si el usuario logado, tiene un Rol determinado, los roles a empear serán los definidos a nivel del Portlet.

```
if (renderRequest.isUserInRole("power-user")) {
}
```

- **getUserPrincipal()** → Obtiene el objeto Principal

Permisos

Liferay permite definir nuevos permisos a asociar tanto a los Portlets, como a las entidades. Para ello habrá que definir un fichero XML con el esquema

esquema del fichero **default.xml**

```
<?xml version="1.0"?>
<!DOCTYPE resource-action-mapping PUBLIC "-//Liferay//DTD Resource Action Mapping
6.2.0//EN"
"http://www.liferay.com/dtd/liferay-resource-action-mapping_6_2_0.dtd">
```

El fichero por convencion se denomina **default.xml** y se ubica relativo al classpath **src/main/java** se ha de configurar a través de la propiedad **resource.actions.configs** del fichero **src/main/java/portlet.properties**

*seccion de porlet.properties que configura el fichero en el que se delega la configuracion del esquema de seguridad, en este caso el fichero estará en **src/main/java/resource-actions/default.xml***

```
resource.actions.configs=resource-actions/default.xml
```

NOTE

Se puede encontrar la configuracion de permisos para los portlets del core en **portal/portal-impl/src/resource-actions**

El nodo principal de dicho esquema es

*nodo principal del esquema del fichero **default.xml***

```
<resource-action-mapping>

</resource-action-mapping>
```

Se pueden definir permisos asociados a

- Portlets

nodo de configuracion de permisos asociados a un portlet

```
<portlet-resource>
  <portlet-name>guest-book</portlet-name>
  <permissions>
    ...
  </permissions>
</portlet-resource>
```

- Entidades del modelo

```
<model-resource>
  <model-name>com.liferay.docs.guestbook.model.Entry</model-name>
  <portlet-ref>
    <portlet-name>guest-book</portlet-name>
  </portlet-ref>
  <permissions>
</model-resource>
```

Permisos para Portlets

Se puede configurar la pagina de configuracion de permisos de los Portlet, indicando que permisos son configurables y que roles tienen ciertos permisos asociados por defecto.

El acceso a la pagina de configuracion de los permisos para el portlet, se obtiene desde el menú de configuracion del portlet con la opción **Configuracion**.

Se pueden añadir nuevas acciones para las cuales se pueden asociar roles que pueden realizarlas, para definir las acciones

```
<portlet-resource>
  <portlet-name>guest-book</portlet-name>
  <permissions>
    <supports>
      <action-key>GUESTBOOK-ADD</action-key>
    </supports>
    ....
  </permissions>
</portlet-resource>
```

Se pueden activar acciones para los roles por defecto del portal

```

<portlet-resource>
  <portlet-name>guest-book</portlet-name>
  <permissions>
    <supports>
      <action-key>GUESTBOOK-ADD</action-key>
    </supports>
    <site-member-defaults>
      <action-key>VIEW</action-key>
      <action-key>GUESTBOOK-ADD</action-key>
    </site-member-defaults>
    <guest-defaults>
      <action-key>VIEW</action-key>
    </guest-defaults>
    <guest-unsupported>
      <action-key>GUESTBOOK-ADD</action-key>
    </guest-unsupported>
  </permissions>
</portlet-resource>

```

NOTE

Todas las **action-key** soportadas por defecto por el portal se pueden encontrar en la clase **ActionKey**.

Permisos para las entidades

Similar a los anteriores, pudiendose filtrar por un paquete de entidades o por clases concretas.

```

<!-- Permisos sobre el paquete de model -->
<model-resource>
  <model-name>com.liferay.docs.guestbook.model</model-name>
  <portlet-ref>
    <portlet-name>guest-book</portlet-name>
  </portlet-ref>
  <permissions>
    <supports>
      <action-key>ADD_GUESTBOOK</action-key>
      <action-key>ADD_ENTRY</action-key>
    </supports>
    <site-member-defaults>
      <action-key>ADD_ENTRY</action-key>
    </site-member-defaults>
    <guest-defaults />
    <guest-unsupported>
      <action-key>ADD_GUESTBOOK</action-key>
      <action-key>ADD_ENTRY</action-key>
    </guest-unsupported>
  </permissions>
</model-resource>

```



```

        </guest-unsupported>
    </permissions>
</model-resource>

<!-- Permisos sobre la clase Guestbook en particular -->
<model-resource>
    <model-name>com.liferay.docs.guestbook.model.Guestbook</model-name>
    <portlet-ref>
        <portlet-name>guest-book</portlet-name>
    </portlet-ref>
    <permissions>
        <supports>
            <action-key>ADD_ENTRY</action-key>
            <action-key>DELETE</action-key>
            <action-key>PERMISSIONS</action-key>
            <action-key>UPDATE</action-key>
            <action-key>VIEW</action-key>
        </supports>
        <site-member-defaults>
            <action-key>ADD_ENTRY</action-key>
            <action-key>VIEW</action-key>
        </site-member-defaults>
        <guest-defaults>
            <action-key>VIEW</action-key>
        </guest-defaults>
        <guest-unsupported>
            <action-key>UPDATE</action-key>
        </guest-unsupported>
    </permissions>
</model-resource>

<!-- Permisos sobre la clase Entry en particular -->
<model-resource>
    <model-name>com.liferay.docs.guestbook.model.Entry</model-name>
    <portlet-ref>
        <portlet-name>guest-book</portlet-name>
    </portlet-ref>
    <permissions>
        <supports>
            <action-key>DELETE</action-key>
            <action-key>PERMISSIONS</action-key>
            <action-key>UPDATE</action-key>
            <action-key>VIEW</action-key>
        </supports>
        <site-member-defaults>
            <action-key>VIEW</action-key>
        </site-member-defaults>
        <guest-defaults>

```

```

        <action-key>VIEW</action-key>
    </guest-defaults>
    <guest-unsupported>
        <action-key>UPDATE</action-key>
    </guest-unsupported>
</permissions>
</model-resource>

```

Habilitacion de permisos sobre los registros

Hasta ahora solo se han definido que permisos por defecto se van a asignar a los Portlets y a los bean de Modelo, pero para los Bean de modelo, no son efectivos, habrá que registrar dichos permisos en la Base de Datos.

Los permisos en el Backend (Service builder) se llaman **Resources**, el API proporciona un servicio **ResourceLocalService** que permite gestionar los permisos.

El primer paso será, al añadir un nuevo registro de la entidad, registrar tambien sus permisos. En el método **add** del **LocalServiceImpl** despues de haber añadido el registro, dado que se precisa su **id**, se registran los permisos sobre el nuevo registro.

NOTE

Los Servicios creados con Service Builder ya obtienen la dependencia con **resourceLocalService**.

```

resourceLocalService.addResources(serviceContext.getCompanyId(), serviceContext
    .getScopeGroupId(), userId,
    Guestbook.class.getName(), guestbookId, false, true, true);

```

El primer boolean indica si el permiso es de Portlet, en caso de ser false, será un permiso de entidad.

El segundo boolean indica si se han de añadir permisos de grupo.

