eBook Gratuit

APPRENEZ liferay

eBook gratuit non affilié créé à partir des **contributeurs de Stack Overflow.**

#liferay

Table des matières

À propos

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.

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

Chapitre 1: Commencer avec liferay

Remarques

Liferay Portal CE est un logiciel de portail construit en Java par Liferay Inc.

Liferay DXP (Digital Experience Platform) est une plate-forme construite sur le portail Liferay pour les solutions numériques, intégrant des outils d'analyse de la satisfaction des clients et des utilisateurs, ainsi que des performances et des outils de qualité professionnelle. Il était connu sous le nom de *Liferay Portal EE* .

Depuis la version 7.0, il est construit avec **OSGi** via **Apache Felix** .

Versions

Version	Nom de code	Type de libération	Date de sortie
7.0.1 GA2	Wilberforce	Edition communautaire	2016-06-10
7.0.10	Wilberforce	Plateforme d'expérience numérique	2016-05-04
7.0.0 GA1	Wilberforce	Edition communautaire	2016-03-31
6.2.3 GA4	Newton	Edition communautaire	2015-04-17
6.2.2 GA3	Newton	Edition communautaire	2015-01-15
6.2.1 GA2	Newton	Edition communautaire	2014-02-28
6.2.10 GA1	Newton	Edition pour entreprise	2013-12-03
6.2.0 GA1	Newton	Edition communautaire	2013-11-01
6.1.2 GA3	Paton	Edition communautaire	2013-08-23
6.1.30 GA3	Paton	Edition pour entreprise	2013-08-16
6.1.1 GA2	Paton	Edition communautaire	2012-07-31
6.1.20 GA2	Paton	Edition pour entreprise	2012-07-31
6.1.10 GA1	Paton	Edition pour entreprise	2012-02-15
6.1.0 GA1	Paton	Edition communautaire	2012-01-01
6.0.12 SP2	Bunyan	Edition pour entreprise	2011-11-07
6.0.6	Bunyan	Edition communautaire	2011-03-04

Version	Nom de code	Type de libération	Date de sortie
6.0.11 SP1	Bunyan	Edition pour entreprise	2011-01-13
5.2 SP5	Augustin	Edition pour entreprise	2010-10-20
6.0.10	Bunyan	Edition pour entreprise	2010-09-10
6.0.5	Bunyan	Edition communautaire	2010-08-16
6.0.4	Bunyan	Edition communautaire	2010-07-23
6.0.3	Bunyan	Edition communautaire	2010-07-20
6.0.2	Bunyan	Edition communautaire	2010-06-08
5.2 SP4	Augustin	Edition pour entreprise	2010-05-19
6.0.1	Bunyan	Edition communautaire	2010-04-20
5.1 SP5	Calvin	Edition pour entreprise	2010-03-12
6.0.0	Bunyan	Edition communautaire	2010-03-04
5.2 SP3	Augustin	Edition pour entreprise	2010-01-07
5.2 SP2	Augustin	Edition pour entreprise	2009-11-17
5.1 SP4	Calvin	Edition pour entreprise	2009-10-23
5.2 SP1	Augustin	Edition pour entreprise	2009-08-07
5.1 SP3	Calvin	Edition pour entreprise	2009-07-20
5.2	Augustin	Edition pour entreprise	2009-06-01
5.2.3	Augustin	Edition communautaire	2009-05-12
5.1 SP2	Calvin	Edition pour entreprise	2009-05-12
5.2.2	Augustin	Edition communautaire	2009-02-26
5.1 SP1	Calvin	Edition pour entreprise	2009-02-18
5.2.1	Augustin	Edition communautaire	2009-02-03
5.2.0	Augustin	Edition communautaire	2009-01-26
5.1 SP	Calvin	Edition pour entreprise	2008-12-16
5.1.2	Calvin	Edition communautaire	2008-10-03

Version	Nom de code	Type de libération	Date de sortie
5.1.1	Calvin	Edition communautaire	2008-08-11
5.1.0	Calvin	Edition communautaire	2008-07-17
5.0.1 RC	Luther	Edition communautaire	2008-04-14
5.0.0 RC	Luther	Edition communautaire	2008-04-09

Examples

Une installation de base pour le développement et les tests

La dernière version de Liferay CE est simple:

1. Allez sur https://www.liferay.com/downloads .
2. Choisissez un paquet parmi ceux listés. Pour les débutants, le bundle Tomcat est un bon choix. Cliquez sur "Télécharger".

3. Décompressez le package de téléchargement chaque fois que vous le souhaitez. Le répertoire décompressé sera le répertoire `LIFERAY_HOME` .
4. Pour démarrer Liferay, exécutez simplement le script `LIFERAY_HOME/tomcat-x.xx.xx/bin/startup.sh` ; Seuls les environnements Windows exécutent le script `LIFERAY_HOME\tomcat-x.xx.xx\bin\startup.bat` .
5. Par défaut, une fois que Liferay est actif, un navigateur ouvre son URL locale (http: // localhost: 8080 /) .
6. Pour vous connecter, utilisez l'e-mail `test@liferay.com` et le `test` mot de passe.
7. Pour arrêter Liferay, exécutez simplement le script `LIFERAY_HOME/tomcat-x.xx.xx/bin/shutdown.sh` ; Seuls les environnements Windows exécutent le script `LIFERAY_HOME\tomcat-x.xx.xx\bin\shutdown.bat` .

Avec ces étapes, vous aurez Liferay opérationnel dans un mode "démo". Liferay utilisera un DB Hypersonic par défaut, mais il est impropre à la production. De plus, `test@liferay.com` est un compte administrateur avec un mot de passe par défaut, il doit donc être modifié à terme. Pourtant, ces étapes sont bonnes pour avoir une idée de l'apparence et du fonctionnement de Liferay.

Lire Commencer avec liferay en ligne: https://riptutorial.com/fr/liferay/topic/932/commencer-avec-liferay

Chapitre 2: Communication inter portlet

Introduction

Ce manuel contient les différentes manières dont le portlet peut coordonner ou communiquer entre eux et les différents scénarios pour lesquels une approche particulière est utilisée.

Remarques

Les références:

1. Param de rendu public
2. JSR 286 specs
3. Session de portlet

Examples

Utilisation du paramètre de rendu public

Cette approche a été introduite dans JSR 286.

Dans JSR 168, rendu paramètres définis dans *processAction* d'un portlet sont disponibles que dans cet portlet.With les paramètres de rendu publique disposent, les paramètres de rendu définis dans le *processAction* d'un portlet seront disponibles en rendre d'autres portlets also.In afin de configurer ceci, pour tous les portlets supportant ceci:

Ajouter la `<supported-public-render-parameter>` , juste avant la fin de la balise du portlet dans `portlet.xml`

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

Ajoutez la `<public-render-parameter>` juste avant la fin de la `<portlet-app>`

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

Dans la méthode `processAction` , la valeur du paramètre doit être définie dans la réponse

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

Après que nous en ayons fini avec la configuration de tout le portlet requis, après avoir exécuté la phase d'action du portlet concerné, le paramètre devrait être disponible en phase de rendu pour tous les portlets de la page, quelle que soit l'application.).

