



**EBook Gratis**

# APRENDIZAJE

---

## liferay

Free unaffiliated eBook created from  
**Stack Overflow contributors.**

**#liferay**

# Tabla de contenido

Acerca de.....	1
<b>Capítulo 1: Empezando con liferay.....</b>	<b>2</b>
Observaciones.....	2
Versiones.....	2
Examples.....	4
Una instalación básica para desarrollo y pruebas.....	4
<b>Capítulo 2: Comunicación entre portlets.....</b>	<b>6</b>
Introducción.....	6
Observaciones.....	6
Examples.....	6
Usando el parámetro de renderizado público.....	6
Usando sesión de portlet.....	7
Usando la función de eventos.....	7
<b>Capítulo 3: Configurando SSL.....</b>	<b>10</b>
Observaciones.....	10
Examples.....	10
Cómo habilitar SSL en Tomcat y Liferay.....	10
<b>Capítulo 4: Configurar el Administrador de etiquetas de Google (GTM) en el ciclo de vida ú.....</b>	<b>11</b>
Introducción.....	11
Examples.....	11
Usando GTM para configurar eventos GA.....	11
<b>Capítulo 5: Crear un planificador de cuarzo en liferay.....</b>	<b>16</b>
Observaciones.....	16
Examples.....	16
Crear un planificador de cuarzo para mostrar alguna información.....	16
Crear un programador dinámico de cuarzo mediante programación.....	18
<b>Capítulo 6: Depurar el servidor de liferay remoto a través de Eclipse.....</b>	<b>20</b>
Examples.....	20
Depure el servidor de tiempo de vida remoto a través de Eclipse (sin el conector Eclipse d.....	20
<b>Capítulo 7: Desplegando un complemento.....</b>	<b>23</b>

Examples.....	23
Despliegue en Glassfish.....	23
<b>Capítulo 8: Ganchos en Liferay.....</b>	<b>24</b>
Observaciones.....	24
Examples.....	24
JSP Hook.....	24
Puntales ganchos de acción.....	25
Hola usuario "Nombre" con ganchos.....	26
Modelo Listener Hook.....	28
<b>Fondo.....</b>	<b>28</b>
<b>Diferencias.....</b>	<b>28</b>
<b>Ejemplo.....</b>	<b>28</b>
portal.propiedades.....	29
liferay-hook.xml.....	29
<b>Empezando.....</b>	<b>29</b>
<b>Desarrollo del oyente.....</b>	<b>31</b>
<b>Configuración de propiedades.....</b>	<b>33</b>
Explicación.....	34
<b>Construir e implementar.....</b>	<b>34</b>
<b>Déjeme saber si tiene alguna pregunta, comentario, inquietud, etc. ¡Todos los comentarios.....</b>	<b>37</b>
<b>Capítulo 9: Usando consultas SQL dinámicas y personalizadas en Liferay.....</b>	<b>38</b>
Introducción.....	38
Observaciones.....	38
Examples.....	38
Usando la consulta dinámica en Liferay.....	38
<b>Capítulo 10: Usando el servicio web Restful en Liferay.....</b>	<b>40</b>
Examples.....	40
Consumir el servicio JSON de Liferay para solicitudes GET.....	40
<b>Creditos.....</b>	<b>44</b>

---

## Acerca de

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: [liferay](#)

It is an unofficial and free liferay ebook created for educational purposes. All the content is extracted from [Stack Overflow Documentation](#), which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official liferay.

The content is released under Creative Commons BY-SA, and the list of contributors to each chapter are provided in the credits section at the end of this book. Images may be copyright of their respective owners unless otherwise specified. All trademarks and registered trademarks are the property of their respective company owners.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to [info@zzzprojects.com](mailto:info@zzzprojects.com)

# Capítulo 1: Empezando con liferay

## Observaciones

**Liferay Portal CE** es un software de portal construido en Java por Liferay Inc.

**Liferay DXP** (Digital Experience Platform) es una plataforma construida sobre el Portal de Liferay para soluciones digitales, que integra herramientas de análisis de satisfacción del cliente y del usuario y herramientas y rendimientos de calidad de nivel empresarial. Fue conocido como anteriormente conocido como *Liferay Portal EE* .

Desde la versión 7.0 se construye utilizando **OSGi** a través de **Apache Felix** .

## Versiones

Versión	Nombre clave	Tipo de lanzamiento	Fecha de lanzamiento
7.0.1 GA2	Wilberforce	Edición comunitaria	2016-06-10
7.0.10	Wilberforce	Plataforma de experiencia digital	2016-05-04
7.0.0 GA1	Wilberforce	Edición comunitaria	2016-03-31
6.2.3 GA4	Newton	Edición comunitaria	2015-04-17
6.2.2 GA3	Newton	Edición comunitaria	2015-01-15
6.2.1 GA2	Newton	Edición comunitaria	2014-02-28
6.2.10 GA1	Newton	Edición de Empresa	2013-12-03
6.2.0 GA1	Newton	Edición comunitaria	2013-11-01
6.1.2 GA3	Paton	Edición comunitaria	2013-08-23
6.1.30 GA3	Paton	Edición de Empresa	2013-08-16
6.1.1 GA2	Paton	Edición comunitaria	2012-07-31
6.1.20 GA2	Paton	Edición de Empresa	2012-07-31
6.1.10 GA1	Paton	Edición de Empresa	2012-02-15
6.1.0 GA1	Paton	Edición comunitaria	2012-01-01
6.0.12 SP2	Bunyan	Edición de Empresa	2011-11-07
6.0.6	Bunyan	Edición comunitaria	2011-03-04

<b>Versión</b>	<b>Nombre clave</b>	<b>Tipo de lanzamiento</b>	<b>Fecha de lanzamiento</b>
6.0.11 SP1	Bunyan	Edición de Empresa	2011-01-13
5.2 SP5	Agustín	Edición de Empresa	2010-10-20
6.0.10	Bunyan	Edición de Empresa	2010-09-10
6.0.5	Bunyan	Edición comunitaria	2010-08-16
6.0.4	Bunyan	Edición comunitaria	2010-07-23
6.0.3	Bunyan	Edición comunitaria	2010-07-20
6.0.2	Bunyan	Edición comunitaria	2010-06-08
5.2 SP4	Agustín	Edición de Empresa	2010-05-19
6.0.1	Bunyan	Edición comunitaria	2010-04-20
5.1 SP5	Calvin	Edición de Empresa	2010-03-12
6.0.0	Bunyan	Edición comunitaria	2010-03-04
5.2 SP3	Agustín	Edición de Empresa	2010-01-07
5.2 SP2	Agustín	Edición de Empresa	2009-11-17
5.1 SP4	Calvin	Edición de Empresa	2009-10-23
5.2 SP1	Agustín	Edición de Empresa	2009-08-07
5.1 SP3	Calvin	Edición de Empresa	2009-07-20
5.2	Agustín	Edición de Empresa	2009-06-01
5.2.3	Agustín	Edición comunitaria	2009-05-12
5.1 SP2	Calvin	Edición de Empresa	2009-05-12
5.2.2	Agustín	Edición comunitaria	2009-02-26
5.1 SP1	Calvin	Edición de Empresa	2009-02-18
5.2.1	Agustín	Edición comunitaria	2009-02-03
5.2.0	Agustín	Edición comunitaria	2009-01-26
5.1 SP	Calvin	Edición de Empresa	2008-12-16
5.1.2	Calvin	Edición comunitaria	2008-10-03

Versión	Nombre clave	Tipo de lanzamiento	Fecha de lanzamiento
5.1.1	Calvin	Edición comunitaria	2008-08-11
5.1.0	Calvin	Edición comunitaria	2008-07-17
5.0.1 RC	Lutero	Edición comunitaria	2008-04-14
5.0.0 RC	Lutero	Edición comunitaria	2008-04-09

## Examples

### Una instalación básica para desarrollo y pruebas.

Ejecutar el último CE de Liferay es sencillo:

1. Vaya a <https://www.liferay.com/downloads> .
2. Elija un paquete entre los enumerados. Para los principiantes, el paquete Tomcat es una buena opción. Haga clic en "Descargar".



3. Descomprima el paquete de descarga cada vez que lo encuentre conveniente. El directorio descomprimido será el directorio `LIFERAY_HOME` .
4. Para iniciar Liferay, simplemente ejecute el script `LIFERAY_HOME/tomcat-x.xx.xx/bin/startup.sh` ; solo en entornos Windows ejecute el script `LIFERAY_HOME\tomcat-x.xx.xx\bin\startup.bat` .
5. De forma predeterminada, una vez que Liferay esté activo, un navegador abrirá su URL local ( <http://localhost:8080/> ) .
6. Para iniciar sesión, use el correo electrónico `test@liferay.com` y la `test` contraseña.
7. Para detener Liferay, simplemente ejecute el script `LIFERAY_HOME/tomcat-x.xx.xx/bin/shutdown.sh` ; solo en entornos Windows ejecute el script `LIFERAY_HOME\tomcat-x.xx.xx\bin\shutdown.bat` .

Con estos pasos, tendrá a Liferay en funcionamiento en modo "demo". Liferay usará una base de

datos hipersónica por defecto, pero no es apta para la producción. Además, `test@liferay.com` es una cuenta de administrador con una contraseña predeterminada, por lo que debe cambiarse eventualmente. Sin embargo, estos pasos son buenos para tener una idea de cómo se ve y funciona Liferay.