El tercer boolean indica si se han de añadir permisos de Guest (invitado)

De forma analoga, a la insercion de los permisos asociados a un registro, se han de borrar dichos permisos cuando el registro se elimina.

```

resourceLocalService.deleteResource(serviceContext.getCompanyId(), Entry.class.getName(),
    ResourceConstants.SCOPE_INDIVIDUAL,
    entryId);

```

NOTE

La constante **ResourceConstants.SCOPE_INDIVIDUAL**, se emplea cuando se elimina un recurso con un usuario como unico propietario.

Comprobacion de permisos

La comprobacion de permisos en Liferay, se basa en el servicio **PermissionChecker**, al cual se tiene acceso a traves de **ThemeDisplay**.

```
PermissionChecker permissionChecker = serviceContext.getThemeDisplay()
    .getPermissionChecker();
permissionChecker.hasPermission(entry.getGroupId(), Entry.class.getName(), entry
    .getEntryId(), actionId);
```

El segundo parametro, puede hacer referencia a una clase o a un paquete.

El ultimo parametro, será el literal que represente la acción (permiso) que se desea validar que el actual usuario tiene.

Por convencion, la comprobacion de permisos en el core de Liferay se basa en clases estaticas Helper, que por convenio adoptan el nombre **<Aplicacion>ModelPermission** cuando se trata de permisos a nivel de todas las clases del modelo y **<Entidad>Permission** para clases particulares del modelo. Luego por mantener la coherencia se recomienda seguir esta arquitectura, que fomenta la claridad del codigo y su reutilizacion.

Ejemplo de clase que comprueba los permisos genericos sobre las clases del modelo

```
public class GuestbookModelPermission {

    public static final String RESOURCE_NAME = "com.liferay.docs.guestbook.model";

    public static void check(PermissionChecker permissionChecker, long groupId, String
actionId)
        throws PortalException {

        if (!contains(permissionChecker, groupId, actionId)) {
            throw new PrincipalException();
        }
    }

    public static boolean contains(PermissionChecker permissionChecker, long groupId,
String actionId) {

        return permissionChecker.hasPermission(groupId, RESOURCE_NAME, groupId, actionId
);
    }
}
```

Ejemplo de clase que comprueba los permisos particulares sobre una de las clases del modelo

```
public class GuestbookPermission {

    public static void check(PermissionChecker permissionChecker, long guestbookId,
String actionId)
        throws PortalException, SystemException {

        if (!contains(permissionChecker, guestbookId, actionId)) {
            throw new PrincipalException();
        }
    }

    public static boolean contains(PermissionChecker permissionChecker, long guestbookId,
String actionId)
        throws PortalException, SystemException {

        Guestbook guestbook = GuestbookLocalServiceUtil.getGuestbook(guestbookId);

        return permissionChecker.hasPermission(guestbook.getGroupId(), Guestbook.class
.getName(),
            guestbook.getGuestbookId(), actionId);

    }
}
```

Una vez definidos los Helper, se pueden emplear en las JSPs

Ejemplo de validacion de permisos de tipo VIEW sobre

```
if (GuestbookPermission.contains(permissionChecker, curGuestbook.getGuestbookId(),
"VIEW")) {

}
```

Acceso a la pantalla de configuracion de Permisos de una entidad

El API de JSPs ofrece una etiqueta que permite el acceso a la pagina de configuracion de los permisos de una entidad

```
<liferay-security:permissionsURL
    modelResource="<%=Entry.class.getName()%>"
    modelResourceDescription="<%=entry.getMessage()%>"
    resourcePrimKey="<%=String.valueOf(entry.getEntryId())%>"
    var="permissionsURL" />
```

Selector de permisos

En la UI se puede emplear la etiqueta **liferay-ui:input-permissions** para mostrar un selector de permisos al dar de alta los recursos.

```
<liferay-ui:input-permissions
    modelName="<%= JournalFolder.class.getName() %>"
/>
```

Assets

API de Liferay, que permite incluir características propias de Liferay como etiquetas (tags), categorías (categories) o comentarios a las entidades que maneja el portal.

Todas las entidades que maneja Liferay en su Core, son Assets.

Para poder crear una entidad de tipo Asset, se precisa crear dicha entidad con el API de Service Builder.

Para convertir una entidad persistente gestionada por Service Builder, se ha de hacer referencia a `AssetEntry`

Contenido de Service.xml

```
<!DOCTYPE service-builder PUBLIC "-//Liferay//DTD Service Builder 6.2.0//EN"
"http://www.liferay.com/dtd/liferay-service-builder_6_2_0.dtd">
<service-builder package-path="com.liferay.biblioteca">
    <namespace>Biblioteca</namespace>
    <entity name="Libro" local-service="true" remote-service="false">
        .
        .
        .
        <reference package-path="com.liferay.portlet.asset" entity="AssetEntry" />
    </entity>
</service-builder>
```

Una vez declarada la entidad, al generar la capa de servicios, se tendrá una clase **LibroLocalServiceImpl** que heredará de **AssetEntryLocalService**.

Actualizacion/Insercion de un registro de tipo Asset

Tanto para la actualizacion como para la insercion de un nuevo registro, se ha de invocar el método de **AssetEntryLocalService** llamado **updateEntry**.

Si se inserta, en el metodo **add** de **LibroLocalServiceImpl**, primero se ha de insertar la instancia de tipo Entidad (Libro) y luego invocar a **updateEntry**

Si se actualiza, en el método **update** de **LibroLocalServiceImpl**, primero se invoca a **super.update** y luego a **updateEntry**.

*Firma del método **updateEntry***

```
AssetEntry updateEntry(  
    long userId, long groupId, Date createDate, Date modifiedDate,  
    String className, long classPK, String classUuid, long classTypeId,  
    long[] categoryIds, String[] tagNames, boolean visible,  
    Date startDate, Date endDate, Date expirationDate, String mimeType,  
    String title, String description, String summary, String url,  
    String layoutUuid, int height, int width, Integer priority,  
    boolean sync)  
throws PortalException, SystemException
```

Los parametros que recibe el método son los siguientes

- **userId** → Identificador del usuario que actualiza el contenido.

```
serviceContext.getUserId();
```

- **groupId** → Ambito del contenido, si no tiene ambito, algo raro, establecer en 0.

```
serviceContext.getScopeGroupId();
```

- **createDate** → Fecha de creación del contenido.
- **modifiedDate** → Fecha de modificacion de lcontenido.
- **className** → Nombre de la clase de la entidad. [NombreDeLaClase].class.getName().
- **classPK** → Identificador de la instancia de la entidad. Habitualmente la primary key de la tabla donde se persiste.
- **classUuid** → Segundo identificador, que garantiza la universalidad de la instancia, es especialmente empleada al exportar contenido entre portales.
- **classTypeId** → Identificador de la variante del tipo entidad, si es que las acepta, en otro caso su valor es 0.