Utilisation de la session de portlet

C'est une approche qui existe depuis la JSR 168. Elle nous permet de partager des attributs en utilisant une session de portlet. Une session de portlet peut avoir différents types de portées:

1. Portée du portlet (attributs disponibles uniquement dans le portlet)
2. Champ d'application (attributs disponibles dans l'application entière [war])

Pour utiliser cette approche, il n'est pas nécessaire de créer des entrées dans la configuration de portlet, car la session de portlet est facilement disponible dans la demande de portlet:

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

ou

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

L'attribut ne peut être récupéré qu'à partir de la portée respective.Comme pour l'attribut défini dans la portée du portlet, nous devons le récupérer en utilisant

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

La principale limite de cette approche est le manque de partage entre autres portlet, en dehors de l' application scope.In pour y remédier, il est Liferay approche spécifique à ajouter `<private-session-attributes` > pour `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>
```

pour tous les portlets, où les attributs sont définis et récupérés.

Utilisation de la fonction événementielle

Le mécanisme d'événement est une version étendue du paramètre de rendu public, avec une fonctionnalité supplémentaire permettant de transmettre des objets personnalisés à d'autres portlets, mais avec une surcharge de la phase d'événement.

Pour ce faire, ce mécanisme consiste à

1. Portlet de l'éditeur
2. Portlet de processeur (consommateur), les deux pouvant faire partie de différentes applications de portlet.

Commencer avec,

Ajouter la `<supported-publishing-event>` au portlet `portlet.xml` dans `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>
```

Ajouter la `<supported-processing-event>` au portlet du processeur dans `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>
```

Ajoutez la `<event-definition>` aux deux portlets, en définissant le nom de l'événement et tapez dans `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>
```

Ensuite, nous devons créer une classe pour le type d'événement (dans le cas d'un type personnalisé)

```
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;
  }
```

}

Désormais, dans le portlet de l'éditeur, l'événement doit être publié dans la phase d'action

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

Post nous avons publié l'événement, il doit être traité par le portlet de l'éditeur dans la phase de l'événement.

La phase d'événement a été introduite dans JSR 286 et est exécutée avant la phase de rendu du portlet, le cas échéant

```
@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());
    }

}
```

qui peuvent ensuite être extraites du paramètre de rendu via la requête de rendu.

Lire Communication inter portlet en ligne:
https://riptutorial.com/fr/liferay/topic/8370/communication-inter-portlet

Chapitre 3: Configuration de SSL

Remarques

Assurez-vous d'avoir un certificat SSL valide fourni par un tiers. Vous pouvez également utiliser un certificat auto-signé, mais uniquement pour le développement. Letsencrypt fournit des certificats gratuits pouvant être utilisés dans la production ....

Utilisez keytool pour importer le certificat dans la chaîne de clés de java.

Examples

Comment activer SSL sur Tomcat et Liferay

Assurez-vous que votre fichier de configuration tomcat, server.xml a cette ligne:

```
<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"
               />
```

Il est important de choisir les bons protocoles ssl, vous pouvez ajouter plus de sslprotocols en séparant les virgules comme ceci:

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

Assurez-vous ensuite que votre fichier portal-ext.properties dans Liferay possède les lignes de configuration suivantes:

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

Lire Configuration de SSL en ligne: https://riptutorial.com/fr/liferay/topic/4320/configuration-de-ssl

Chapitre 4: Configurer le gestionnaire de tags Google (GTM) dans liferay

Introduction

Cette documentation n'est pas spécifique à liferay mais peut être utilisée en référence à n'importe quelle application Web.

Liferay fournit Google Analytics (appelé GA ahead) par défaut, après avoir configuré l'ID GA Analytics - ##### dans Paramètres du site. Mais cela fournit des fonctionnalités limitées, permettant uniquement de suivre les vues de page (titre de page et URL). étendre davantage, nous pouvons soit intégrer le script GA directement sur le thème du site pour déclencher les événements requis ou utiliser GTM.

Examples

Utilisation de GTM pour configurer les événements GA

GTM simplifie tout le processus de gestion des tags.Dans la terminologie GTM

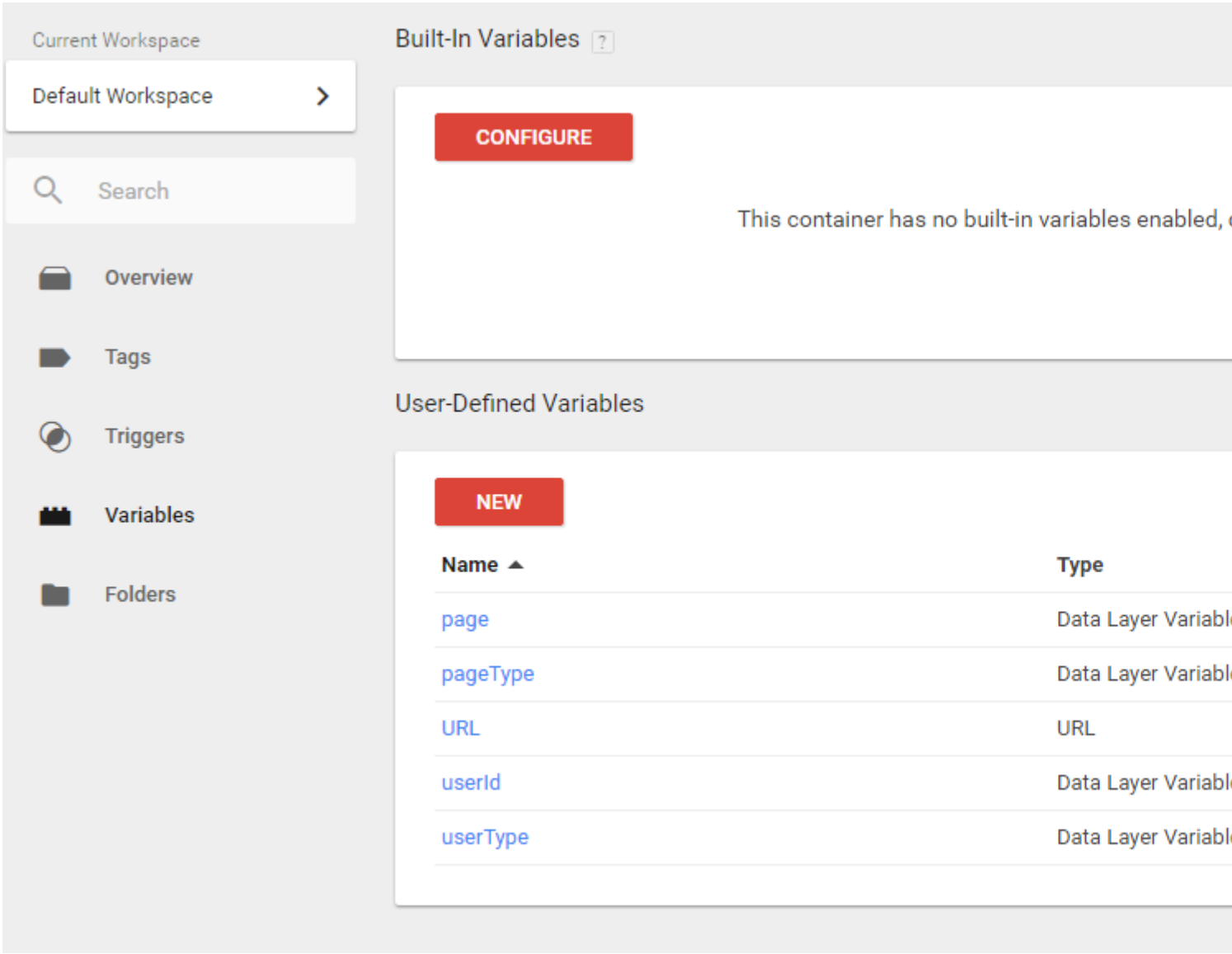
1. Nous avons mis un extrait de code javascript GTM sur la page concernée, dans portal_normal.vm dans le thème personnalisé de liferay, contenant l'ID GTM et une structure de couche de données (si nécessaire) pour mapper les valeurs d'une page à des variables.
2. Correspondant aux variables de la couche de données, nous devons créer des variables à la fin du GTM, qui récupèrent les données de la couche de données.
3. Par la suite, nous créons des balises, qui sont essentiellement des champs qui mappent les variables de la couche de données aux événements, qui sont déclenchés sous certaines conditions, conduisant à des événements envoyés aux outils de suivi respectifs (GA, dans notre cas).