Lea Empezando con liferay en línea: <https://riptutorial.com/es/liferay/topic/932/empezando-con-liferay>

# Capítulo 2: Comunicación entre portlets

## Introducción

Este manual contiene las diversas formas en que el portlet puede coordinarse o comunicarse entre sí y los distintos escenarios para los que se utiliza un enfoque particular.

## Observaciones

Referencias:

1. [Público render param](#)
2. [Especificaciones JSR 286](#)
3. [Sesion de portlet](#)

## Examples

### Usando el parámetro de renderizado público

Este enfoque fue introducido en JSR 286.

En JSR 168, los parámetros de representación establecidos en *processAction* de un portlet estaban disponibles solo en ese portlet. Con la función de parámetros de representación públicos, los parámetros de representación establecidos en la *processAction* de un portlet también estarán disponibles en la representación de otros portlets. Para configurar esto, para todos los portlets que soportan esto:

Agregue la etiqueta `<supported-public-render-parameter>` , justo antes de que la etiqueta del portlet termine en `portlet.xml`

```
<security-role-ref>
  <role-name>user</role-name>
</security-role-ref>
<supported-public-render-parameter>{param-name}</supported-public-render-parameter>
</portlet>
```

Agregue `<public-render-parameter>` etiqueta `<public-render-parameter>` justo antes de que finalice la etiqueta `<portlet-app>`

```
<public-render-parameter>
  <identifier>{param-name}</identifier>
  <qname xmlns:x="localhost">x:{param-name}</qname>
</public-render-parameter>
</portlet-app>
```

En el método `processAction` , el valor `param` debe establecerse en la respuesta

```
res.setRenderParameter({param-name},{param-value});
```

Después de haber ejecutado la fase de acción del portlet en cuestión, el parámetro debe estar disponible en la fase de procesamiento para todos los portlets de soporte en la página, independientemente de ser parte de la misma aplicación o de la diferente. ).

## Usando sesión de portlet

Este es un enfoque que ha estado allí desde JSR 168. Nos permite compartir atributos utilizando la sesión de portlet. Una sesión de portlet puede tener diferentes tipos de ámbitos:

1. Alcance del portlet (atributos disponibles solo dentro del portlet)
2. Ámbito de aplicación (atributos disponibles dentro de toda la aplicación [guerra])

Para utilizar este enfoque, no necesitamos realizar ninguna entrada en la configuración del portlet, ya que la sesión del portlet está disponible en la solicitud del portlet:

```
PortletSession session = renderRequest.getPortletSession();  
session.setAttribute("attribute-name", "attribute-value", PortletSession.APPLICATION_SCOPE);
```

o

```
PortletSession session = renderRequest.getPortletSession();  
session.setAttribute("attribute-name", "attribute-value", PortletSession.PORTLET_SCOPE);
```

El atributo solo se puede recuperar solo del ámbito respectivo. Al igual que para el atributo establecido en el ámbito de portlet, necesitamos obtenerlo usando

```
PortletSession session = renderRequest.getPortletSession();  
String attributeValue = (String) session.getAttribute("attribute-name",  
PortletSession.PORTLET_SCOPE);
```

La principal limitación de este enfoque es la falta de intercambio entre otros portlets, fuera del alcance de la aplicación. Para superar esto, existe un enfoque específico de por vida para agregar

<private-session-attributes > a liferay-portlet.xml

```
<private-session-attributes>false</private-session-attributes>  
<header-portlet-css>/css/main.css</header-portlet-css>  
<footer-portlet-javascript>/js/main.js</footer-portlet-javascript>  
<css-class-wrapper>{portlet-name}</css-class-wrapper>  
</portlet>
```

para todos los portlets, donde se establecen y recuperan los atributos.

## Usando la función de eventos

El mecanismo de eventos es una versión extendida del parámetro de render público, con una función adicional para pasar objetos personalizados a otros portlets, pero con una sobrecarga de fase de eventos.

Para lograr esto, este mecanismo consiste en

1. Portlet editor
2. Portlet del procesador (consumidor), donde ambos pueden formar parte de diferentes aplicaciones de portlet.

Para empezar,

Agregue la etiqueta `<supported-publishing-event>` al portlet del editor en `portlet.xml`

```
<security-role-ref>
  <role-name>user</role-name>
</security-role-ref>
<supported-publishing-event>
  <qname xmlns:x="http:sun.com/events">x:Employee</qname>
</supported-publishing-event>
</portlet>
```

Agregue la etiqueta `<supported-processing-event>` al portlet del procesador en `portlet.xml`

```
<security-role-ref>
  <role-name>user</role-name>
</security-role-ref>
<supported-processing-event>
  <qname xmlns:x="http:sun.com/events">x:Employee</qname>
</supported-processing-event>
</portlet>
```

Agregue la etiqueta `<event-definition>` a ambos portlets, definiendo el nombre del evento y escriba `portlet.xml`

```
<event-definition>
  <qname xmlns:x="http:sun.com/events">x:Employee</qname>
  <value-type>com.sun.portal.portlet.users.Employee</value-type>
</event-definition>
</portlet-app>
```

A continuación, necesitamos crear una clase para el tipo de evento (en el caso de un tipo personalizado)

```
public class Employee implements Serializable {
  public Employee() {
  }
  private String name;
  private int userId;

  public String getName() {
    return name;
  }

  public void setName(String name) {
    this.name = name;
  }
}
```

```
public int getUserId() {
    return userId;
}
public void setUserId(int id)
{
    this.userId = id;
}
```

```
}
```

Ahora, en el portlet del editor, el evento debe publicarse en la fase de acción

```
QName qname = new QName("http:sun.com/events" , "Employee");
Employee emp = new Employee();
emp.setName("Rahul");
emp.setUserId(4567);
res.setEvent(qname, emp);
```

Después de que hayamos publicado el evento, debe ser procesado por el portlet del editor en la fase del evento.

La fase de eventos se introdujo en JSR 286 y se ejecuta antes de la fase de procesamiento del portlet, cuando corresponda

```
@ProcessEvent(qname = "{http:sun.com/events}Employee")
public void processEvent(EventRequest request, EventResponse response) {

    Event event = request.getEvent();
    if(event.getName().equals("Employee")){
        Employee payload = (Employee)event.getValue();
        response.setRenderParameter("EmpName",
            payload.getName());
    }

}
```

que luego se puede recuperar del parámetro de renderización a través de la petición de render

Lea Comunicación entre portlets en línea:

<https://riptutorial.com/es/liferay/topic/8370/comunicacion-entre-portlets>

# Capítulo 3: Configurando SSL

## Observaciones

Asegúrese de tener un certificado ssl válido proporcionado por un tercero. También puede usar un certificado autofirmado, pero solo para dev. [Letsencrypt](#) proporciona certificados gratuitos que se pueden usar en la producción ...

Utilice keytool para importar el certificado a la cadena de teclado de java.

## Examples

### Cómo habilitar SSL en Tomcat y Liferay

Asegúrese de que su archivo de configuraciones de tomcat, server.xml tenga esta línea:

```
<Connector port="8443" protocol="org.apache.coyote.http11.Http11Protocol"
    maxHttpHeaderSize="8192" SSLEnabled="true"
    maxThreads="150" minSpareThreads="25" maxSpareThreads="75"
    enableLookups="false" disableUploadTimeout="true"
    acceptCount="100" scheme="https" secure="true"
    clientAuth="false" useBodyEncodingForURI="true"
    sslEnabledProtocols="TLSv1.2"
    keystorePass="passwordtokeystore"
    keystoreFile="/path/to/.keystoreChain"
    truststoreFile="%JAVA_HOME%/jdk1.8.0_91/jre/lib/security/cacerts"
/>
```

Es importante elegir los protocolos ssl correctos, puede agregar más protocolos ssl con una separación de coma entre los protocolos ssl como este:

```
sslEnabledProtocols="TLSv1, TLSv1.1, TLSv1.2"
```

Luego, asegúrese de que su archivo portal-ext.properties en Liferay tenga estas líneas de configuración:

```
web.server.protocol=https
```

Lea Configurando SSL en línea: <https://riptutorial.com/es/liferay/topic/4320/configurando-ssl>

# Capítulo 4: Configurar el Administrador de etiquetas de Google (GTM) en el ciclo de vida útil

## Introducción

Esta documentación no es específica de liferay, pero puede usarse con referencia a cualquier aplicación web.

Liferay proporciona Google Analytics (conocido como GA a continuación) de manera predeterminada, después de configurar la ID GA - ##### de Analytics en la configuración del sitio. Pero esto proporciona una funcionalidad limitada, que solo permite realizar un seguimiento de las vistas de página (título de página y URL). amplíelo aún más, podemos incrustar el script GA directamente en el tema del sitio para activar los eventos requeridos o usar GTM.

## Examples

### Usando GTM para configurar eventos GA

GTM simplifica todo el proceso de gestión de etiquetas. En la terminología de GTM

1. Colocamos un fragmento de código GTM en la página correspondiente, en `portal_normal.vm` en tema personalizado en el ciclo de vida, que contiene el ID de GTM y una estructura de capa de datos (si es necesario) para asignar valores de página a variables
2. De acuerdo con las variables de la capa de datos, necesitamos crear Variables al final de GTM, que recuperan datos de la capa de datos
3. Posteriormente, creamos etiquetas, que son básicamente campos que asignan variables de la capa de datos a eventos, que se activan en ciertas condiciones, lo que lleva a que los eventos se envíen a la herramienta de seguimiento correspondiente (GA, en nuestro caso).