- **categoryIds** → Identificadores de las categorías a las que ha sido asociado la instancia. El asset framework se encarga de almacenarlo.

```
serviceContext.getAssetCategoryIds();
```

- **assetTagNames** → Etiquetas asociadas a la instancia. El asset framework se encarga de almacenarlo.

```
serviceContext.getAssetTagNames()
```

- **visible** → booleano que indica que la entidad ha sido aprobada.
- **startDate** → Fecha de publicacion. Puede ser null.
- **endDate** → Fecha de ocultacion. Puede ser null.
- **expirationDate** → Ultima fecha posible para la ocultación. Puede ser null.
- **mimetype** → Tipo de contenido.
- **title** → Nombre de la instancia de la entidad.
- **description** → Descripcion de la instancia de la entidad.
- **summary** → Descripcion corta de la instancia de la entidad.
- **url** → URL opcional con la que asociar a la instancia de la entidad. Puede ser null.
- **layoutUuid** → Identificador del layout para visualizar la instancia. Puede ser null.
- **height** → Puede valer 0.
- **width** → Puede valer 0.
- **priority** → Especifica la importancia de la instancia frente al resto de instancias, los valores enteros mas pequeños, tienen mas prioridad que los grandes. Puede ser null.
- **sync** → Puese ser false.

Un ejemplo de uso podría ser

Ejemplo de actualización de entidad de tipo Asset

```
long classTypeId = 0;
boolean visible = true;
Date startDate = null;
Date endDate = null;
Date expirationDate = null;
String mimeType = ContentTypes.TEXT_HTML;
String title = insult.getInsultString();
String description = insult.getInsultString();
String summary = insult.getInsultString();
String url = null;
String layoutUuid = null;
int height = 0;
int width = 0;
Integer priority = null;
boolean sync = false;

assetEntryLocalService.updateEntry(
    userId, groupId, insult.getCreateDate(),
    insult.getModifiedDate(), Insult.class.getName(),
    insult.getInsultId(), insult.getUuid(), classTypeId,
    serviceContext.getAssetCategoryIds(),
    serviceContext.getAssetTagNames(), visible, startDate, endDate,
    expirationDate, mimeType, title, description, summary, url,
    layoutUuid, height, width, priority, sync);

//Indexación del contenido
Indexer indexer = IndexerRegistryUtil.nullSafeGetIndexer(Insult.class);
indexer.reindex(insult);
```

Borrado de un registro de tipo Asset

Si se borra, en el metodo **delete** de **LibroLocalServiceImpl**, primero se ha de borrar la instancia de tipo Entidad (Libro) y luego invocar a **deleteEntry**

Ejemplo de borrado de entidad de tipo Asset

```
assetEntryLocalService.deleteEntry(
    Insult.class.getName(), insult.getInsultId());

Indexer indexer = IndexerRegistryUtil.nullSafeGetIndexer(Insult.class);
indexer.delete(insult);
```


Asset Renderer - Gestion de Asset con herramientas genericas (Asset Publisher Portlet y Asset Renderer portlet)

Dado que todas las entidades de tipo Asset, tienen una serie de campos comunes, como title, description y summary, en Liferay se proporcionan Portlets para trabajar con el concepto de Asset, mas allá de si son entradas de Blog, de la Wiki o un contenido personalizado.

Para que Liferay puede procesar un contenido personalizado como lo hace con los contenidos base, hay que crear una serie de recursos.

- Clase que herede de **BaseAssetRenderer** → Esta clase es basicamente un Wrapper sobre la clase Asset, que gestiona permisos de lectura y escritura sobre el Asset y la renderizacion, basicamente un jsp, que emplearán los portlet Asset Publisher Portlet y Asset Renderer portlet para mostrar el contenido especifico de este tipo de Asset.

Ejemplo de Clase que hereda de BaseAssetRenderer

```
public class GuestbookAssetRenderer extends BaseAssetRenderer {

    private Guestbook _guestbook;

    public GuestbookAssetRenderer (Guestbook guestbook) {
        _guestbook = guestbook;
    }

    @Override
    public boolean hasEditPermission(PermissionChecker permissionChecker) {
        long guestbookId = _guestbook.getGuestbookId();
        boolean contains = false;
        try {
            contains = GuestbookPermission.contains(permissionChecker,
guestbookId, ActionKeys.UPDATE);
        } catch (PortalException pe) {
            _log.error(pe.getLocalizedMessage());
        } catch (SystemException se) {
            _log.error(se.getLocalizedMessage());
        }
        return contains;
    }

    @Override
    public boolean hasViewPermission(PermissionChecker permissionChecker) {
        long guestbookId = _guestbook.getGuestbookId();
        boolean contains = false;
```

```

        try {
            contains = GuestbookPermission.contains(permissionChecker,
guestbookId, ActionKeys.VIEW);
        } catch (PortalException pe) {
            _log.error(pe.getLocalizedMessage());
        } catch (SystemException se) {
            _log.error(se.getLocalizedMessage());
        }
        return contains;
    }

    @Override
    public String getClassName() {
        return Guestbook.class.getName();
    }

    @Override
    public long getClassPK() {
        return _guestbook.getGuestbookId();
    }

    @Override
    public long getGroupId() {
        return _guestbook.getGroupId();
    }

    @Override
    public String getSummary(Locale locale) {
        return "Name: " + _guestbook.getName();
    }

    @Override
    public String getTitle(Locale locale) {
        return _guestbook.getName();
    }

    @Override
    public long getUserId() {
        return _guestbook.getUserId();
    }

    @Override
    public String getUsername() {
        return _guestbook.getUsername();
    }

    @Override
    public String getUuid() {

```

```

        return _guestbook.getUuid();
    }

    @Override
    public String render(RenderRequest renderRequest, RenderResponse renderResponse,
        String template) throws Exception {
        if (template.equals(TEMPLATE_FULL_CONTENT)) {
            renderRequest.setAttribute("gb_guestbook", _guestbook);
            return "/html/guestbookadmin/" + template + ".jsp";
        }
        else {
            return null;
        }
    }

    private Log _log;

    @Override
    protected String getIconPath(ThemeDisplay themeDisplay) {
        return themeDisplay.getURLPortal() + "/guestbook-portlet/guestbook.png";
    }
}

```

NOTE

La constante **TEMPLATE_FULL_CONTENT** esta definida en la clase **BaseAssetRenderer** y define un nombre generico para las plantillas que sirven para representar todo el contenido de un Asset. En esta plantilla habrá que recoger el parametro de la request **gb_guestbook**

NOTE

A la hora de representenar contenido de un Asset en la vista, conviene asegurarse de que el contenido de los campos no contiene codigo malicioso, para ello el API de los **Model** generados con ServiceBuilder proporciona un método **toEscapedModel**, que escapa el contenido

```
guestbook.toEscapedModel();
```

- Clase que herede de **BaseAssetRendererFactory** → Clase que implementa el patrón factoria para la instanciación de la anterior.