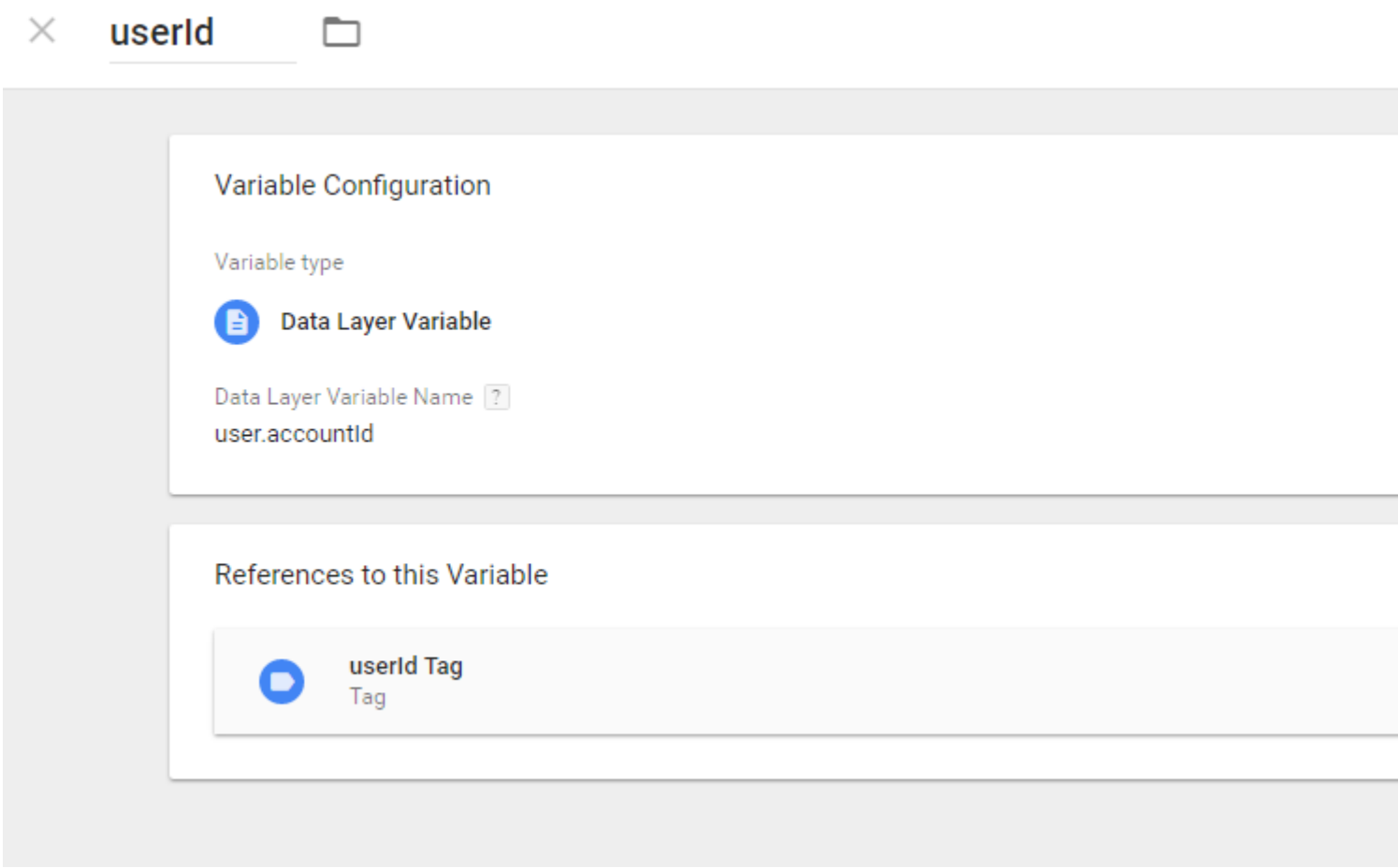
Vous trouverez ci-dessous un exemple d'extrait de code javascript de GTM intégré à une page.

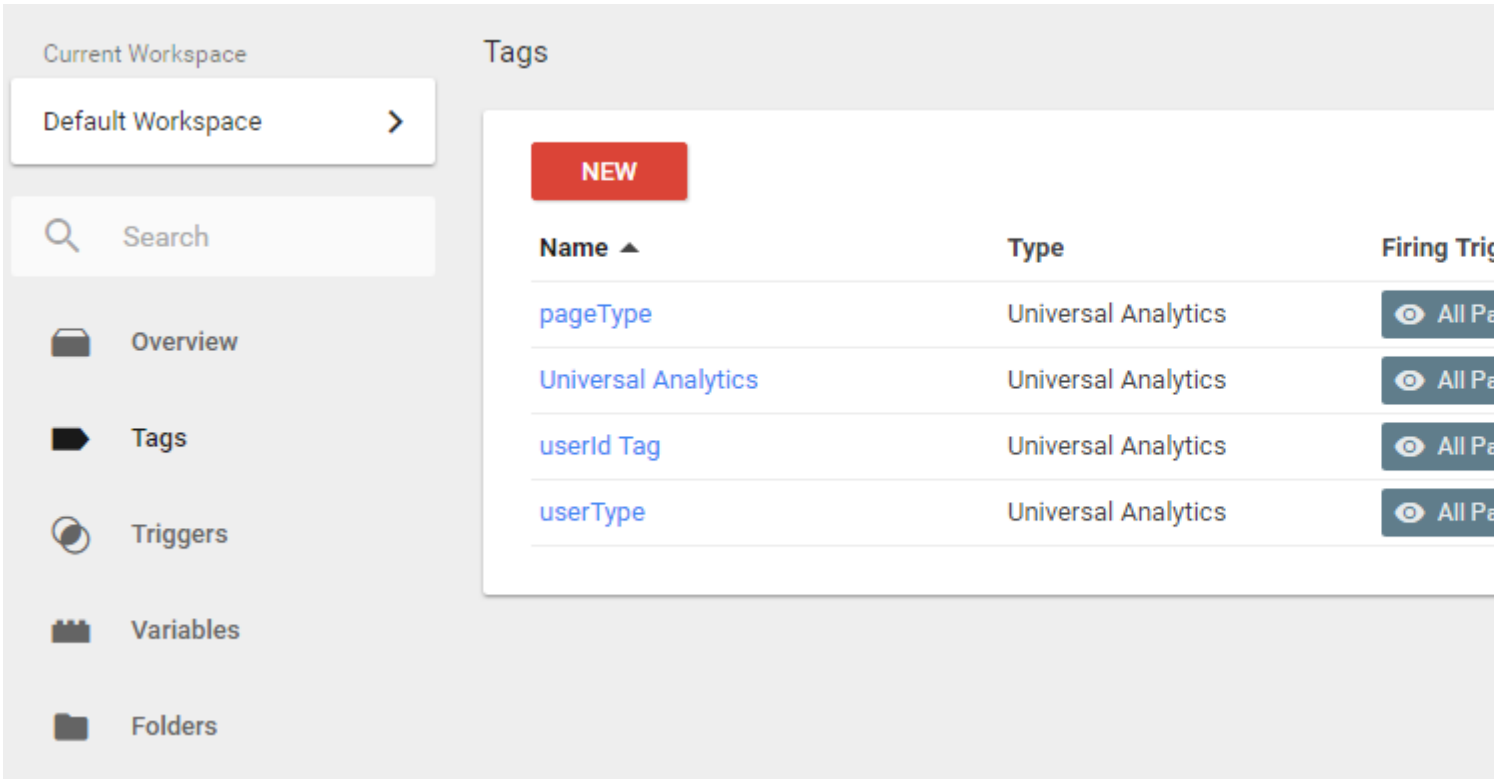
```
<body>
<!-- 1) Data layer section -->
<script type="text/javascript">
    dataLayer = [{
            "page" : "<? Virtual path of the page ?>"
            ,"pageType" : "<? Page type ?>"
            ,"user" : {
                        "type" : "<? User type ?>"
                        ,"userId" : "<? Logged user id ?>"
                        ,"country" : "<? Logged user country ?>"
                        ,"userRole" : "<? Role of user ?>"
                    }
        }];
  </script>
<!-- 2) GTM Container -->
```