A continuación se muestra una muestra del fragmento de código GTM javascript incrustado en una página,

```
<body>
<!-- 1) Data layer section -->
<script type="text/javascript">
  dataLayer = [{
    "page" : "<? Virtual path of the page ?>"
    , "pageType" : "<? Page type ?>"
    , "user" : {
      "type" : "<? User type ?>"
      , "userId" : "<? Logged user id ?>"
      , "country" : "<? Logged user country ?>"
      , "userRole" : "<? Role of user ?>"
    }
  }
  ];
```

```
</script>
<!-- 2) GTM Container -->
<noscript><iframe src="//www.googletagmanager.com/ns.html?id=GTM-PK9HK8"
height="0" width="0" style="display:none;visibility:hidden"></iframe></noscript>
<script>(function(w,d,s,l,i){w[l]=w[l]||[];w[l].push({'gtm.start':
new Date().getTime(),event:'gtm.js'});var f=d.getElementsByTagName(s)[0],
j=d.createElement(s),dl=l!='dataLayer'?'&l='+l:'';j.async=true;j.src=
'//www.googletagmanager.com/gtm.js?id='+i+dl;f.parentNode.insertBefore(j,f);
})(window,document,'script','dataLayer','<GTM-ID>');</script>
<!-- End Google Tag Manager -->
```

Después de incluir este script en la página, debemos configurar las variables y etiquetas respectivas desde el final de GTM.

Current Workspace

Default Workspace >

Search

Overview

Tags

Triggers

Variables

Folders

### Built-In Variables ?

**CONFIGURE**

This container has no built-in variables enabled, ...

### User-Defined Variables

**NEW**

Name ▲	Type
page	Data Layer Variable
pageType	Data Layer Variable
URL	URL
userId	Data Layer Variable
userType	Data Layer Variable



userId



### Variable Configuration

Variable type

 Data Layer Variable

Data Layer Variable Name 

user.accountId

### References to this Variable

 **userId Tag**  
Tag

Current Workspace

Default Workspace 



Search



Overview



Tags



Triggers



Variables



Folders

### Tags

NEW

Name ▲

Type

Firing Triggers

[pageType](#)

Universal Analytics

 All Pa

[Universal Analytics](#)

Universal Analytics

 All Pa

[userId Tag](#)

Universal Analytics

 All Pa

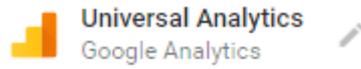
[userType](#)

Universal Analytics

 All Pa

### Tag Configuration

Tag type



Tracking ID ?

Track Type

Event Tracking Parameters

Category

Action

Label

Value

Non-Interaction Hit

> More Settings

Después de haber configurado los campos obligatorios, podemos ver eventos en la consola de GA en una vista de usuario.

Metric Total: 4	
Event Category	Event Action
1. accountId	10135
2. page	/web/france/WCM
3. pageType	Other Page
4. type	Guest

Para ver los datos enviados desde el portal a GA, podemos usar el complemento de [depuración de Google Analytics](#) para ver los eventos que se envían a GA a través de la consola del navegador.

Lea [Configurar el Administrador de etiquetas de Google \(GTM\) en el ciclo de vida útil en línea](#): <https://riptutorial.com/es/liferay/topic/8723/configurar-el-administrador-de-etiquetas-de-google-gtm-en-el-ciclo-de-vida-util>

# Capítulo 5: Crear un planificador de cuarzo en liferay

## Observaciones

Un programador sirve para realizar tareas en segundo plano en ciertos intervalos definidos.

Según el [portlet Liferay DTD](#)

<! - El elemento de entrada del planificador contiene los datos declarativos de un planificador. ->

! ELEMENT entrada-planificador (planificador-descripción, planificador-evento-oyente-clase, disparador)

<! - El valor de descripción del programador describe un programador. ->

! ELEMENT programador-descripción (#PCDATA)

<! - El valor de la clase scheduler-event-listener debe ser una clase que implemente com.liferay.portal.kernel.messaging.MessageListener. Esta clase recibirá un mensaje en un intervalo regular especificado por el elemento desencadenante. ->

! ELEMENT scheduler-event-listener-class (#PCDATA)

<! - El elemento desencadenante contiene datos de configuración para indicar cuándo activar la clase especificada en scheduler-event-listener-class. ->

! ELEMENT gatillo (cron | simple)

## Examples

### Crear un planificador de cuarzo para mostrar alguna información

Para crear un programador, la entrada debe crearse en

```
liferay-portlet.xml
```

demostrando la clase del planificador y el valor de activación para el tiempo de activación del planificador

```
<portlet-name>GetSetGo</portlet-name>
  <icon>/icon.png</icon>
  <scheduler-entry>
    <scheduler-description>This scheduler logs User count from portal</scheduler-
description>
    <scheduler-event-listener-class>com.example.scheduler.SchedulerSample</scheduler-
event-listener-class>
```

```
<trigger>
  <simple>
    <simple-trigger-value>
      5
    </simple-trigger-value>
    <time-unit>minute</time-unit>
  </simple>
</trigger>
</scheduler-entry>
```

La entrada dada proporciona

1. Descripción del programador
2. Nombre de clase, que implementa la clase `MessageListener`
3. Disparador, que proporciona intervalos para definir el punto de disparo para el programador

-Usando a Cron

-Utilizando valor de disparo simple

En el ejemplo dado, el programador se activará cada 5 minutos.

A continuación necesitamos crear una clase de planificador.

```
package com.example.scheduler;

import com.liferay.portal.kernel.exception.SystemException;
import com.liferay.portal.kernel.log.Log;
import com.liferay.portal.kernel.log.LogFactoryUtil;
import com.liferay.portal.kernel.messaging.Message;
import com.liferay.portal.kernel.messaging.MessageListener;
import com.liferay.portal.kernel.messaging.MessageListenerException;
import com.liferay.portal.service.UserLocalServiceUtil;

public class SchedulerSample implements MessageListener {

    @Override
    public void receive(Message arg0) throws MessageListenerException {
        Log log=LogFactoryUtil.getLog(SchedulerSample.class);

        try {
            log.info("User Count for portal:"+UserLocalServiceUtil.getUsersCount());
        } catch (SystemException e) {

            log.info("User count is currently unavailable");
        }
    }
}
```

Este programador simplemente muestra el número de usuarios del portal de salida después de cada intervalo de activación a la consola del servidor.

## Crear un programador dinámico de cuarzo mediante programación

Hay escenarios específicos en los que podríamos necesitar crear un programador de cuarzo, basado en las aportaciones del usuario sobre cuándo debería activarse un programador, además de que podemos manejar casos, donde tenemos ciertas funcionalidades predefinidas, que deben activarse en función de Acción del usuario, en un determinado período.

Este ejemplo recibe la entrada del usuario en la temporización de activación, para activar un scheduler. Here `ScheduledJobListener` clase implements `MessageListener`, que contiene la lógica de negocio para ser ejecutado en impulsar la petición scheduler. The está programado usando `SchedulerEngineHelperUtil` clase para desencadenar el trabajo, después de configurar los params requeridos:

1. Activador (utilizando la cadena de texto cron y el nombre del trabajo)
2. Mensaje (usando la implementación para la clase `MessageListener` y `portletId`)
3. Los tipos de almacenamiento del programador (que es `MEMORY_CLUSTERED` por defecto, se pueden establecer como `PERSISTED` para ser almacenados en DB)
4. `DestinationNames` (que es `SCHEDULER_DISPATCH` para Liferay) que decide el destino del Bus de mensajes que se utilizará

El siguiente fragmento de código es parte de la fase de acción del portlet que interactúa con el usuario para crear y programar un trabajo de cuarzo.

```
//Dynamic scheduling
String portletId= (String)req.getAttribute(WebKeys.PORTLET_ID);

String jobName= ScheduledJobListener.class.getName();

Calendar startCalendar = new GregorianCalendar(year , month, day, hh, mm, ss);
String jobCronPattern = SchedulerEngineHelperUtil.getCronText(startCalendar, false);
//Calendar object & flag for time zone sensitive calendar

Trigger trigger=new
CronTrigger(ScheduledJobListener.class.getName(), ScheduledJobListener.class.getName(),
jobCronPattern);

Message message=new Message();
message.put(SchedulerEngine.MESSAGE_LISTENER_CLASS_NAME, jobName);
message.put(SchedulerEngine.PORTLET_ID, portletId);

try {
    SchedulerEngineHelperUtil.schedule(
trigger, StorageType.PERSISTED, "Message_Desc", DestinationNames.SCHEDULER_DISPATCH,
message, 0);
} catch (SchedulerException e)
{
    e.printStackTrace();
}
```