Ejemplo de Clase que hereda de BaseAssetRenderer

```
public class GuestbookAssetRendererFactory extends BaseAssetRendererFactory {
    public static final String CLASS_NAME = Guestbook.class.getName();
    public static final String TYPE = "guestbook";

    @Override
    public AssetRenderer getAssetRenderer(long classPK, int type) throws PortalException,
    SystemException {
        Guestbook guestbook = GuestbookLocalServiceUtil.getGuestbook(classPK);
        return new GuestbookAssetRenderer(guestbook);
    }

    @Override
    public String getClassName() {
        return CLASS_NAME;
    }

    @Override
    public String getType() {
        return TYPE;
    }

    @Override
    public boolean hasPermission(PermissionChecker permissionChecker, long classPK,
    String actionId) throws Exception {
        return GuestbookPermission.contains(permissionChecker, classPK, actionId);
    }

    @Override
    public boolean isLinkable() {
        return _LINKABLE;
    }

    private static final boolean _LINKABLE = true;
}
```

Una vez definidas las dos tipologías, se ha de registrar la Factoría en el fichero **/WEB-INF/liferay-portlet.xml**

```
<asset-renderer-factory>
com.liferay.docs.guestbook.asset.GuestbookAssetRendererFactory</asset-renderer-factory>
```

NOTE

La anterior configuración se sitúa entre las etiquetas `<control-panel-entry-weight>` y `<header-portlet-css>`

Contenido Relacionado (Asset Related)

Permite el enlazado a otros contenidos permitiendo al usuario una navegacion por los contenidos mas comoda, sin tener que realizar búsquedas.

Para su gestion se necesita que la entidad sea un Asset y además incluir la referencia a **AssetLink** en la entidad de Service Builder

Contenido de Service.xml

```
<!DOCTYPE service-builder PUBLIC "-//Liferay//DTD Service Builder 6.2.0//EN"
"http://www.liferay.com/dtd/liferay-service-builder_6_2_0.dtd">
<service-builder package-path="com.liferay.biblioteca">
    <namespace>Biblioteca</namespace>
    <entity name="Libro" local-service="true" remote-service="false">
        .
        .
        .
        <reference package-path="com.liferay.portlet.asset" entity="AssetEntry" />
        <reference package-path="com.liferay.portlet.asset" entity="AssetLink" />
    </entity>
</service-builder>
```

Añadir / Actualizar

En los métodos de add y update de la implementacion del Servicio **-LocalServiceImpl**, se ha de invocar la actualización de los enlaces, para lo cual y gracias a la referencia anterior, el API ha generado un atributo **assetLinkLocalService** que permite gestionar los contenidos relacionados.

Actualización de enlaces a contenidos relacionados

```
assetLinkLocalService.updateLinks(
    userId,
    assetEntry.getEntryId(),
    serviceContext.getAssetLinkEntryIds(),
    AssetLinkConstants.TYPE_RELATED);
```

El método **updateLinks** necesita

- **UserId** → Identificador de usuario que crea los enlaces
- **EntryId** → El identificador del contenido principal que posee los enlaces
- **AssetLinkEntryIds** → Los identificadores de los contenidos secundarios a los que se está enlazando el contenido principal
- **Type** → Tipo de enlazado

Borrado

Y para el método **delete**

Borrado de enlaces a contenidos relacionados

```
AssetEntry assetEntry = assetEntryLocalService.fetchEntry(Insult.class.getName(),
    insultId);

assetLinkLocalService.deleteLinks(assetEntry.getEntryId());
```

Modificacion desde UI

Para poder incluir los enlaces desde la interface de usuario, se emplea el componente **liferay-ui:input-asset-links**, que debe estar dentro de los **au:fieldset**

Interface de usuario para la inclusión de enlaces a contenidos

```
<liferay-ui:panel defaultState="closed" extended="<%= false %>"
    id="insultAssetLinksPanel" persistState="<%= true %>" title="related-assets">
    <au:fieldset>
        <liferay-ui:input-asset-links
            className="<%= Insult.class.getName() %>"
            classPK="<%= insultId %>"
        />
    </au:fieldset>
</liferay-ui:panel>
```

Para que el menu de añadir un nuevo link, aparezca un nombre legible de la entidad, y no el nombre canonico, se ha de incluir en el fichero **src/content/Language.properties** la siguiente propiedad

```
model.resource.com.liferay.docs.insult.model.Insult=Insult
```

Visualizacion desde UI

Para la visualización de los links, se ha de tener implementada la renderizacion del Asset para la entidad.

Se han de recuperar los datos del Asset

```
<%
long insultId = ParamUtil.getLong(renderRequest, "insultId");
Insult ins = InsultLocalServiceUtil.getInsult(insultId);
AssetEntry assetEntry = AssetEntryLocalServiceUtil.getEntry(Insult.class.getName(),
ins.getInsultId());
%>
```

Y mostrarlos con la etiqueta **liferay-ui:asset-links**

```
<liferay-ui:asset-links
    assetEntryId="<%= (assetEntry != null) ? assetEntry.getEntryId() : 0%>"
    className="<%= Insult.class.getName() %>"
    classPK="<%= ins.getInsultId() %>" />
```

Categorías y Tags

Una de las características de los Asset es que se les puede categorizar

El API de JSP proporciona un par de etiquetas particulares para poder visualizar errores con respecto a las Categorías y las Etiquetas.

- **liferay-ui:asset-categories-error**
- **liferay-ui:asset-tags-error**

Además se puede emplear la etiqueta básica **lui:input** para definir los componentes de formulario para establecer los valores de las etiquetas y las categorías.

Parte de JSP que permite mostrar en un panel colapsable los campos de formulario para establecer las etiquetas y las categorías

```
<liferay-ui:asset-categories-error />
<liferay-ui:asset-tags-error />
<liferay-ui:panel defaultState="closed" extended="<%= false %>"
id="insultsCategorizationPanel" persistState="<%= true %>" title="categorization">
    <lui:fieldset>
        <lui:input name="categories" type="assetCategories" />
        <lui:input name="tags" type="assetTags" />
    </lui:fieldset>
</liferay-ui:panel>
```

Para visualizar los valores asignados a estos campos se proporcionan las etiquetas

- **liferay-ui:asset-categories-summary**

- liferay-ui:asset-tags-summary

Parte de JSP que muestra los valores de las etiquetas y las categorias

```
<label>Categories</label>
<liferay-ui:asset-categories-summary
  className="<%= insult.getClass().getName() %>"
  classPK="<%= insult.getPrimaryKey() %>"
/>

<label>Tags</label>
<liferay-ui:asset-tags-summary
  className="<%= insult.getClass().getName() %>"
  classPK="<%= insult.getPrimaryKey() %>"
/>
```

Valoraciones

Para añadir las valoraciones a una entidad, basta con definirla como **Asset**.

Si se define el **AssetRenderer**, en la visualizacion **FULL_CONTENT_VIEW**, se muestran las valoraciones, pudiendo establecerlas.

Para incluirlas en un Portlet propio, basta con añadir la etiqueta **<liferay-ui:ratings**

```
<liferay-ui:ratings className="<%=Insult.class.getName()%>"
classPK="<%=ins.getInsultId()%>" type="stars" />
```

Comentarios

Se puede añadir a la vista la etiqueta **liferay-ui:discussion** que permite añadir comentarios a los contenidos.


```

<liferay-ui:panel-container extended="<%=false%>"
    id="insultCommentsPanelContainer" persistState="<%=true%>">

    <liferay-ui:panel collapsible="<%=true%>" extended="<%=true%>"
        id="insultCommentsPanel" persistState="<%=true%>"
        title='<%=LanguageUtil.get(pageContext, "comments")%>'>

        <portlet:actionURL name="invokeTaglibDiscussion" var="discussionURL" />

        <%
String currentUrl = PortalUtil.getCurrentURL(request);
%>

        <liferay-ui:discussion className="<%=Insult.class.getName()%>"
            classPK="<%=ins.getInsultId()%>"
            formAction="<%=discussionURL%>" formName="fm2"
            ratingsEnabled="<%=true%>" redirect="<%=currentUrl%>"
            subject="<%=ins.getInsultString()%>"
            userId="<%=ins.getUserId()%>" />
    </liferay-ui:panel>
</liferay-ui:panel-container>

```

Se han de añadir dos URLs

- formAction → La URL que permite realizar el alta del comentario, es siempre la misma

```

<portlet:actionURL name="invokeTaglibDiscussion" var="discussionURL" />

```

- redirect → La URL a la que se redirecciona una vez procesada el alta del comentario.

```

<%
String currentUrl = PortalUtil.getCurrentURL(request);
%>

```

Busquedas e Indices

Liferay permite la definicion de indices sobre los contenidos (Asset) generados, de tal forma que puedan participar de las funcionalidades de busqueda rapida, que presentan mayor rendimiento que las busquedas directas sobre la base de datos.