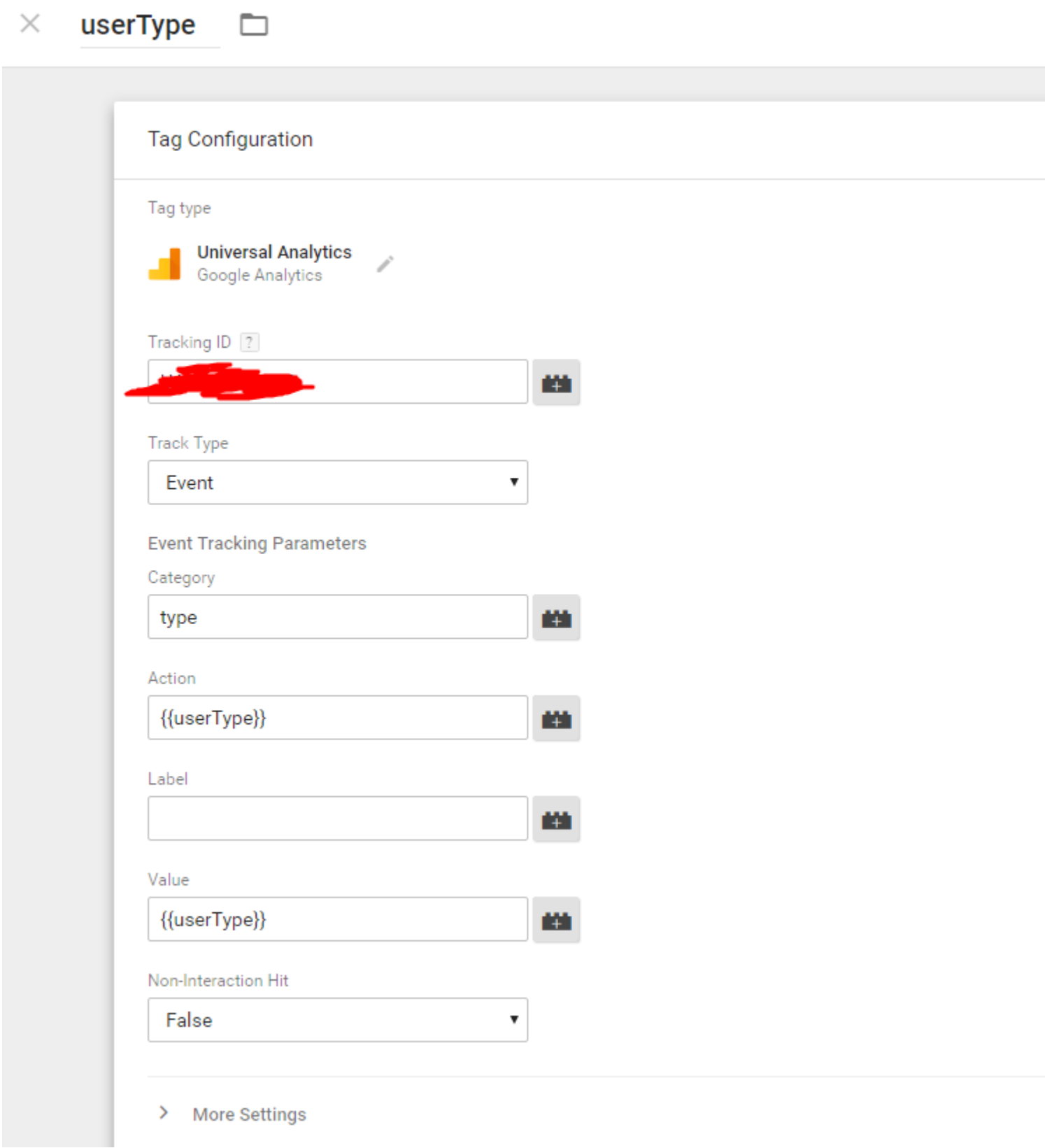
```
<noscript><iframe src="//www.googletagmanager.com/ns.html?id=GTM-PK9HK8"
height="0" width="0" style="display:none;visibility:hidden"></iframe></noscript>
<script>(function(w,d,s,l,i){w[l]=w[l]||[];w[l].push({'gtm.start':
new Date().getTime(),event:'gtm.js'});var f=d.getElementsByTagName(s)[0],
j=d.createElement(s),dl=l!='dataLayer'?'&l='+l:'';j.async=true;j.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

Après avoir inclus ce script dans la page, nous devons configurer les variables et les balises respectives à la fin de GTM.

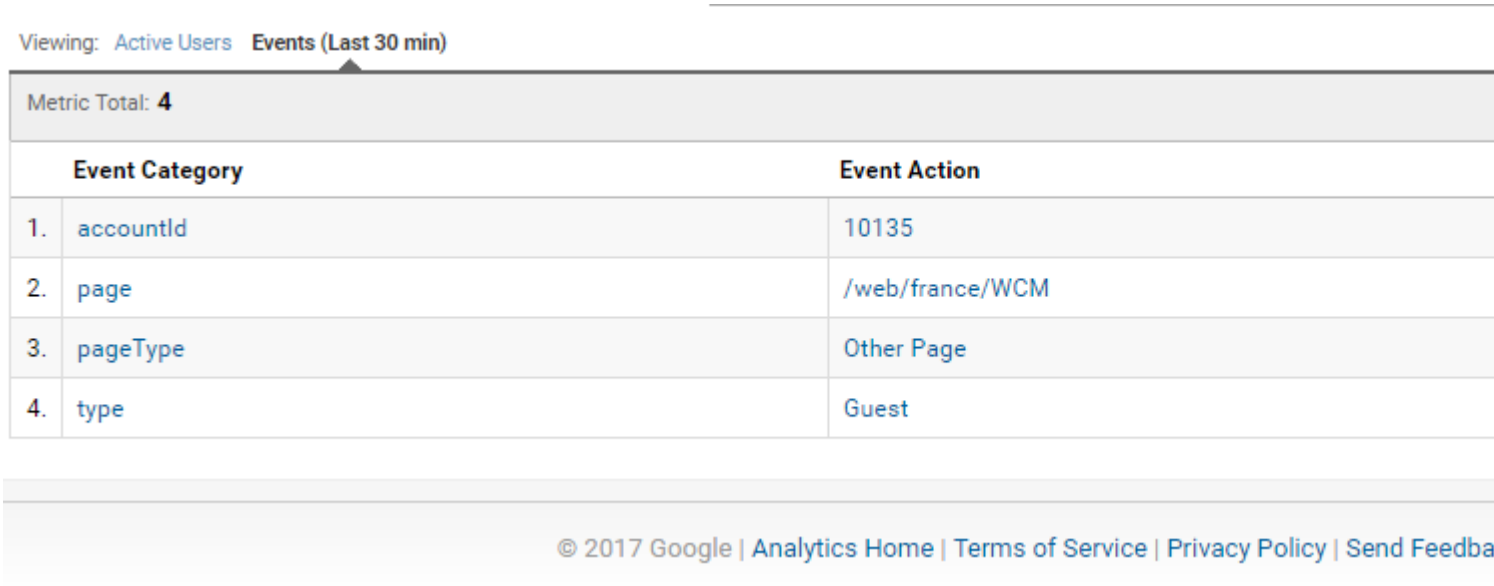



userId
Variable Configuration
Variable type
Data Layer Variable
Data Layer Variable Name
user.accountId
References to this Variable
userId Tag
Tag


Current Workspace
Default Workspace
Search
Overview
Tags
Triggers
Variables
Folders
Tags
NEW
Name
Type
Firing Trig
pageType
Universal Analytics
Universal Analytics
Universal Analytics
userId Tag
Universal Analytics
userType
Universal Analytics



Post nous avons configuré les champs requis, nous pouvons afficher les événements sur la console GA sur une vue utilisateur.



Viewing: Active Users **Events (Last 30 min)**

Metric Total: **4**

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

© 2017 Google | Analytics Home | Terms of Service | Privacy Policy | Send Feedba

Pour afficher les données envoyées du portail à GA, nous pouvons utiliser le plug-in Google Analytics Debugger pour afficher les événements envoyés à GA via la console du navigateur.

Lire Configurer le gestionnaire de tags Google (GTM) dans liferay en ligne: https://riptutorial.com/fr/liferay/topic/8723/configurer-le-gestionnaire-de-tags-google--gtm--dans-liferay

Chapitre 5: Créer un planificateur Quartz en mode de vie

Remarques

Un planificateur permet d'effectuer des tâches en arrière-plan à certains intervalles définis.

Selon la DTD du portlet Liferay

<! - L'élément scheduler-entry contient les données déclaratives d'un planificateur. ->

! ELEMENT scheduler-entry (scheduler-description?, Scheduler-event-listener-class, trigger)

<! - La valeur de description du planificateur décrit un planificateur. ->

! ELEMENT scheduler-description (#PCDATA)

<! - La valeur scheduler-event-listener-class doit être une classe qui implémente com.liferay.portal.kernel.messaging.MessageListener. Cette classe recevra un message à un intervalle régulier spécifié par l'élément déclencheur. ->

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

<! - L'élément trigger contient des données de configuration pour indiquer quand déclencher la classe spécifiée dans scheduler-event-listener-class. ->

! ELEMENT trigger (cron | simple)

Examples

Créer un planificateur de quartz pour afficher des informations

Pour créer un planificateur, l'entrée doit être créée dans