Aquí, para crear texto cron, los parámetros se recuperan de la entrada del usuario. Para el texto cron, también podemos usar la referencia dada para crear el patrón cron

```

1. Seconds
2. Minutes
3. Hours
4. Day-of-Month
5. Month
6. Day-of-Week
7. Year (optional field)
**Expression**      **Meaning**
0 0 12 * * ?       Fire at 12pm (noon) every day
0 15 10 ? * *      Fire at 10:15am every day
0 15 10 * * ?      Fire at 10:15am every day
0 15 10 * * ? *    Fire at 10:15am every day
0 15 10 * * ? 2005 Fire at 10:15am every day during the year 2005
0 * 14 * * ?       Fire every minute starting at 2pm and ending at 2:59pm, every day
0 0/5 14 * * ?     Fire every 5 minutes starting at 2pm and ending at 2:55pm, every day
0 0/5 14,18 * * ?  Fire every 5 minutes starting at 2pm and ending at 2:55pm, AND fire
every 5 minutes starting at 6pm and ending at 6:55pm, every day
0 0-5 14 * * ?     Fire every minute starting at 2pm and ending at 2:05pm, every day
0 10,44 14 ? 3 WED  Fire at 2:10pm and at 2:44pm every Wednesday in the month of March.
0 15 10 ? * MON-FRI Fire at 10:15am every Monday, Tuesday, Wednesday, Thursday and
Friday
0 15 10 15 * ?     Fire at 10:15am on the 15th day of every month
0 15 10 L * ?      Fire at 10:15am on the last day of every month
0 15 10 L-2 * ?    Fire at 10:15am on the 2nd-to-last last day of every month
0 15 10 ? * 6L     Fire at 10:15am on the last Friday of every month
0 15 10 ? * 6L     Fire at 10:15am on the last Friday of every month
0 15 10 ? * 6L 2002-2005 Fire at 10:15am on every last friday of every month during
the years 2002, 2003, 2004 and 2005
0 15 10 ? * 6#3    Fire at 10:15am on the third Friday of every month
0 0 12 1/5 * ?     Fire at 12pm (noon) every 5 days every month, starting on the first day
of the month.
0 11 11 11 11 ?   Fire every November 11th at 11:11am.

```

y cree directamente una cadena crontext para usar en función de la entrada del usuario

```
String jobCronPattern="0 */5 * * * ?";
```

Aquí, en este caso, se dispara cada cinco minutos.

Referencias:

1. [Creación dinámica del planificador](#)
2. [Aplicación de programador](#)
3. [Preguntas frecuentes de cuarzo](#)

Lea [Crear un planificador de cuarzo en liferay en línea](#):

<https://riptutorial.com/es/liferay/topic/7998/crear-un-planificador-de-cuarzo-en-liferay>

---

# Capítulo 6: Depurar el servidor de liferay remoto a través de Eclipse

## Examples

Depure el servidor de tiempo de vida remoto a través de Eclipse (sin el conector Eclipse del conector IDE de Liferay Remote)

Para depurar una instancia del servidor, comience en modo de depuración. Para hacerlo, configure estos parámetros para pasarlos al servidor:

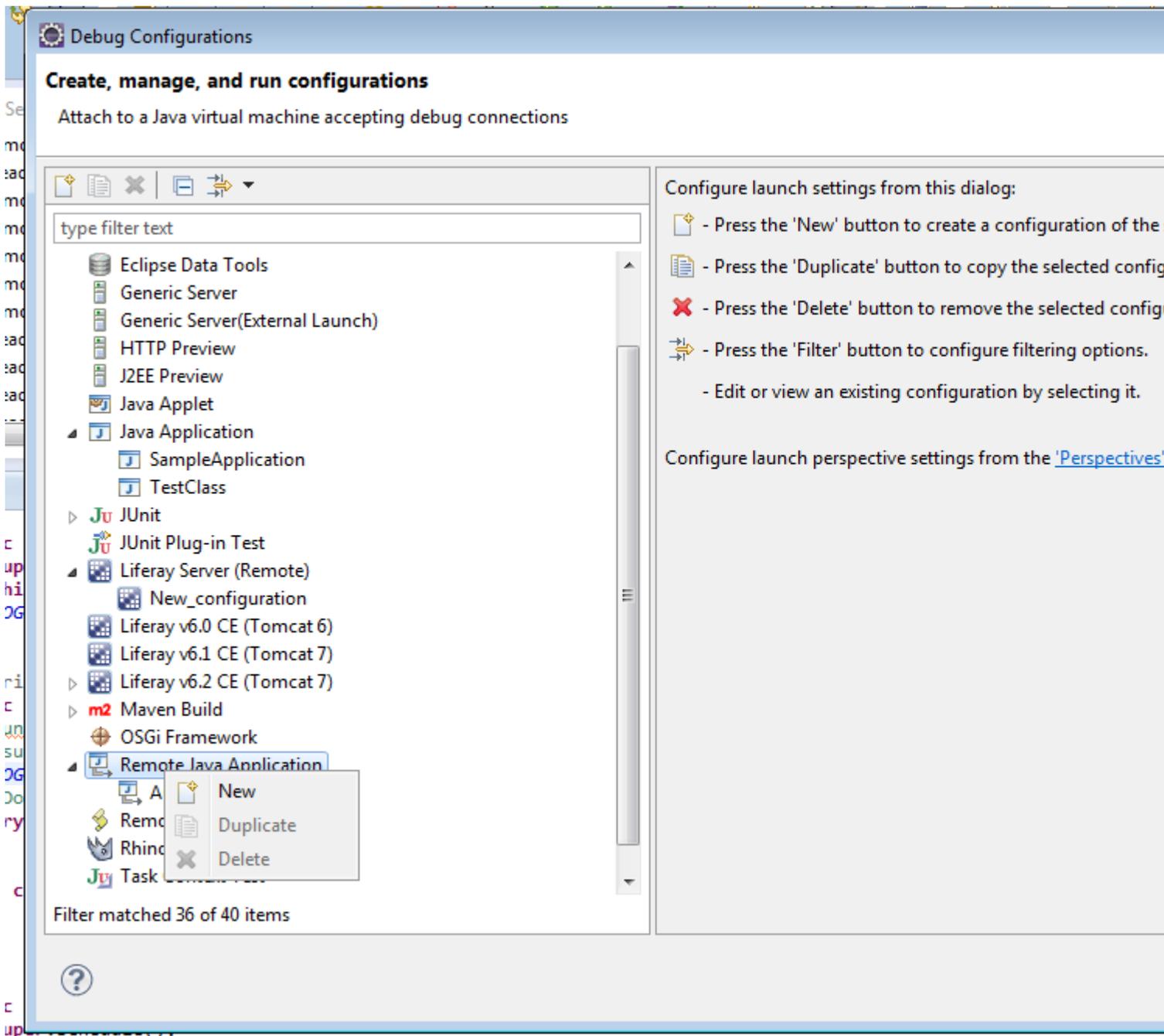
```
-Xdebug -Xrunjdp:transport=dt_socket,address=8000,server=y,suspend=n
```

a setenv.bat (Windows) o setenv.sh (Unix)

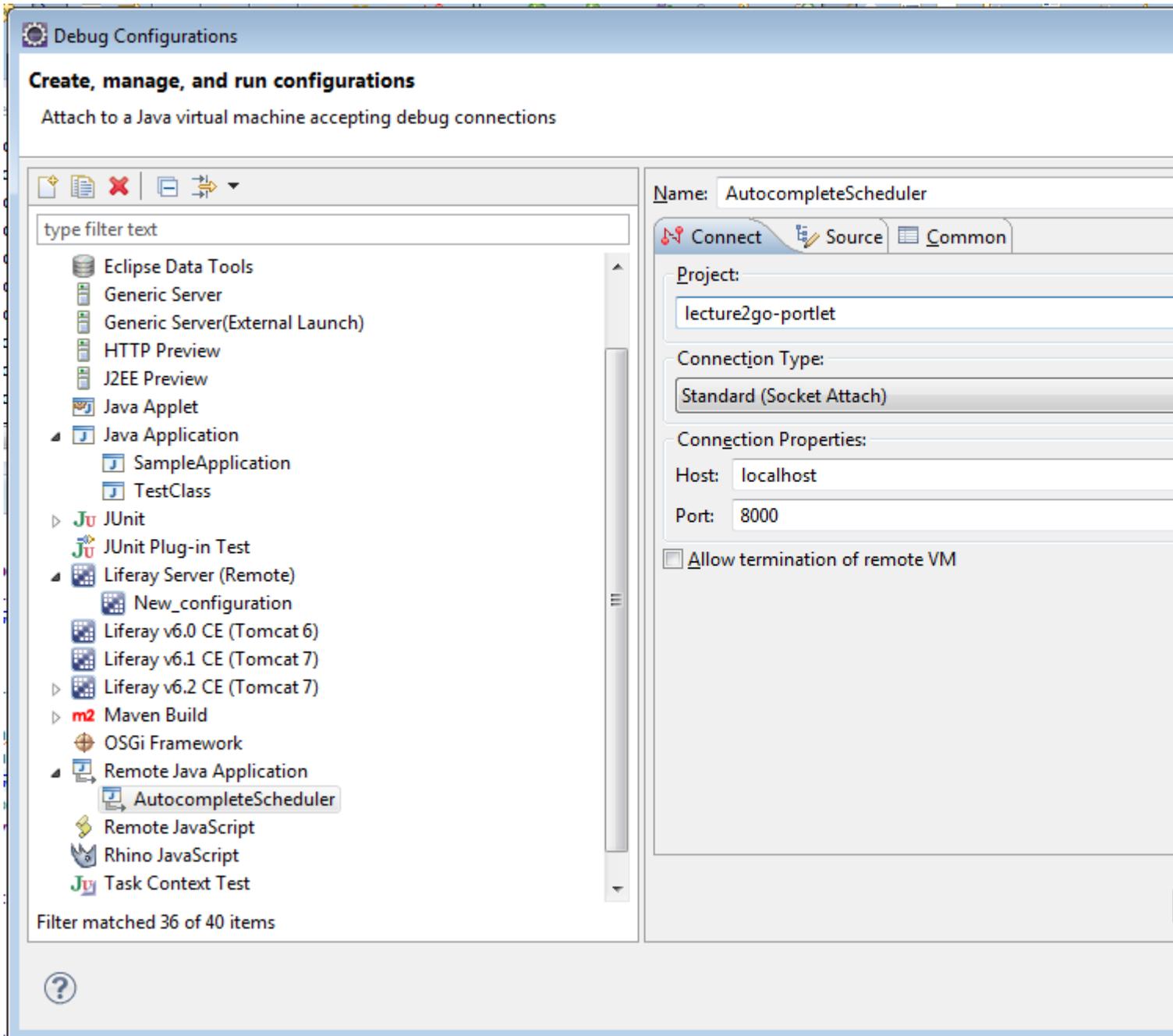
Estos inicializan el servidor en modo de depuración y escuchan las solicitudes de depuración en el puerto dado. Inicia el servidor y publica la configuración.