Liferay por debajo emplea Lucene para la gestion de los indices.

Para poder indexar los contenidos (Asset), se han de seguir los siguientes pasos

- Crear una clase que herede de **BaseIndexer**

Ejemplo de Clase que hereda de BaseIndexer

```
public class EntryIndexer extends BaseIndexer {

    public static final String[] CLASS_NAMES = { Entry.class.getName() };
    public static final String PORTLET_ID = "guestbook-portlet";

    public EntryIndexer() {
        setPermissionAware(true);
    }

    @Override
    public String[] getClassNames() {
        return CLASS_NAMES;
    }

    @Override
    public String getPortletId() {
        return PORTLET_ID;
    }

    @Override
    public boolean hasPermission(PermissionChecker permissionChecker, String
entryClassName, long entryClassPK, String actionId) throws Exception {
        return GuestbookPermission.contains(permissionChecker, entryClassPK
,ActionKeys.VIEW);
    }

    @Override
    protected void doDelete(Object obj) throws Exception {
        Entry entry = (Entry)obj;
        deleteDocument(entry.getCompanyId(), entry.getEntryId());
    }

    @Override
    protected Document doGetDocument(Object obj) throws Exception {
        Entry entry = (Entry)obj;
        Document document = getBaseModelDocument(PORTLET_ID, entry);
        document.addDate(Field.MODIFIED_DATE, entry.getModifiedDate());
        document.addText(Field.CONTENT, entry.getMessage());
        document.addText(Field.TITLE, entry.getName());
        document.addText("email", entry.getEmail());
        document.addKeyword(Field.GROUP_ID, getSiteGroupId(entry.getGroupId()));
        document.addKeyword(Field.SCOPE_GROUP_ID, entry.getGroupId());

        return document;
    }
}
```

```

    }

    @Override
    protected Summary doGetSummary(Document document, Locale locale, String snippet,
PortletURL portletURL) throws Exception {
        Summary summary = createSummary(document);
        summary.setMaxContentLength(200);
        return summary;
    }

    @Override
    protected void doReindex(Object obj) throws Exception {
        Entry entry = (Entry)obj;
        Document document = getDocument(entry);
        SearchEngineUtil.updateDocument(getSearchEngineId(), entry.getCompanyId(
), document);
    }

    @Override
    protected void doReindex(String className, long classPK) throws Exception {
        Entry entry = EntryLocalServiceUtil.getEntry(classPK);
        doReindex(entry);
    }

    @Override
    protected void doReindex(String[] ids) throws Exception {
        long companyId = GetterUtil.getLong(ids[0]);
        reindexEntries(companyId);
    }

    @Override
    protected String getPortletId(SearchContext searchContext) {
        return PORTLET_ID;
    }

    protected void reindexEntries(long companyId) throws PortalException
, SystemException {
        final Collection<Document> documents = new ArrayList<Document>();
        ActionableDynamicQuery actionableDynamicQuery = new
EntryActionableDynamicQuery() {

            @Override
            protected void addCriteria(DynamicQuery dynamicQuery) {
            }

            @Override
            protected void performAction(Object object) throws
PortalException {

```

```

        Entry entry = (Entry) object;
        Document document = getDocument(entry);
        documents.add(document);
    }
};

actionableDynamicQuery.setCompanyId(companyId);
actionableDynamicQuery.performActions();
SearchEngineUtil.updateDocuments(getSearchEngineId(), companyId,
documents);
}
}

```

- Registrar dicha clase en el fichero **liferay-portlet.xml**

```

<liferay-portlet-app>
    <portlet>
        ...
        <indexer-class>com.liferay.docs.insult.search.InsultIndexer</indexer-
class>
        ...

```

- Incluir en los métodos de add, update y delete del **-LocalServiceImp** la logica relacionada con el indice.

add

```

Indexer indexer = IndexerRegistryUtil.nullSafeGetIndexer(Insult.class);
indexer.reindex(insult);

```

update

```

Indexer indexer = IndexerRegistryUtil.nullSafeGetIndexer(Insult.class);
indexer.reindex(insult);

```

delete

```

Indexer indexer = IndexerRegistryUtil.nullSafeGetIndexer(Insult.class);
indexer.delete(insult);

```

- Configurar la vista para que haga uso del indice.

Para ello, se emplea el Portlet ya existente **Busqueda Web**, donde se deberá de añadir en la configuracion avanzada, el nuevo tipo de datos añadiendolo al JSON de configuracion.

```
"values": [  
    "com.liferay.portal.model.User",  
    "com.liferay.portlet.bookmarks.model.BookmarksEntry",  
    "com.liferay.portlet.bookmarks.model.BookmarksFolder",  
    "com.liferay.portlet.blogs.model.BlogsEntry",  
    "com.liferay.portlet.documentlibrary.model.DLFileEntry",  
    "com.liferay.portlet.documentlibrary.model.DLFolder",  
    "com.liferay.portlet.journal.model.JournalArticle",  
    "com.liferay.portlet.journal.model.JournalFolder",  
    "com.liferay.portlet.messageboards.model.MBMessage",  
    "com.liferay.portlet.wiki.model.WikiPage",  
    "com.liferay.docs.insult.model.Insult"  
],
```

API

Las clases más importantes para gestionar la funcionalidad de búsqueda son

- `com.liferay.portal.kernel.search.SearchContext` → Permite la definición de un contexto de búsqueda.
- `com.liferay.portal.kernel.search.SearchContextFactory` → Permite la obtención de `SearchContext`.
- `com.liferay.portal.kernel.search.Indexer` → Clase que modela el índice.
- `com.liferay.portal.kernel.search.IndexerRegistryUtil` → Clase que proporciona el acceso al índice.
- `com.liferay.portal.kernel.search.BaseIndexer` → Clase a heredar para implementar las funcionalidades de indexado.
- `com.liferay.portal.kernel.search.SearchEngineUtil` → Permite buscar documentos en el índice para su actualización.
- `com.liferay.portal.kernel.search.Hits` → Documentos que cumplen los criterios de búsqueda.
- `com.liferay.portal.kernel.search.Document` → Clase se define la información indexada.