```
liferay-portlet.xml
```

vérification de la classe du planificateur et de la valeur de déclenchement pour la synchronisation du déclenchement du planificateur

```
<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>
```

L'entrée donnée fournit

1. Description du planificateur
2. Nom de classe, qui implémente la classe MessageListener
3. Déclencheur, qui fournit des intervalles pour définir le point de déclenchement du planificateur

   -Utiliser Cron

   -Utilisation de la valeur de déclenchement simple

Dans l'exemple donné, le planificateur se déclenchera toutes les 5 minutes.

Ensuite, nous devons créer une classe de planificateur

```
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");
        }

    }

}
```

Ce planificateur affiche simplement le nombre d'utilisateurs du portail de sortie après chaque intervalle de déclenchement sur la console du serveur.

Créer un planificateur de quartz dynamique par programmation

Il existe des scénarios spécifiques dans lesquels nous pourrions avoir besoin de créer un ordonnanceur Quartz, basé sur les entrées des utilisateurs sur le moment où un ordonnanceur doit être déclenché, à part que nous pouvons gérer des cas, avec certaines fonctionnalités prédéfinies qui doivent être déclenchées action de l'utilisateur, à une certaine période.

Cet exemple reçoit une entrée utilisateur au moment du déclenchement, pour déclencher un scheduler.Here `ScheduledJobListener` class `imlements MessageListener` , qui contient la logique métier à exécuter lors du déclenchement du planificateur. Le travail est planifié à l'aide de `SchedulerEngineHelperUtil` classe `SchedulerEngineHelperUtil`

1. Déclencheur (en utilisant la chaîne de texte cron et le nom du travail)
2. Message (utilisant l'implémentation pour la classe MessageListener et portletId)
3. Types de stockage du planificateur (qui est MEMORY_CLUSTERED par défaut, peuvent être définis comme PERSISTED pour être stockés dans la base de données)
4. DestinationNames (qui est SCHEDULER_DISPATCH pour Liferay) qui décide de la destination du Message Bus à utiliser

L'extrait de code ci-dessous fait partie de la phase d'action du portlet qui interagit avec l'utilisateur pour créer et planifier un travail quartz.

```
   //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();
                }
```

Ici, pour créer du texte cron, les paramètres sont récupérés à partir de l'entrée utilisateur. Pour le texte cron, nous pouvons également utiliser la référence donnée pour créer le modèle 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.
```

et créer directement une chaîne de crontext à utiliser en fonction de l'entrée de l'utilisateur

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

Ici, dans ce cas, il se déclenche toutes les cinq minutes.

Les références:

1. Création de planificateur dynamique
2. Application de planification
3. FAQ sur le quartz

Lire Créer un planificateur Quartz en mode de vie en ligne:
https://riptutorial.com/fr/liferay/topic/7998/creer-un-planificateur-quartz-en-mode-de-vie

Chapitre 6: Crochets à Liferay

Remarques

Cela fonctionne avec Liferay Portal jusqu'à la version 6.2.

Examples

Crochet JSP

Les hooks JSP sont un plugin spécial permettant de modifier le portlet principal jsp-s, disons que vous souhaitez modifier le portlet de connexion pour afficher `Welcome in my custom login!` .

La structure minimale pour un plug-in Hook est la suivante:

```
[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` est le fichier qui distingue le type de hook que vous utilisez, ici vous définissez le paramètre approprié pour le hook dans le hook, pour le hook 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 trouve dans Liferay dans `/docroot/html/portlet/login/login.jsp` , pour en faire un crochet, nous devons ajouter un fichier jsp avec le même nom et le même chemin dans notre dossier `custom_jsps` .

Lorsque le hook est déployé, Liferay recherchera la `liferay-hook.xml custom-jsp-dir` dans le `liferay-hook.xml` et remplacera toutes les pages JSP du portail par celles trouvées dans ce répertoire. Les fichiers jsp d'origine sont enregistrés avec le nom `<orginal name>.portal.jsp` pour être restaurés en cas de non- `<orginal name>.portal.jsp` du hook.

Nous pouvons même appeler les JSP d'origine dans le nouveau JSP modifié si nous voulons garder le code adaptable aux mises à jour ou aux mises à niveau de la version de la plateforme

Liferay sous-jacente. Pour ce faire, utilisez le modèle suivant dans votre JSP personnalisé:

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

où `{ JSP file's path }` dans ce cas sera `portlet/login/login.portal.jsp` . Cette opération s'appelle l' *extension du fichier jsp d'origine* .

Ensuite, nous pouvons ajouter du contenu avec:

```
<%
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 %>
```

Crochets d'Action Struts

Ce type de crochet peut être utilisé pour remplacer portail de base (par exemple `c/portal/login`) et portlet actions entretoises (par exemple `/login/forgot_password`), ces actions pour le portail Liferay sont spécifiées dans un `struts-config.xml` fichier dans son `WEB-INF` Dossier `WEB-INF` Pour remplacer une action:

1. `liferay-hook.xml` fichier `liferay-hook.xml` de votre plugin hook sous `docroot/WEB-INF` , ajoutez un élément `docroot/WEB-INF struts-action` dans l'élément hook.
2. Dans l'élément `struts-action` , ajoutez `struts-action-path` qui spécifie le chemin d'action que vous remplacez et `struts-action-impl` qui spécifie votre classe d'action personnalisée. Cela ressemble à:

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

3. Créez une classe d'action de portlet Struts qui étend `BaseStrutsPortletAction` . Un exemple de cette classe est:

```
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);
    }
}
```

L'appel de la méthode en cours de substitution, comme `originalStrutsPortletAction.processAction`, n'est pas obligatoire mais constitue une bonne pratique pour que le comportement de l'action reste inchangé par rapport au portail Liferay. Ce type de hook peut être utilisé pour ajouter de nouvelles actions Struts, c'est la même chose que de modifier une action existante, dans ce cas, `liferay-hook.xml` serait:

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

Bonjour utilisateur "Nom" avec crochets

Cet exemple montre comment créer un simple "Hello User [name]" après la connexion. L'exemple est basé sur l' exécution d'une action personnalisée à l'aide d'un hook

Depuis votre terminal de ligne de commande, accédez au dossier des hooks de votre plug-in SDK. Pour créer un projet hook, vous devez exécuter le script de création. Voici le format à suivre pour exécuter le script:

> créer. [sh | bat] [nom-projet] "[Nom d'affichage du crochet]"

Sous Linux et Mac OS X, vous entrez une commande similaire à celle de cet exemple:

> ./create.sh Bonjour utilisateur "Bonjour utilisateur"

Sous Windows, vous entrez une commande similaire à celle de cet exemple:

> create.bat Bonjour à l'utilisateur "Mon crochet"

L'assistant New Project de Liferay IDE et les scripts de création génèrent des projets hook dans le dossier des hooks de votre SDK Plugin. Le plug-in SDK ajoute automatiquement «-hook» au nom de votre projet.

Que vous ayez créé votre projet hook à partir de l'EDI de Liferay ou de la ligne de commande, vous obtenez la même structure de projet (voir ci-dessus).

- Déterminez l'événement sur lequel vous souhaitez déclencher votre action personnalisée. Recherchez dans la documentation **portal.properties** la propriété d'événement

correspondante. Indice: les propriétés de l'événement ont pour nom .event. Il existe des propriétés d'événement de session, de démarrage, d'arrêt et de portail dans les sections suivantes de la documentation **portal.properties** : Session - Événements de démarrage - Événements d'arrêt - Événements du portail

- Dans votre projet hook, créez une classe Java qui étend la **classe com.liferay.portal.kernel.events.Action.** Remplacez la **méthode 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());
    }
}
```

Important: Si votre action accède à l'objet HttpServletRequest, étendez com.liferay.portal.kernel.events.Action; sinon, étendez com.liferay.portal.struts.SimpleAction.