En eclipse, la configuración de depuración remota debe configurarse para adjuntar la fuente al servidor remoto. Siga los pasos dados:

1. Vaya a *Ejecutar-> Configuraciones de depuración-> Aplicación java remota* :



2. Crea una nueva configuración desde la aplicación Java remota:



3. Ingrese los detalles dados:

```
Host name:localhost (For local instance) or Ip of the machine  
Port:8000 (By default)
```

4. Haga clic en *Depurar* para iniciar archivos adjuntos en la instancia del servidor.

Lea *Depurar el servidor de liferay remoto a través de Eclipse en línea*:

<https://riptutorial.com/es/liferay/topic/7891/depurar-el-servidor-de-liferay-remoto-a-traves-de-eclipse>

---

# Capítulo 7: Desplegando un complemento

## Examples

### Despliegue en Glassfish

Entonces, primero crea un archivo .war, digamos un portlet con el nombre `<YOUR_PLUGIN>.war` .  
Quieres que se ejecute en un dominio de glassfish en el portal Liferay.

Pasos para el éxito:

1. Navegue al Panel de control -> Instalación de complementos en Liferay
2. Hit **Instalar nuevos portlets**
3. **Configuración de hit**
4. Rellene para Implementar Directorio un nuevo lugar para implementación digamos `<YOUR_DOMAIN>/autodeploy2`
5. Verifique que en la siguiente línea el destino sea `<YOUR_DOMAIN>/autodeploy` (es el directorio de implementación predeterminado de Glassfish)
6. Pulsa **guardar**

Ahora la implementación se realizará copiando los archivos en ese directorio nuevo `<YOUR_DOMAIN>/autodeploy2` . El resto se maneja automáticamente. El ajuste toma acción inmediatamente.

Hecho con despliegue: hacer una plantilla de victoria y disfrutar :)

... dejas de bailar y te enfrentas a un bicho. Desea implementar una nueva revisión ... En este caso, continúe leyendo.

Entonces, has construido tu guerra de nuevo y quieres volver a desplegarla. Haz lo siguiente:

1. `<YOUR_DOMAIN>/autodeploy` cosas viejas de la carpeta `<YOUR_DOMAIN>/autodeploy` eliminando el archivo war. No borres ningún otro archivo.
2. El resultado es que `<YOUR_PLUGIN>.war_UnDeployed` archivo `<YOUR_PLUGIN>.war_UnDeployed` .
3. implemente el nuevo archivo copiando la nueva guerra `<YOUR_DOMAIN>/autodeploy2` carpeta `<YOUR_DOMAIN>/autodeploy2` .
4. el resultado es que `<YOUR_PLUGIN>.war_deployed` aparecerá en la carpeta `<YOUR_DOMAIN>/autodeploy` .

Hacer un baile de nuevo :)

Lea [Desplegando un complemento en línea](https://riptutorial.com/es/liferay/topic/1708/desplegando-un-complemento):

<https://riptutorial.com/es/liferay/topic/1708/desplegando-un-complemento>

# Capítulo 8: Ganchos en Liferay

## Observaciones

Esto funciona con Liferay Portal hasta la versión 6.2.

## Examples

### JSP Hook

Los ganchos JSP son un complemento especial de por vida que permite modificar el portlet principal `jsp-s`, digamos que desea modificar el portlet de inicio de sesión para mostrar ¡ Welcome in my custom login! .

La estructura mínima para un complemento de gancho es la siguiente:

```
[project-name]-hook/  
├── docroot/  
│   ├── WEB-INF/  
│   │   ├── src/  
│   │   ├── lib/  
│   │   ├── liferay-hook.xml  
│   │   ├── liferay-plugin-package.properties  
│   │   └── web.xml  
│   └── META-INF/  
│       ├── custom_jsps/  
│       └── MANIFEST.MF
```

`liferay-hook.xml` es el archivo que distingue el tipo de gancho que está utilizando, aquí define dentro de la etiqueta gancho el parámetro adecuado para el gancho, para el gancho JSP:

```
<?xml version="1.0"?>  
<!DOCTYPE hook PUBLIC "-//Liferay//DTD Hook 6.2.0//EN" "http://www.liferay.com/dtd/liferay-hook_6_2_0.dtd">  
  
<hook>  
  <custom-jsp-dir>/custom_jsps</custom-jsp-dir>  
</hook>
```

`login.jsp` se encuentra en Liferay en `/docroot/html/portlet/login/login.jsp` , para hacer un gancho de esto necesitamos agregar un `jsp` con el mismo nombre y ruta en nuestra carpeta `custom_jsps` .

Cuando se implementa el gancho, Liferay buscará en el `liferay-hook.xml` el valor `custom-jsp-dir` y reemplazará todas las JSP del portal con las que se encuentran en este directorio. Los `jsp` originales se guardan con el nombre `<original name>.portal.jsp` para ser restaurados en caso de `<original name>.portal.jsp` de `<original name>.portal.jsp` del gancho.

Incluso podemos llamar a los JSP originales en el nuevo JSP modificado si queremos mantener el

código para que sea adaptable a las actualizaciones o actualizaciones de la versión de la plataforma Liferay subyacente. Para hacer esto, en su JSP personalizado use el siguiente patrón:

```
<liferay-util:buffer var="contentHtml">
  <liferay-util:include page="/html/{ JSP file's path }" />
</liferay-util:buffer>
```

donde { JSP file's path } en este caso será `portlet/login/login.portal.jsp` . Hacer esto se llama *extender el jsp original* .

Entonces podemos agregarle contenido con:

```
<%
contentHtml = StringUtil.add("Stuff I'm adding BEFORE the original content",
contentHtml, "\n");
contentHtml = StringUtil.add(contentHtml, "Stuff I'm adding AFTER the original content", "\n");
%>
<%= contentHtml %>
```

## Puntales ganchos de acción

Este tipo de Hook se puede usar para anular el portal central (por ejemplo, `c/portal/login` ) y las acciones de portlet struts (por ejemplo, `/login/forgot_password` ), estas acciones para Liferay Portal se especifican en un archivo `struts-config.xml` en su `WEB-INF` Carpeta `WEB-INF` Para anular una acción:

1. en el archivo `liferay-hook.xml` de su complemento de `docroot/WEB-INF` en `docroot/WEB-INF` , agregue un elemento `struts-action` dentro del elemento `hook`.
2. Dentro `struts-action` elemento `struts-action` , agregue `struts-action-path` que especifique la ruta de acción que está reemplazando y `struts-action-impl` que especifica su clase de acción personalizada. Esto parece:

```
<struts-action-path>/login/login</struts-action-path>
<struts-action-impl>
  com.myhook.action.ExampleStrutsPortletAction
</struts-action-impl>
</struts-action>
```

3. Cree una clase de acción de portlet de Struts que amplíe `BaseStrutsPortletAction` . Un ejemplo de esta clase es:

```
public class ExampleStrutsPortletAction extends BaseStrutsPortletAction {

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

        System.out.println("Custom Struts Action");

        originalStrutsPortletAction.processAction(originalStrutsPortletAction,
            portletConfig, actionRequest, actionResponse);
    }
}
```

```

public String render(StrutsPortletAction originalStrutsPortletAction,
    PortletConfig portletConfig, RenderRequest renderRequest,
    RenderResponse renderResponse) throws Exception {

    System.out.println("Custom Struts Action");

    return originalStrutsPortletAction.render(null, portletConfig,
        renderRequest, renderResponse);
}
}

```

Invocar el método que se reemplaza, como `originalStrutsPortletAction.processAction`, no es obligatorio, pero es una buena práctica mantener el comportamiento de la Acción sin cambios con respecto al Portal Liferay. Este tipo de gancho también se puede usar para agregar nuevas acciones de Struts, es lo mismo que modificar una acción existente, en este caso, `liferay-hook.xml` sería:

```

<struts-action>
  <struts-action-path>/my/custom/path</struts-action-path>
  <struts-action-impl>
    com.myhook.action.ExampleStrutsAction
  </struts-action-impl>
</struts-action>

```

## Hola usuario "Nombre" con ganchos

Este ejemplo mostrará cómo hacer un simple "Hola usuario [nombre]" después del inicio de sesión. El ejemplo se basa en [realizar una acción personalizada utilizando un gancho](#)

Desde su terminal de línea de comando, navegue a la carpeta de enlaces de su Plugins SDK. Para crear un proyecto de gancho, debe ejecutar el script de creación. Aquí está el formato a seguir para ejecutar el script:

```
create. [sh | bat] [nombre del proyecto] "[Nombre de visualización del enganche]"
```

En Linux y Mac OS X, ingresaría un comando similar al de este ejemplo:

```
./create.sh Hola-usuario "Hola usuario"
```

En Windows, ingresarías un comando similar al de este ejemplo:

```
create.bat Hola-usuario "My Hook"
```

El asistente de nuevo proyecto de Liferay IDE y los scripts de creación generan proyectos de enlace en la carpeta de enlaces de su SDK de plugin. El SDK de complementos agrega automáticamente "-hook" al nombre de su proyecto.