Implementación de búsqueda

De no emplear el Portlet **Busqueda Web**, se puede implementar la búsqueda en el índice, para ello hay que definir un **Contexto de búsqueda**, donde **keywords** son las palabras por las que se desea buscar.

```
SearchContext searchContext = SearchContextFactory.getInstance(request);  
searchContext.setKeywords(keywords);  
searchContext.setAttribute("paginationType", "more");  
searchContext.setStart(0);  
searchContext.setEnd(10);
```

Una vez definido el **Contexto de búsqueda**, se emplea junto con el índice.

```
Indexer indexer = IndexerRegistryUtil.getIndexer("MyEntity");

try {
    Hits hits = indexer.search(searchContext);

    List<Document> docs = hits.toList();
}
catch (SearchException se) {
    // handle search exception
}
```

Web Service

Cuando una entidad se define con service builder como remota, se generan las interfaces y clases necesarias para disponer de acceso via SOAP y JSON.

En caso de disponer de implementación local y remota, se han de implementar las clases **_LocalServiceImpl** y **_ServiceImpl**.

La recomendación, es invocar a la **Local** desde la **Remota** incluyendo en la remota la verificación de la seguridad.

```
@Override
public JournalArticle addArticle(...) throws PortalException {

    JournalFolderPermission.check(getPermissionChecker(), groupId, folderId, ActionKeys
.ADD_ARTICLE);

    return journalArticleLocalService.addArticle(...);
}
```

Sin más, se disponen de los servicios JSON, para los SOAP, habrá que seguir algún paso extra.

SOAP

El listado de servicios SOAP disponibles en el Portal se puede obtener en la url

```
http://[host]:[port]/api/axis
```

Algunos ejemplos de rutas a servicios SOAP

```
http://localhost:8080/api/axis/Portal_CompanyService?wsdl  
http://localhost:8080/api/axis/Portal_UserService?wsdl  
http://localhost:8080/api/axis/Portal_UserGroupService?wsdl
```

Para crear uno propio, basta con definir una entidad con service builder y lanzar sobre el proyecto **-service**, la tarea **buildWSDD**.

Una vez desplegado el WSDD, el servicio SOAP se encuentra disponible en la ruta

```
http://[host]:[port]/<nombre de despliegue de la aplicacion>/api/axis
```

NOTE

nombre de despliegue de la aplicacion sera el nombre de la app concatenando **-service**

Como ejemplo posible

```
http://localhost:8080/foo-service/api/axis
```

JSON

El listado de servicios JSON disponibles en el Portal se puede obtener en la url

```
http://[host]:[port]/api/jsonws/
```

Los servicios JSON se publican directamente al crear un servicio remoto y lanzar **BuildService** sobre las operaciones definidas en ***ServiceImpl**

Hook

Permiten **engancharse** y sobrescribir propiedades y funcionalidades del portal.

Los hook, se configuran en el fichero **liferay-hook.xml**

Los tipos de Hook posibles son * De propiedades del Portal. * De Idioma. * De Action de Struts. * De Vistas (JSP). * De Servicios. * De Indexer Post Procesor.

Los Hook, permiten realizar una tarea semejante a la que permiten los plugin de tipo Ext.

Los Hook son mas sencillos y mas fáciles de implementar que los Ext, a parte de ser la forma recomendada de extender o modificar el comportamiento del portal son los Hook, solamente se emplearán los Ext, cuando los Hook no permitan la modificación pretendida.

Los proyectos de tipo Hook, como el resto se pueden crear empleando ANT o MAVEN.

Hook de propiedades

Reemplaza propiedades definidas en el **portal.properties** del portal.

Entre otras, se pueden definir escuchadores de eventos para

- Login de usuario
- Logout de usuario
- Ejecución de un servicio
- Inicio de aplicación
- Session de usuario

Se configura con la propiedad **<portal-properties>** del fichero **liferay-hook.xml** **<portal-properties>**

No todas las propiedades pueden ser sobreescritas, por ejemplo no podriamos sobrecribir las propiedades de la conexión a base de datos, eso hay que hacerlo con Ext.

El listado de las propiedades que se pueden modificar se encuentra en el fichero https://docs.liferay.com/portal/6.2/definitions/liferay-hook_6_2_0.dtd.html, que es el esquema que sigue el fichero **liferay-hook.xml**

Las propiedades del **portal.properties** asociadas a estos eventos son

- Login
 - login.events.pre
 - login.events.post
- Logout
 - logout.events.pre
 - logout.events.post
- Inicio de aplicación
 - application.startup.events
- Ejecución de servicio
 - servlet.service.events.pre
 - servlet.service.events.post

- Session de usuario
 - `servlet.session.destroy.events`
 - `servlet.session.create.events`

Estas propiedades admiten varios valores separados por comas, siendo los valores que se les han de asociar, los nombres de las clases que han de manejar el evento producido.

Los eventos de Login, Logout y de Servicio, han de ser tratados por clases que extiendan de **`com.liferay.portal.kernel.events.Action`**

Los eventos de Inicio de Aplicación, han de ser tratados por clases que extiendan de **`com.liferay.portal.kernel.events.SimpleAction`**

Los eventos de Session, por clases que extiendan **`com.liferay.portal.kernel.events.SessionAction`**

Hook de Idiomas

Es un caso particular de Hook de propiedades, permite reemplazar y extender los ficheros de localización, se configura con la propiedad **`<language-properties>`** del fichero **`liferay-hook.xml`**

Indicando un fichero de propiedades con el código idiomático y las traducciones correspondientes para las propiedades deseadas.

Se puede obtener un fichero **`Language.properties`** de referencia en la carpeta **`content`** del fichero **`portal-impl.jar`**, que esta en la carpeta **`WEB-INF/lib`** del proyecto **`ROOT`** (el portal)

Hook de Actions de Struts

En Liferay existen muchas funcionalidades accesibles creadas a través de Action de Struts. A través de los Hook, se pueden modificar estos Action o añadir nuevos.

La configuración de estos Action se encuentra en el fichero **`/portal-web/docroot/WEB-INF/struts-config.xml`**

Para modificar un Action ya existente, se ha de definir una nueva entrada en el fichero **`liferay-hook.xml`**, indicando:

- El path del Action
- La nueva clase que lo implemente

```
<hook>
  <struts-action>
    <struts-action-path>/login/login</struts-action-path>
    <struts-action-impl>com.ejemplo.actions.CustomLoginAction</struts-action-impl>
  </struts-action>
</hook>
```

Para añadir un nuevo Action a parte de realizar lo anterior en el fichero **liferay-hook.xml**, se ha de incluir la nueva URL como path autorizado, indicando en el fichero **portal.properties** lo siguiente **auth.public.paths=/portal/miejemplo**

El fichero **portal.properties** deberá ser declarado en el **liferay-hook.xml**

```
<hook>
  <portal-properties>portal.properties</portal-properties>
</hook>
```

Las url de los Action, se compondrán de la siguiente manera <http://localhost:8080/c/portal/miejemplo>

Las url de los nuevos Action, deberán de cumplir el patrón **/portal/***

Para sobrescribir Struts ya existentes la nueva clase ha de heredar de **BaseStrutsPortletAction**, esta herencia permite implementar el siguiente método

```
public void processAction(StrutsPortletAction originalStrutsPortletAction,
    PortletConfig portletConfig, ActionRequest actionRequest,
    ActionResponse actionResponse) throws Exception {}
```

Para nuevos Action la nueva clase ha de heredar de **BaseStrutsAction**, esta herencia permite implementar el siguiente método.

```
@Override
public String execute(HttpServletRequest request, HttpServletResponse response)
throws Exception {
    return "/portal/miVista.jsp2;
}
```

Hook de JSPs

Permiten reemplazar una JSP existente en el portal, basta con copiar el JSP ya existente en el portal, al proyecto y modificar las características necesarias, indicando en el XML la ruta donde se sitúa en nuevo JSP.