- Créez un fichier de propriétés, **portal.properties** , dans le dossier docroot / WEB-INF / src de votre projet hook. Ajoutez ensuite le nom de la propriété d'événement de portail qui correspond à l'événement sur lequel vous souhaitez effectuer votre action. Spécifiez le nom complet de votre classe d'action comme valeur de la propriété.

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

Par exemple, pour effectuer une action de classe juste avant la connexion du portail à un utilisateur, vous devez spécifier la propriété login.events.pre avec votre classe d'action comme valeur. Cela pourrait ressembler à ce paramètre de propriété.

Important: les propriétés du portail telles que login.events.pre acceptant plusieurs valeurs, vous devez ajouter vos valeurs aux valeurs existantes. Vous pouvez modifier les propriétés à plusieurs reprises à partir de crochets supplémentaires.

Modifiez uniquement une propriété de portail qui accepte une valeur unique à partir d'un plug-in à un seul crochet. Si vous modifiez la valeur d'une propriété à partir de plusieurs plug-ins, Liferay ne saura pas quelle valeur utiliser.

- Editez votre **fichier docroot / WEB-INF / liferay-hook.xml** et ajoutez votre nom du fichier de propriétés du portail du crochet comme valeur pour le Élément `<portal-properties>...</portal-properties>` dans votre élément de crochet `<hook>...</hook>` . Par exemple, si le nom du fichier de propriétés de votre hook est **portal.properties** , vous devez spécifier cet élément:

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

- Déployez votre hook, accédez à votre chemin d'accès et entrez `ant clean deploy` vous verrez le fichier .war dans le dossier dist.

Maintenant, si vous vous connectez à liferay, vous verrez dans le journal du serveur un message du type "Hello user Admin".

Crochet d'écoute modèle

Contexte

Les hooks de Listener Model sont un type de plugin Liferay qui écoute les événements pris sur un modèle et exécute le code en réponse. Les hooks d'écouteur de modèle sont similaires aux hooks d'action Struts personnalisés car ils répondent à une action effectuée dans le portail. Cependant, alors que les actions Struts répondent à une action prise par un utilisateur, un écouteur de modèle répond (avant ou après) à un événement impliquant un modèle Liferay.

Différences

Voici quelques exemples d'actions Struts v.

- **Action Struts**
  - Utilisateur en ligne
  - Création de compte
  - Prolonger la session
  - Déplacer le dossier
- **Modèle auditeur**
  - Après la création du dossier
  - Lorsque les informations utilisateur sont mises à jour
  - Après la suppression du signet
  - Avant qu'une association de rôle ne soit créée

  La meilleure ressource pour apprendre l'architecture de Liferay est leur code source. Tous leurs fichiers source sont situés sur GitHub et en affichant leurs JavaDocs. Vous pouvez voir tous les modèles de portail principaux sur les JavaDocs et toutes les actions Struts sur GitHub.

Exemple

Dans ce tutoriel, nous allons développer un écouteur de modèle qui envoie un courrier électronique à un utilisateur après la création de son compte. Pour ce faire, nous allons écrire une

classe appelée **UserModelListener** qui étendra **BaseModelListener** de Liferay. Nous allons brièvement passer en revue la création de hook et couvrirons les modifications nécessaires aux fichiers de configuration suivants

- **portal.properties**
- **liferay-hook.xml**

Commencer

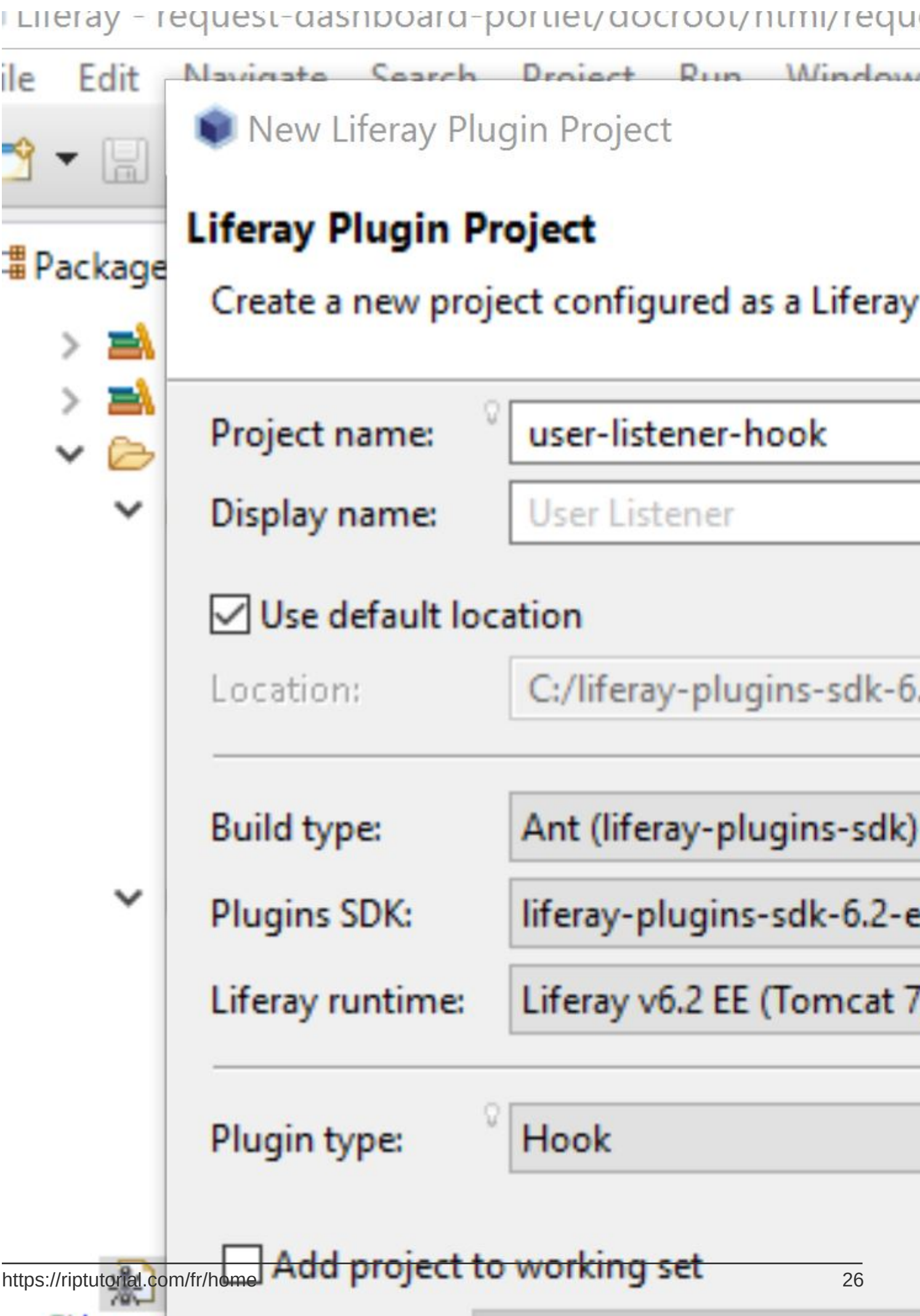
Pour commencer à développer votre hook Model Listener, vous devez d'abord lancer votre application Liferay IDE ou Liferay Developer Studio.