Ya sea que haya creado su proyecto de enganche desde el IDE de Liferay o desde la línea de comandos, termina con la misma estructura de proyecto (ver antes).

- Determine el evento en el que desea activar su acción personalizada. Busque en la

documentación de **portal.properties** para encontrar la propiedad de evento correspondiente. Sugerencia: las propiedades del evento tienen `.event` en su nombre. Hay las propiedades de sesión, inicio, cierre y evento del portal en las siguientes secciones de la documentación de **portal.properties** : [Sesión](#) - [Eventos de inicio](#) - [Eventos de cierre](#) - [Eventos del portal](#)

- En su proyecto de gancho, cree una clase Java que amplíe la **clase `com.liferay.portal.kernel.events.Action`**. Reemplace el **método `Action.run`** (**`HttpServletRequest`, `HttpServletResponse`**) .

```
import com.liferay.portal.kernel.events.Action;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import com.liferay.portal.model.User;
import com.liferay.portal.util.PortalUtil;

public class HelloUser extends Action {
    public void run(HttpServletRequest req, HttpServletResponse res) {
        User user = PortalUtil.getUser(req);
        System.out.println("Hello User "+user.getScreenName());
    }
}
```

Importante: si su acción accede al objeto `HttpServletRequest`, extienda `com.liferay.portal.kernel.events.Action`; de lo contrario, extienda `com.liferay.portal.struts.SimpleAction`.

- Cree un archivo de propiedades, **portal.properties** , dentro de la carpeta `docroot / WEB-INF / src` de su proyecto hook. Luego agregue el nombre de la propiedad de evento del portal que corresponde al evento en el que desea realizar su acción. Especifique el nombre completo de su clase de acción como el valor de la propiedad.

```
`login.events.post=HelloUser`
```

Por ejemplo, para realizar una acción de clase justo antes de que el portal inicie sesión en un usuario, debe especificar la propiedad `login.events.pre` con su clase de acción como su valor. Podría parecerse a esta configuración de propiedad.

Importante: ya que las propiedades del portal como `login.events.pre` aceptan varios valores, debe agregar sus valores a los valores existentes. Puede modificar repetidamente las propiedades de ganchos adicionales.

Solo modifique una propiedad del portal que acepte un valor único de un solo complemento de enlace. Si modifica el valor de una propiedad de varios complementos, Liferay no sabrá qué valor usar.

- Edite su **archivo `docroot / WEB-INF / liferay-hook.xml`** y agregue su nombre del archivo de propiedades del portal del gancho como el valor para el `<portal-properties>...</portal-properties>` dentro de su El elemento `<hook>...</hook>` . Por ejemplo, si el nombre del archivo de propiedades de su gancho es **portal.properties** , especificaría este elemento:

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

- Implemente su enganche, vaya a su ruta de enganche e ingrese a `ant clean deploy` verá la `.war` en la carpeta `dist`.

Ahora, si inicia sesión en liferay, verá en el registro del servidor un mensaje como "Hola, administrador del usuario".

## Modelo Listener Hook

# Fondo

Los ganchos de escucha del modelo son un tipo de complemento Liferay que escucha los eventos tomados en un modelo y ejecuta el código en respuesta. Los ganchos de escucha de modelo son similares a los ganchos de acción de Struts personalizados, ya que responden a una acción realizada en el portal. Sin embargo, mientras que las acciones de Struts responden a una acción realizada por un usuario, un escucha de modelo responde (antes o después) de un evento que involucra un modelo de Liferay.

# Diferencias

Aquí hay algunos ejemplos de Struts Actions v. Model Listeners para comparación.

- **Puntales de acción**
  - Inicio de sesión de usuario
  - Creación de cuenta
  - Ampliar sesión
  - Mover carpeta
- **Escucha modelo**
  - Después de crear la carpeta
  - Cuando la información del usuario se actualiza
  - Después de eliminar el marcador
  - Antes de que se haga una asociación de roles.

El mejor recurso para aprender la arquitectura de Liferay es a través de su código fuente. Todos sus archivos de origen se encuentran en GitHub y al ver sus JavaDocs. Puede ver todos los modelos de portal principales [en los JavaDocs](#) y todas las Acciones de Struts [en el GitHub](#).

# Ejemplo

En este tutorial, desarrollaremos una escucha de modelo que envía un correo electrónico a un

usuario después de que se crea su cuenta por primera vez. Para hacer esto, vamos a escribir una clase llamada **UserModelListener** que extenderá la **BaseModelListener** de Liferay. Repasaremos brevemente la creación de ganchos y cubriremos las modificaciones necesarias a los siguientes archivos de configuración

- **portal.propiedades**
- **liferay-hook.xml**

---

## Empezando

Para comenzar a desarrollar su gancho de escucha de modelo, primero debe iniciar su aplicación Liferay IDE o Liferay Developer Studio.

Tanto Liferay IDE como Liferay Developer Studio son entornos de desarrollo **Eclipse** personalizados. Son sorprendentemente similares y un conjunto de direcciones debería ser suficiente para ambos entornos.

Dentro de tu entorno de desarrollo ejecuta los siguientes pasos.

1. En la esquina superior izquierda, haga clic en **Archivo**
2. Pase el mouse sobre **Nuevo**
3. Haga clic en **Proyecto Liferay Plugin**

Usted generará esta ventana.

## New Liferay Plugin Project

### Liferay Plugin Project

Create a new project configured as a Liferay

Project name:

Display name:

Use default location

Location:

Build type:

Plugins SDK:

Liferay runtime:

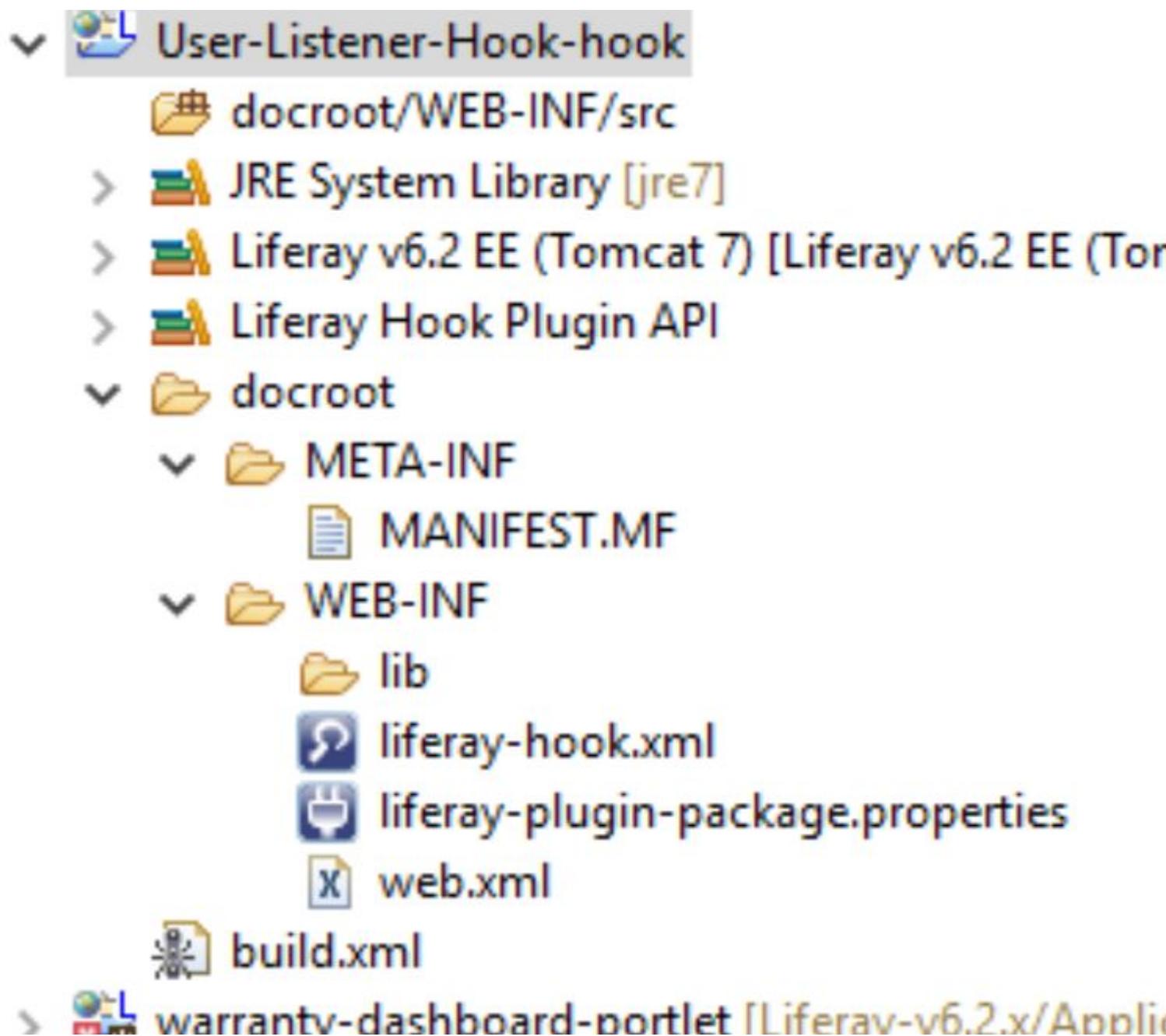
Plugin type:

Add project to working set

- Seleccione **Usar ubicación predeterminada**
- Tipo de construcción: **hormiga**
- Tipo de complemento: **Gancho**

Asegúrese de que su proyecto se encuentre dentro de su **directorio de ganchos SDK de Liferay Plugins** . Deberá seleccionar su **SDK** y su **Runtime en** consecuencia.

En la perspectiva de su **Explorador de paquetes** verá la siguiente estructura de directorios.



## Desarrollo del oyente

Ahora que ha creado su gancho, deberá crear su clase personalizada **UserModelListener** . Esta clase ampliará la clase **BaseModelListener** de Liferay.

La clase `BaseModelListener` de Liferay es una clase abstracta que implementa la interfaz `ModelListener`. No desea implementar la interfaz `ModelListener` directamente, ya que le exigirá que invalide todos sus métodos.

La interfaz **`ModelListener`** le proporciona los siguientes métodos a través de la clase abstracta **`BaseModelListener`** .

- `onAfterAddAssociation`
- `onAfterCreate`
- `onAfterRemove`
- `onAfterRemoveAssociation`
- `onAfterUpdate`
- `onBeforeAddAssociation`
- `onBeforeCreate`
- `onBeforeRemove`
- `onBeforeRemoveAssociation`
- `onBeforeUpdate`

Cree su clase **`UserModelListener`** dentro del siguiente directorio. Para crear la clase a través de la GUI simplemente ejecute los siguientes comandos

- Haga clic en **Archivo** en la esquina superior izquierda
- Pase el mouse sobre **Nuevo**
- Haga clic en **clase**

```
docroot/  
  WEB-INF/  
    src/
```

Ingrese la información que se muestra a continuación

 New Java Package

## Java Package

Create a new Java package.

Creates folders corresponding to packages.

Source folder:

Name:

Create package-info.java

, empaquetada dentro de **com.example.code**, para el modelo **DLFolder** tendríamos la siguiente propiedad

```
value.object.listener.com.liferay.portal.model.DLFolder =  
com.example.code.CustomerDLFolderModelListener
```

Por último, ubique su archivo `liferay-hook.xml`. En la vista **Fuente**, escriba lo siguiente.

```
<?xml version="1.0"?>  
<!DOCTYPE hook PUBLIC "-//Liferay//DTD Hook 6.2.0//EN" "http://www.liferay.com/dtd/liferay-  
hook_6_2_0.dtd">  
  
<hook>  
  <portal-properties>portal.properties</portal-properties>  
</hook>
```

## Explicación

1. La línea uno es un **prólogo** opcional que especifica la versión del documento y (en algunos casos) el conjunto de caracteres.
2. La línea 2 es una **Definición DocType** formal (DTD) que define explícitamente qué elementos y atributos son válidos
3. Las líneas 3 y 5 están formadas por el **elemento** principal de **enlace** (uno de los elementos válidos admitidos por esta DTD)
4. La línea 4 anula y extiende el archivo **portal.properties** en **\$ {liferay.home}**

Para ver qué otros elementos se pueden usar en este archivo XML, puede hacer referencia a la URL dentro de la [Definición de DocType](#). Esto es estándar para todos los archivos **XML** y **SGML** con una **DTD**. Otro ejemplo de un archivo **XML de Liferay** con un **DTD** es **service.xml** (implementación de **ORM** de Liferay basada en **Hibernate**).

---

## Construir e implementar

Construir y desplegar ganchos es un proceso simple. El desarrollo de Liferay Plugin es compatible con la compilación y la automatización de dependencias con

- Hormiga
- Hiedra
- Maven
- Gradle

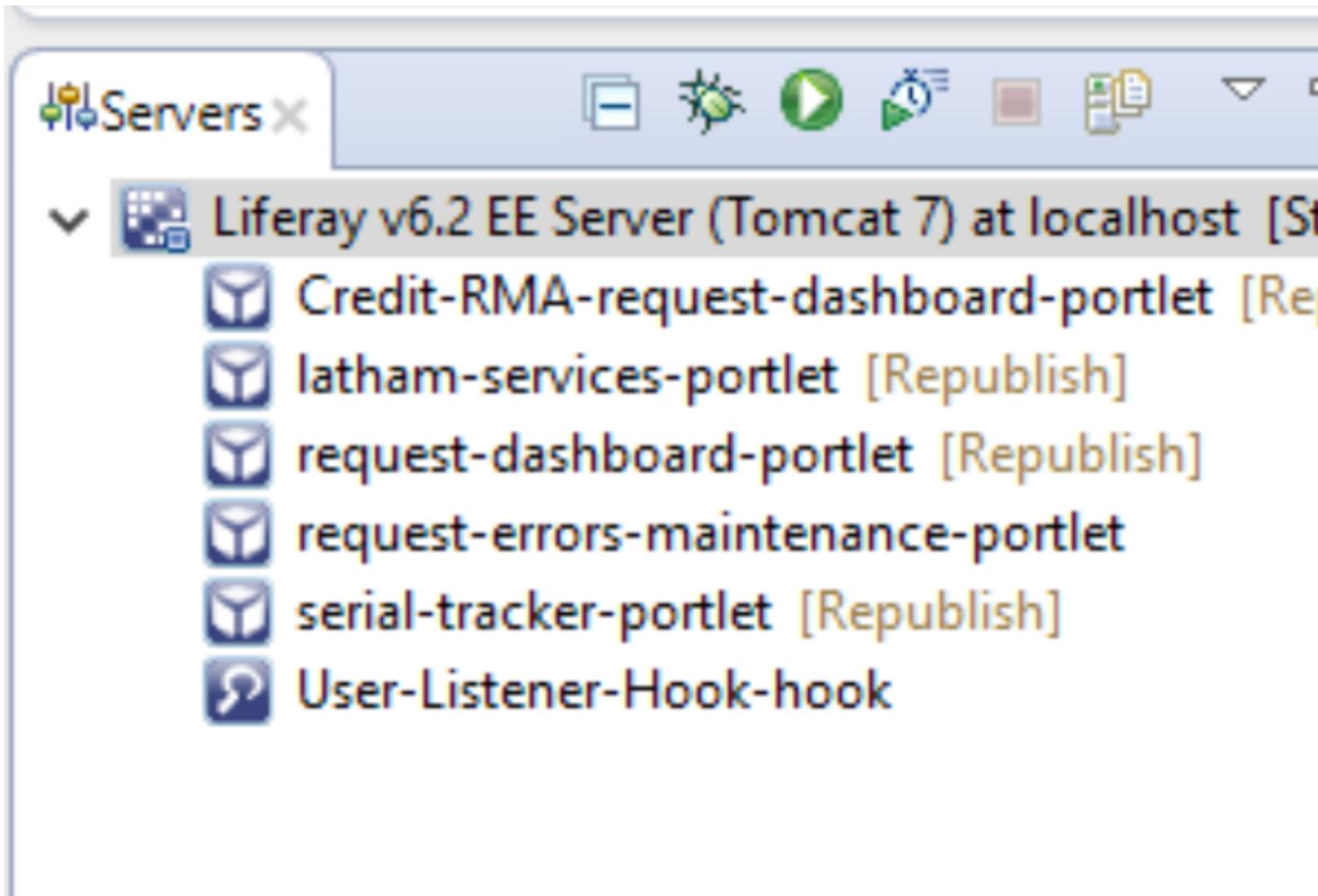
En nuestro ejemplo utilizamos **Ant** para la automatización de la construcción. El archivo **build.xml** contiene los comandos de compilación (conocidos como **objetivos** en **Ant**). Para construir su gancho simplemente ejecute los siguientes comandos.

1. **Encuentra** tu archivo **build.xml**

2. En su IDE, arrastre el archivo **build.xml** a la perspectiva **Ant**
3. Expande el archivo y ejecuta **todos los** objetivos.