NOTE

El editor del XML, permite referenciar a los XML y realiza la copia fácilmente.

Si únicamente se quiere añadir un trozo de html al jsp existente, no es necesario copiar y pegar todo el código, se puede importar con la etiqueta

```
<liferay-util:include page="/html/portlet/login/login.portal.jsp" />
```

Se puede manipular la JSP original, para añadir o eliminar alguna parte de forma programática, a partir de la etiqueta **<liferay-util:buffer>**, que crea un String con el JSP de partida, pudiendo realizar sustituciones, eliminaciones a modo de substring.

```
<liferay-util:buffer var="html">
  <liferay-util:include page= "/html/portlet/blogs/search.portal.jsp" />
</liferay-util:buffer>

<% html = StringUtil.add( html, "Didn't find what you were looking for? " +
  "Refine your search and try again!", "\n");
%>

<%= html %>
```

Hook de Servicio

Toda la funcionalidad de Liferay esta expuesta por su capa de servicios, los Hook de servicios, permiten cambiar el comportamiento de estos servicios, indicando una nueva implementación.

El procedimiento se basa en extender las clases que proporciona el API, denominadas clases Wrapper, en lugar de sobrescribir las clases que representan el servicio directamente.

Un caso concreto sería el servicio que permite manejar los usuarios del portal.

Para modificar su funcionalidad, deberíamos extender la clase **UserLocalServiceWrapper** y no la interface **UserLocalService** y luego añadir la siguiente entrada en el fichero **liferay-hook.xml**

```
<service>
  <service-type>com.liferay.portal.service.UserLocalService</service-type>
  <service-impl>MyUserLocalService</service-impl>
</service>
```

Hook de Indexador

Permite modificar resultados de búsquedas, índices y consultas del portal.

Por ejemplo, por defecto cuando se añaden usuarios al portal, solo se indexan el nombre y el apellido, pero campos como el titulo, no, luego búsquedas por el campo titulo, no serán resueltas.

Los Hook de indexación, permiten ampliar los campos indexados para cada entidad.

Se ha de añadir la siguiente configuracion al fichero **liferay-hook.xml**

```
<indexer-post-processor>
  <indexer-class-name>com.liferay.portal.model.User</indexer-class-name>
  <indexer-post-processor-impl>
    com.ejemplo.MiIndexerPostProcesor
  </indexer-post-processor-impl>
</indexer-post-processor>
```

Donde se indica

- Cual es la entidad sobre la que se va a buscar.
- Y la clase que implemente el post-procesamiento de la búsqueda que extienda de **BaseIndexerPostProcessor**.

```
@Override
public void postProcessDocument(Document document, Object obj) throws Exception {
    User userEntity = (User) obj;
    String indexerUserTitle = userEntity.getJobTitle();
    if (indexerUserTitle.length() > 0)
        document.addText(Field.TITLE, indexerUserTitle);
}
```

Temas

Un tema es un plugin que permite modificar el aspecto de las paginas que forman el portal por completo, siendo difícil saber que el portal esta hecho con liferay.

La forma de crear nuevos temas, se basa en la herencia de otros, modificando solo aquellos aspectos que se quieren personalizar.

La personalización de los Temas, viene marcada por cuatro tipos de recursos

- CSS
- Plantillas (Freemarker, Velocity o JSP)
- Javascript
- XML

Se pueden instalar desde el propio Portal accediendo al Market de Liferay, o bien creando un Plugin de tipo Tema y desplegandolo.



























Proyecto de tipo Tema

Se pueden crear los proyectos de Temas siguiendo ANT o MAVEN.

En ambos dos casos, se ha de configurar el Tema padre, que será el Tema de donde se extraigan las características por defecto.

Una vez definido el proyecto, se empieza la personalización, que se basa en la sobre-escritura de las propiedades que proporciona el padre.

Tiene la siguiente estructura

- ▼  > Ejemplo-Tema-Burbujas [Liferay 6.2]
 - >  > src/main/resources
 - >  JRE System Library [J2SE-1.5]
 - >  Maven Dependencies
 - >  bin
 - ▼  > src
 - ▼  > main
 - ▼  > webapp
 - >  > css
 - >  > images
 - >  > js
 - >  > templates
 - >  > WEB-INF
 - ▼  target
 - ▼  Ejemplo-Tema-Burbujas-1.0.0-SNAPSHOT
 - >  css
 - >  images
 - >  js
 -  META-INF
 - ▼  templates
 -  init_custom.ftl
 -  navigation.ftl
 -  portal_normal.ftl
 -  portal_pop_up.ftl
 -  portlet.ftl
 - >  WEB-INF

El Tema padre en los proyectos MAVEN, se especifica en el pom.xml del proyecto

```
<properties>
  <liferay.theme.parent>_styled</liferay.theme.parent>
  <liferay.theme.type>ftl</liferay.theme.type>
</properties>
```

La herencia de los temas se puede hacer solo sobre temas que estén instalados en Liferay.

Los posibles valores que pueden tomar, por defecto serán

- classic
- welcome (a partir de la versión 6.2)
- _styled (Maquetación mínima)
- _unstyled (Sin maquetación)

Lo mas habitual es empezar con **_styled**

Se pueden encontrar los ficheros empleados como base en el directorio **<dir instalacion liferay>\tomcat-7.0.42\webapps\ROOT\html\themes**

Para la generacion del tema, habrá que mezclar los ficheros creados en el desarrollo con los ficheros base del tema, para ello se lanza la tarea de Maven **liferay:theme-merge**

Parametrizacion del Tema

Es posible parametrizar los temas, añadiendo los parámetros al fichero **liferay-look-and-feel.xml** con la siguiente etiqueta

```
<settings>
  <setting key="mi-setting" value="mi-valor-1" />
</settings>
```

Siendo estos parámetros accesibles desde las plantillas de Velocity con

```
$theme_display.getThemeSetting("mi-setting")
```

O de freemarker con

```
<#assign myObject = theme_display.getThemeSetting("mi-setting") />
```

La etiqueta setting tiene los siguientes parámetros

- Key: Clave para acceder a la propiedad
- Configurable: Booleano que permite editar la propiedad.
- Type: Tipo de control para representar el parametro (Select, checkbox)
- Options: Valores posibles para el tipo select.
- Value: Valor por defecto para el parametro.

Plantillas

Los temas tienen una serie de plantillas creadas para cada uno de los tipos de recursos que se pueden encontrar en el sitio, estas plantillas están en la carpeta **webapp/templates**

Las hay de

- Freemarker, extensión ftl, la opción recomendada.
- Velocity, extensión vm, actualmente es un motor de plantilla en deshuso, sin mantenimiento y con pocas herramientas para poder emplearlo.
- Jsp, esta posibilidad no está recomendada, por presentar problemas de rendimiento en la generación dinámica de recursos.