> Liferay IDE et Liferay Developer Studio sont tous deux des environnements de développement **Eclipse** personnalisés. Ils sont étonnamment similaires et un ensemble de directions devrait être suffisant pour les deux environnements.

Dans votre environnement de développement, exécutez les étapes suivantes.

1. Dans le coin supérieur gauche, cliquez sur **Fichier**
2. Passez votre souris sur **Nouveau**
3. Cliquez sur **Liferay Plugin Project**

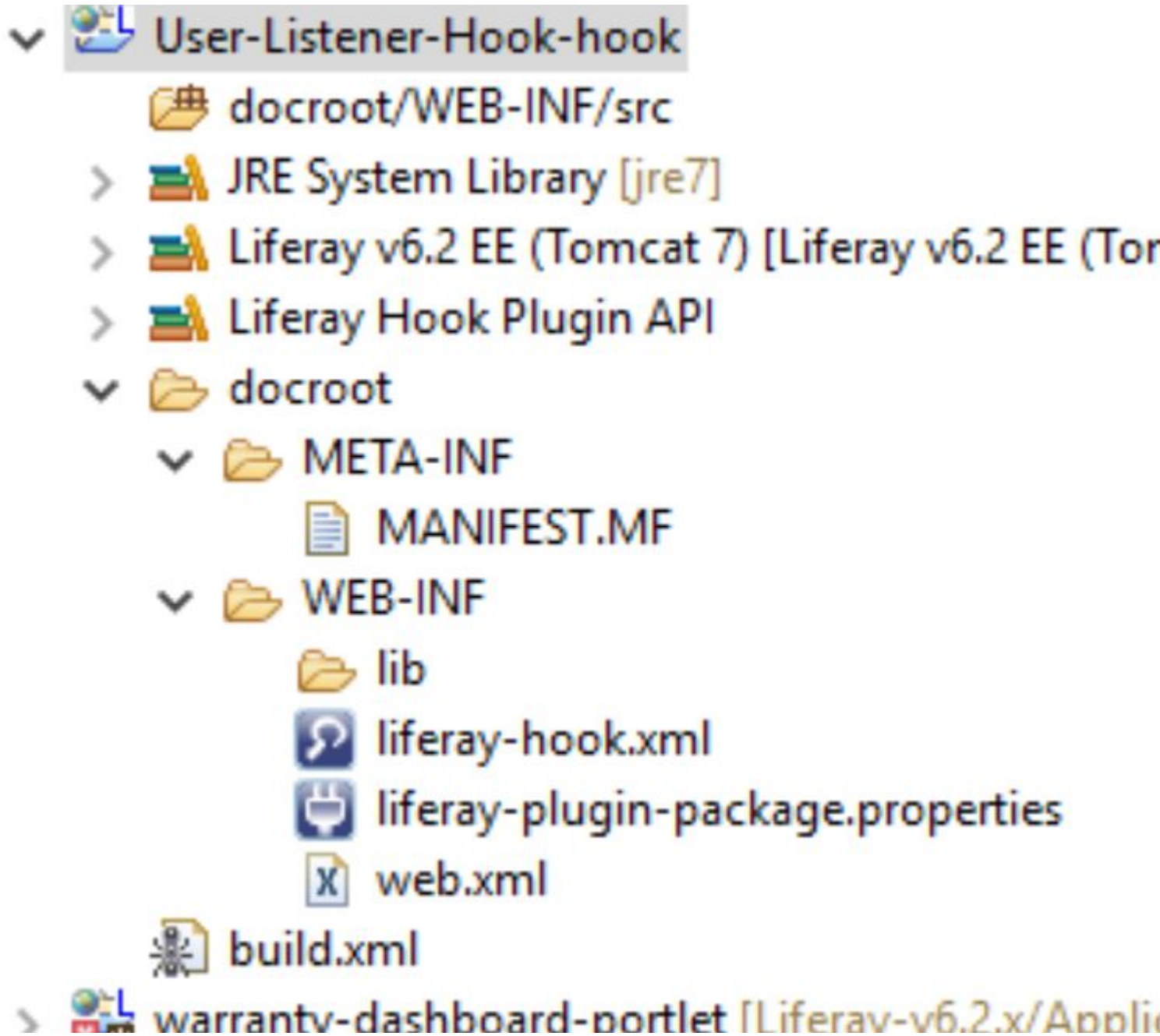
Vous allez engendrer cette fenêtre.


Liferay - request-dashboard-portlet/docroot/html/requ
ile Edit Navigate Search Project Run Window
New Liferay Plugin Project
Liferay Plugin Project
Create a new project configured as a Liferay
Package
Project name:
user-listener-hook
Display name:
User Listener
Use default location
Location:
C:/liferay-plugins-sdk-6.
Build type:
Ant (liferay-plugins-sdk)
Plugins SDK:
liferay-plugins-sdk-6.2-e
Liferay runtime:
Liferay v6.2 EE (Tomcat 7
Plugin type:
Hook
Add project to working set

- Sélectionnez **Utiliser l'emplacement par défaut**
- Type de construction: **Ant**
- Type de plugin: **Crochet**

  Assurez-vous que votre projet se trouve dans le **répertoire SDK** Hook de **Liferays Plugins** . Vous devrez sélectionner votre **SDK** et votre **runtime en** conséquence.

Dans votre perspective **Explorateur de packages** , vous verrez la structure de répertoires suivante.



Développement de l'auditeur

Maintenant que vous avez créé votre hook, vous devrez créer votre classe **UserModelListener**

personnalisée. Cette classe étendra la classe BaseModelListener de Liferay.

> La classe BaseModelListener de Liferay est une classe abstraite qui implémente l'interface ModelListener. Vous ne souhaitez pas implémenter directement l'interface ModelListener, car vous devrez remplacer toutes ses méthodes.

Les méthodes suivantes vous sont fournies par l'interface **ModelListener** via la classe abstraite **BaseModelListener** .

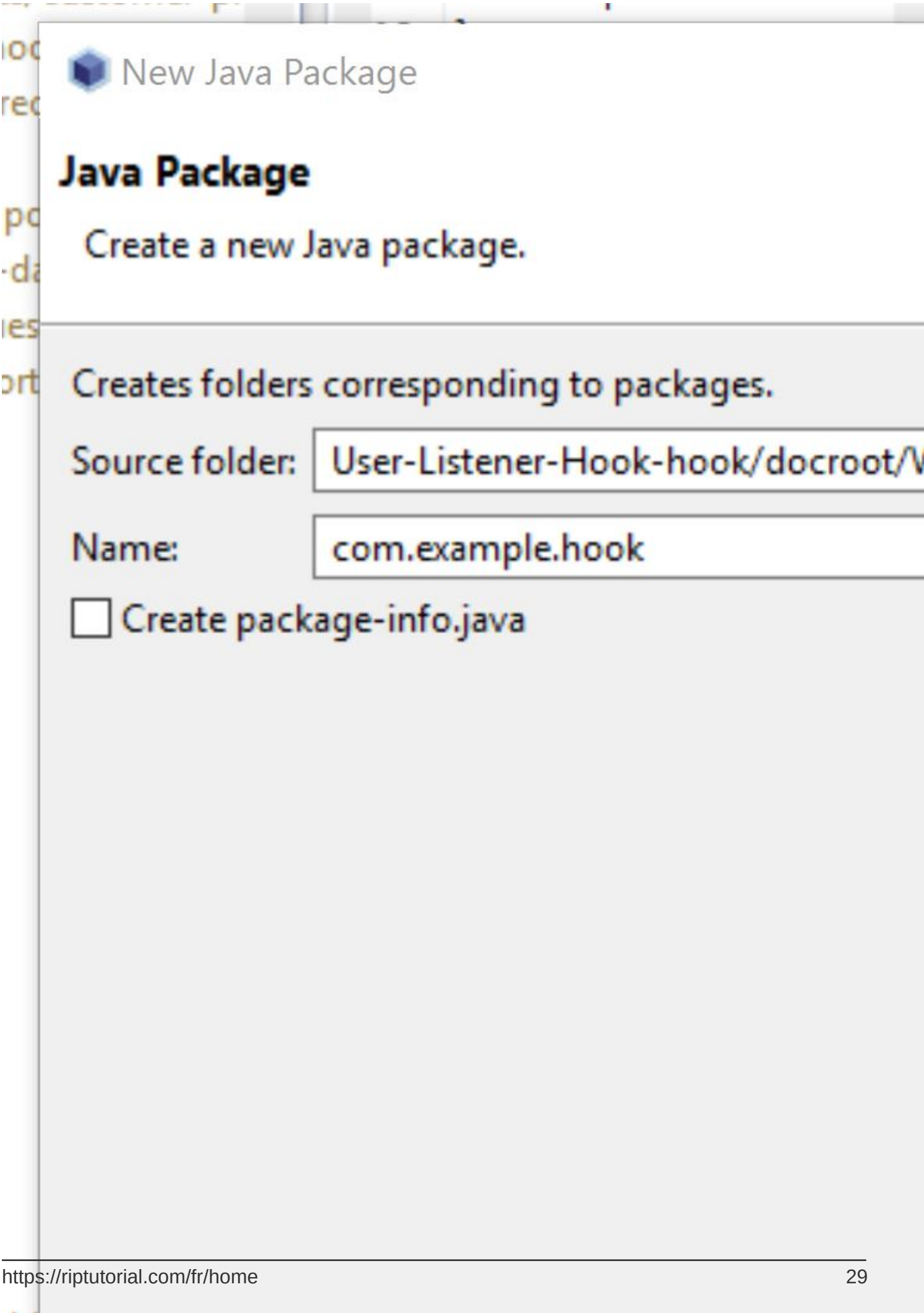
- onAfterAddAssociation
- onAfterCreate
- onAfterRemove
- onAfterRemoveAssociation
- onAfterUpdate
- onBeforeAddAssociation
- onBeforeCreate
- onBeforeRemove
- onBeforeRemoveAssociation
- onBeforeUpdate