- ▼  User-Listener-Hook-hook [User-Listener-Hook-ho
  - >  all [from import ../build-common-plugin.xml]
  - >  build-client [from import ../build-common-pl]
  - >  build-db [from import ../build-common-plugi]
  - >  build-lang [from import ../build-common-plu]
  - >  build-lang-cmd [from import ../build-commo]
  - >  build-service [from import ../build-common-p]
  - >  build-wsdd [from import ../build-common-pl]
  - >  build-wsdl [from import ../build-common-plu]
  - >  build-xsd [from import ../build-common-plug]
  - >  clean [from import ../build-common-plugin.x]
  - >  clean-portal-dependencies [from import ../bui]
  - >  compile [from import ../build-common-plugi]
  - >  compile-import-shared [from import ../build-]
  - >  compile-java [from import build-common.xml]
  - >  compile-test [from import ../build-common-p]
  - >  compile-test-cmd [from import ../build-comm]
  - >  compile-test-integration [from import ../build]
  - >  compile-test-unit [from import ../build-comm]
  - >  create [from import build-common.xml [from]
  - >  deploy [default] [from import ../build-commo]
  - >  direct-deploy [from import ../build-common-
  - >  format-javadoc [from import build-common.x]

2. Localice **User-Listener-Hook** en la selección **Disponible**
3. Una vez resaltado haga clic en el botón **Agregar** y haga clic en **Aceptar**
4. Haga clic en el botón **Reproducir** en la perspectiva **Servidor**



---

**Déjeme saber si tiene alguna pregunta, comentario, inquietud, etc. ¡Todos los comentarios constructivos son muy apreciados!**

---

Lea Ganchos en Liferay en línea: <https://riptutorial.com/es/liferay/topic/3712/ganchos-en-liferay>

---

# Capítulo 9: Usando consultas SQL dinámicas y personalizadas en Liferay

## Introducción

Hay situaciones cuando se trata de la capa de servicio en el ciclo de vida útil, cuando necesitamos consultar la base de datos con demasiadas cláusulas o tratar con varias tablas. En tales casos, usamos cualquiera de los siguientes:

- 1) Consulta dinámica (envoltura en la API de criterios de hibernación)
- 2) consultas SQL personalizadas

## Observaciones

Referencias:

1. [SQL personalizado](#)
2. [Consulta dinámica](#)

## Examples

### Usando la consulta dinámica en Liferay

Para la mayoría de los escenarios que involucran entidades de la capa de servicio, podemos conformarnos con las llamadas de servicio predeterminadas, también con la ayuda de los buscadores. Para escenarios simples que involucran múltiples entidades, avanzamos hacia el uso de la API de consulta dinámica. Esta es una API de contenedor. para la API de criterios utilizada en Hibernate. Puede usarse para casos en los que necesitamos generar consultas dinámicas, que no son de naturaleza muy compleja, utilizando varias construcciones de la API. Para empezar, algunas de las construcciones más utilizadas son: `DynamicQueryFactoryUtil` para construir consultas

`RestrictionsFactoryUtil` utiliza para proporcionar restricciones en los campos de comparación con un determinado valor para reducir los resultados que coinciden con un determinado valor o dentro de un rango, etc.

`ProjectionFactoryUtil` utiliza para proporcionar proyecciones para obtener campos que formarán parte del resultado de la búsqueda, es decir, en lugar de proporcionar la entidad completa, proporcionará solo ciertos campos o aplicará la función de agregación (como min.max, avg) en el mismo.

`PropertyFactoryUtil` utiliza para la comparación de algunas propiedades de la clase de entidad para realizar mayormente comparacion con otros campos de una consulta

La implementación de estas clases está presente en el paquete dao.orm.jpaa con todos los métodos disponibles.

Lea Usando consultas SQL dinámicas y personalizadas en Liferay en línea:

<https://riptutorial.com/es/liferay/topic/10863/usando-consultas-sql-dinamicas-y-personalizadas-en-liferay>

---

# Capítulo 10: Usando el servicio web Restful en Liferay

## Examples

Consumir el servicio JSON de Liferay para solicitudes GET.

Liferay expone muchos servicios predeterminados y personalizados disponibles para otros sistemas a través de JSON. Para explorar servicios en una instancia particular de liferay, use una URL dada - Una instancia local en este caso:

```
http://localhost:8080/api/jsonws/
```

Context Path

- Address
- add-address
  - add-address
  - delete-address
  - get-address
  - get-addresses
  - update-address

- AnnouncementsDelivery
- update-delivery

- AnnouncementsEntry
- add-entry
  - add-entry
  - delete-entry
  - get-entry
  - update-entry

- AnnouncementsFlag
- add-flag
  - delete-flag

## /user/get-user-by-email-address

**com.liferay.portal.service.impl.UserService**  
Provides the remote service for accessing, adding, and deleting users. It also includes permission checks.

### **getUserByEmailAddress**

## Parameters

- companyId** long
- emailAddress** java.lang.String

## Return Type

**com.liferay.portal.model.User**

## Exception

- com.liferay.portal.kernel.exception.PortalException**
- com.liferay.portal.kernel.exception.SystemException**

## Execute

companyId  long

emailAddress  java.lang.String

Seleccione el servicio requerido, consuma el servicio con la sintaxis y los parámetros dados:

```
/user/get-user-by-email-address
```

Utilice `companyId` and `emailAddress` para recuperar al usuario con los tipos de datos esperados, así como las posibles excepciones que debe manejar el consumidor.

El siguiente ejemplo consume este servicio de un portlet. El método de clase de utilidad dado realiza una llamada al servicio web, pasando los argumentos necesarios:

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;

import org.json.simple.JSONObject;
import org.json.simple.parser.JSONParser;
import org.json.simple.parser.ParseException;

import sun.misc.BASE64Encoder;

import com.liferay.portal.kernel.util.StringUtil;
import com.liferay.portal.theme.ThemeDisplay;

public class WebServiceUtil {

public static String requestWebService(ThemeDisplay themeDisplay) {

    String url="http://localhost:8080/api/jsonws/user/get-user-by-email-address/company-
id/{company-id}/email-address/{email-address}";

    String groupId= Long.toString(themeDisplay.getCompanyId());
    String userEmail="test@liferay.com";

    String[] searchList={"{company-id}", "{email-address}"};
    String[] replList={groupId, userEmail};

    //Path params are replaced with args to make web service call
    url=StringUtil.replace(url, searchList, replList);

    System.out.println(url);
    StringBuilder sb = new StringBuilder();
    JSONObject jsonObject=new JSONObject();
    try
    {
        URL urlVal = new URL(url);
        HttpURLConnection conn = (HttpURLConnection) urlVal.openConnection();

        //The user credentials are directly used here only for the purpose of example,always
        fetch these details from an external props file.

        String uname ="test@liferay.com";
        String pswd="test";
        String authStr=uname+": "+pswd;

        //Encoding username+pswd to be added to request header for making web service
        call
        String authStrEnc=new BASE64Encoder().encode(authStr.getBytes());

        /*Authorization type is set to consume web service
```

```

and encoded combination is set in header to authenticate caller*/

conn.setRequestMethod("GET");
conn.setRequestProperty("Accept", "application/json");
conn.setRequestProperty("Authorization", "Basic "+authStrEnc);

BufferedReader brf = new BufferedReader(new InputStreamReader(conn.getInputStream()));

JSONParser json=new JSONParser();
JSONObject=(JSONObject) json.parse(brf);

int cp;
while ((cp = brf.read()) != -1) {
    sb.append((char) cp);
}
}
catch(IOException e)
{
    System.out.println("Something went wrong while reading/writing in stream!!");
}
catch (ParseException e) {
    System.out.println("Parse error");
}

//For purpose of simplicity we have fetched one of the fields from JSON response
return (String)jsonObject.get("firstName");

}

}

```

Lea Usando el servicio web Restful en Liferay en línea:

<https://riptutorial.com/es/liferay/topic/7821/usando-el-servicio-web-restful-en-liferay>

# Creditos

S. No	Capítulos	Contributors
1	Empezando con liferay	<a href="#">brandizzi</a> , <a href="#">Community</a> , <a href="#">Pier Paolo Ramon</a> , <a href="#">Pierpaolo Cira</a> , <a href="#">rp.</a>
2	Comunicación entre portlets	<a href="#">a_horse_with_no_name</a> , <a href="#">Shivam Aggarwal</a>
3	Configurando SSL	<a href="#">EI0din</a> , <a href="#">rp.</a> , <a href="#">Tofik Sahraoui</a>
4	Configurar el Administrador de etiquetas de Google (GTM) en el ciclo de vida útil	<a href="#">Shivam Aggarwal</a>
5	Crear un planificador de cuarzo en liferay	<a href="#">Shivam Aggarwal</a>
6	Depurar el servidor de liferay remoto a través de Eclipse	<a href="#">4444</a> , <a href="#">Shivam Aggarwal</a>
7	Desplegando un complemento	<a href="#">mico</a> , <a href="#">Pier Paolo Ramon</a> , <a href="#">rp.</a>
8	Ganchos en Liferay	<a href="#">Chris Maggiulli</a> , <a href="#">EI0din</a> , <a href="#">KLajdPaja</a> , <a href="#">Pier Paolo Ramon</a> , <a href="#">rp.</a>
9	Usando consultas SQL dinámicas y personalizadas en Liferay	<a href="#">Shivam Aggarwal</a>
10	Usando el servicio web Restful en Liferay	<a href="#">4444</a> , <a href="#">Shivam Aggarwal</a>