Centrándose en Freemarker, se encuentran las siguientes plantillas

- init_custom.ftl: Fichero para la definición de variables a emplear en el resto de plantillas
- init.ftl: Fichero con las variables que define Liferay, que son empleadas en el resto de plantillas.
- navigation.ftl: Menú principal de navegación.
- portal_normal.ftl: Estructura general HTML de todas las páginas del portal.
- portal_pop_up.ftl: Estructura HTML de los popups que nos encontramos dentro del portal.
- portlet.ftl: Estructura HTML de cada uno de los portlets. Básicamente es un section con su encabezado, el menú de opciones de cada portlet y el contenido del mismo.

Se puede acceder a los servicios de Liferay desde las plantillas.

```
<#assign layoutLocalService =  
serviceLocator.findService("com.liferay.portal.service.LayoutLocalService")) />  
  
<#assign url = layoutLocalService.getFriendlyURLLayout(layout.getOwnerId(), "/home")) />
```

En caso de que este servicio pueda lanzar excepciones se puede hacer.

```
<#assign userLocalService =  
serviceLocator.findExceptionSafeService("com.liferay.portal.service.UserLocalService"))  
</>  
  
<#assign user = userLocalService.getUserById(getterUtil.getLong("12345")) />
```

Thumbnail y Favicon

Sobre-escritura de Thumbnail y Favicon del tema.

- Screenshot, se recomienda un ancho de 1024px.
- Thumbnail, se recomienda un ancho de 150px.

Si estamos en uno MAVEN, los ficheros se copian dentro de una carpeta **webapp/images**

Estilos

Para modificar los estilos, se dispone principalmente de dos ficheros css.

- main.css: Importación de todos los estilos, se podría añadir uno nuevo
- custom.css: Fichero vacío, que se importa en ultimo lugar, destinado a incluir los estilos personalizados.

Javascript

En los Temas, existe un único fichero javascript, main.js, que se ejecuta siempre que se carga una página.

Si se desea añadir un nuevo js al Tema, este se ha de añadir al fichero **portal-normal.ftl**, como se haría en cualquier HTML, empleando la variable **javascript_folder**

```
<script src="${javascript_folder}/miFichero.js" type="text/javascript"/>
```

Liferay desde la versión 6.0, viene con un framework javascript Alloy UI.

En el fichero main.js, se encuentran definidos 3 funciones javascript, donde se puede incluir código

- AUI().ready: Se ejecuta cuando se termina de carga la pagina
- Liferay.Portlet.ready: Se ejecuta cada vez que se cargue un portlet de la página actual.
- Liferay.on: Se ejecutará después de que todos los portlets de la página estén cargados.

Por defecto liferay minimiza los ficheros js y css para que sea mas rápido su carga, esto se puede cambiar con propiedades del **portal.properties**

```
javascript.fast.load=false
theme.css.fast.load=false
minifier.enabled=false
```

Esquemas de color

Un esquema de color es una variante de un Tema, en el que se mantienen el Tema original únicamente variando la gama de colores empleados.

Para definirlo basta con crear una nueva entrada en el fichero **WEB-INF/liferay-look-and-feel.xml**, dentro de la etiqueta **theme**, que sea **color-scheme**.

En ese nodo, se hace referencia a aquellos recursos que hayan sido modificados, que podrán ser

- CSS
- Imágenes

Para ello, se tiene los sub-nodos

- Default-cs. Para indicar si es el por defecto (true/false).
- css-class. Nombre de la clase CSS que se añadirá al nodo principal para cambiar estilos y adecuarlos al esquema de color.
- color-scheme-images-path. Para esta propiedad, es interesante el empleo de la variable **\${images-path}**. Suele emplearse valores como **\${images-path}/color_schemes/Defecto**

Por ultimo se han de cargar los CSS desde main.css

```
@import url(color_schemes/Defecto.css);
```

Ext

Plugin de extension que se basa en la generacion de ficheros de configuracion con sufijo **ext**, que el portal ya está preparado para leer.

Sobreescritura Servicio

Los Servicios de Liferay, son Beans de Spring que se definen en un fichero **/META-INF/portal-spring.xml** que se encuentra en **/<tomcat dir>/ROOT/WEB-INF/lib/portal-impl.jar**.

Este fichero forma parte de la configuracion de liferay, porque en el fichero **portal.properties** dentro del mismo jar, hay un parametro de configuracion que lo referencia **spring.configs**

```
spring.configs=\
    META-INF/base-spring.xml,\
    META-INF/hibernate-spring.xml,\
    META-INF/infrastructure-spring.xml,\
    META-INF/management-spring.xml,\
    META-INF/util-spring.xml,\
    META-INF/jpa-spring.xml,\
    META-INF/executor-spring.xml,\
    META-INF/audit-spring.xml,\
    META-INF/cluster-spring.xml,\
    META-INF/editor-spring.xml,\
    META-INF/jcr-spring.xml,\
    META-INF/ldap-spring.xml,\
    META-INF/messaging-core-spring.xml,\
    META-INF/messaging-misc-spring.xml,\
    META-INF/mobile-device-spring.xml,\
    META-INF/notifications-spring.xml,\
    META-INF/poller-spring.xml,\
    META-INF/rules-spring.xml,\
    META-INF/scheduler-spring.xml,\
    META-INF/search-spring.xml,\
    META-INF/workflow-spring.xml,\
    META-INF/counter-spring.xml,\
    META-INF/mail-spring.xml,\
    \
    META-INF/portal-spring.xml,\
    \
    META-INF/portlet-container-spring.xml,\
    META-INF/staging-spring.xml,\
    META-INF/virtual-layouts-spring.xml,\
    META-INF/monitoring-spring.xml,\
    #META-INF/dynamic-data-source-spring.xml,\
    #META-INF/shard-data-source-spring.xml,\
    #META-INF/memcached-spring.xml,\
    \
    classpath*:META-INF/ext-spring.xml
```

El planteamiento que rige **ext**, se basa en esta misma configuracion, como se puede observar, la ultima referencia de esta configuracion, es hacia un fichero **META-INF/ext-spring.xml** que se encuentra en el classpath del portal.

La idea es definir un jar que contenga un fichero con ese nombre, donde se redefina el Servicio deseado

```
. Definicion de una nueva implementacion del en el fichero *META-INF/ext-spring.xml*
----
<?xml version="1.0"?>
```

```
<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-3.0.xsd"
class="bare">http://www.springframework.org/schema/beans/spring-beans-3.0.xsd</a>
default-destroy-method="destroy" default-init-method="afterPropertiesSet">
```

```
<bean id="com.liferay.portal.service.UserLocalService"
class="com.liferay.portal.service.impl.CustomUserLocalServiceImpl" />
</beans>
```

En ese mismo jar basta con añadir una clase *CustomUserLocalServiceImpl* que extienda a la original *UserLocalServiceImpl* y reimplemente los métodos deseados.

```
//include::contenido\desarrollo\CustomFields-ExpandoAPI.adoc[]

//include::contenido\desarrollo\Seguridad.adoc[]

//include::contenido\desarrollo\Workflow.adoc[]
```