Créez votre classe **UserModelListener** dans le répertoire suivant. Pour créer la classe via l'interface graphique, exécutez simplement les commandes suivantes

- Cliquez sur **Fichier** dans le coin supérieur gauche
- Passez votre souris sur **Nouveau**
- **Classe de** clic

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

Entrez les informations ci-dessous


New Java Package
Java Package
Create a new Java package.
Creates folders corresponding to packages.
Source folder:
User-Listener-Hook-hook/docroot/
Name:
com.example.hook
Create package-info.java

emballé à l' intérieur **com.example.code,** pour le modèle **DLFolder** nous aurions la propriété suivante

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

Enfin, localisez votre fichier liferay-hook.xml. Dans la vue **Source** , écrivez ce qui suit.

```
<?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>
```

Explication

1. La première ligne est un **prologue** facultatif qui spécifie la version du document et (dans certains cas) le jeu de caractères.
2. La ligne 2 est une **définition de type de document** formelle (DTD) qui définit explicitement quels éléments et attributs sont valides
3. Les lignes 3 et 5 sont constituées de l' **élément Hook** parent (l'un des éléments valides pris en charge par cette DTD)
4. La ligne 4 remplace et étend le fichier **portal.properties** dans **$ {liferay.home}**

   Pour voir quels autres éléments peuvent être utilisés dans ce fichier XML, vous pouvez référencer l'URL dans la définition DocType . Ceci est standard pour tous **les** fichiers **XML** et **SGML** avec une **DTD** . **Service.xml** (l'implémentation **ORM** de Liferay basée sur **Hibernate**) est un autre exemple de fichier **XML Liferay** avec une **DTD** .

Construire et déployer

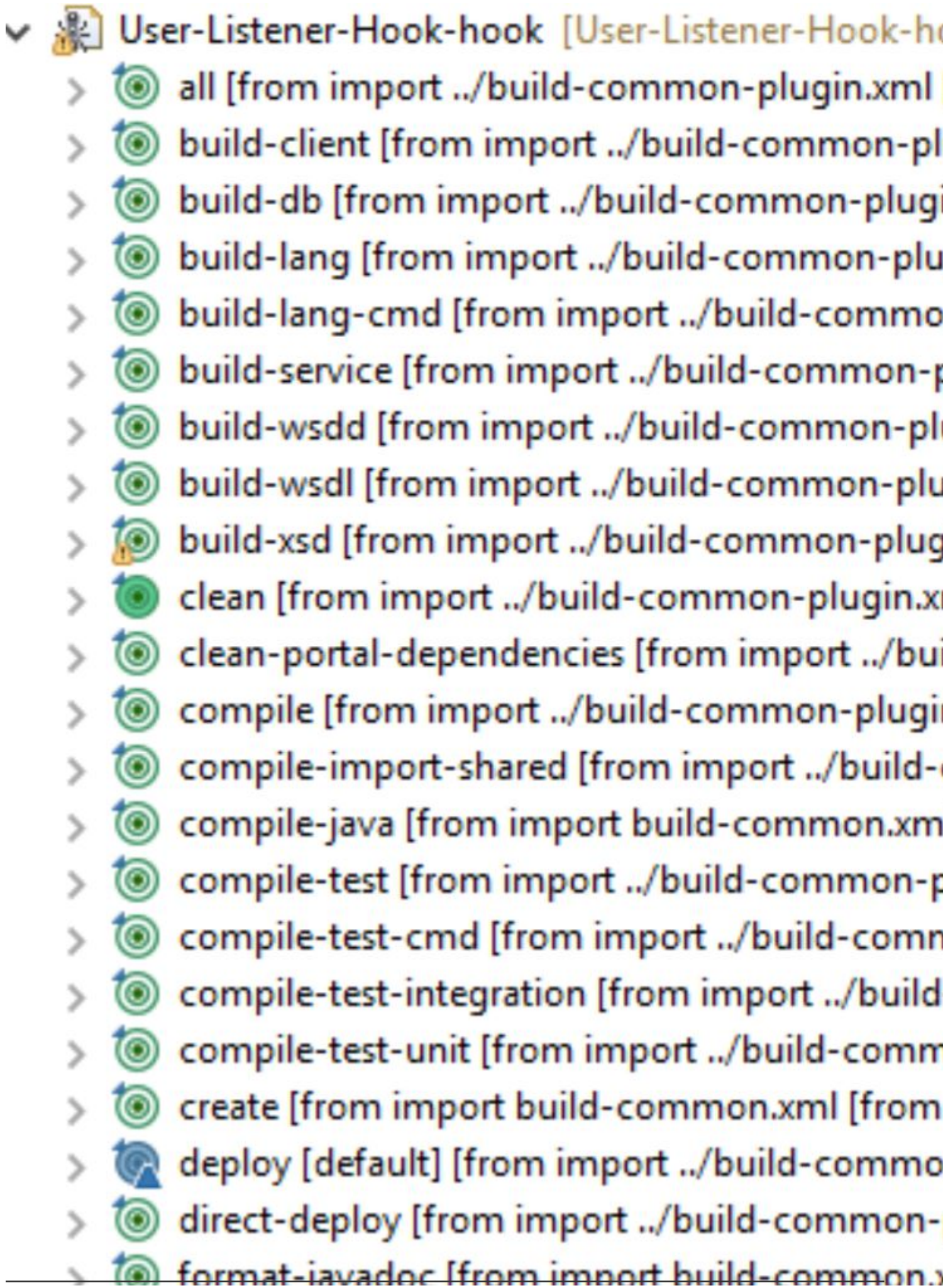
Construire et déployer des hooks est un processus simple. Le développement de plug-ins Liferay prend en charge l'automatisation de la construction et des dépendances avec

- Fourmi
- Lierre
- Maven
- Gradle

Dans notre exemple, nous avons utilisé **Ant** pour l'automatisation de la construction. Le **fichier build.xml** contient les commandes de génération (appelées **cibles** dans **Ant**). Pour construire votre hook, exécutez simplement les commandes suivantes.

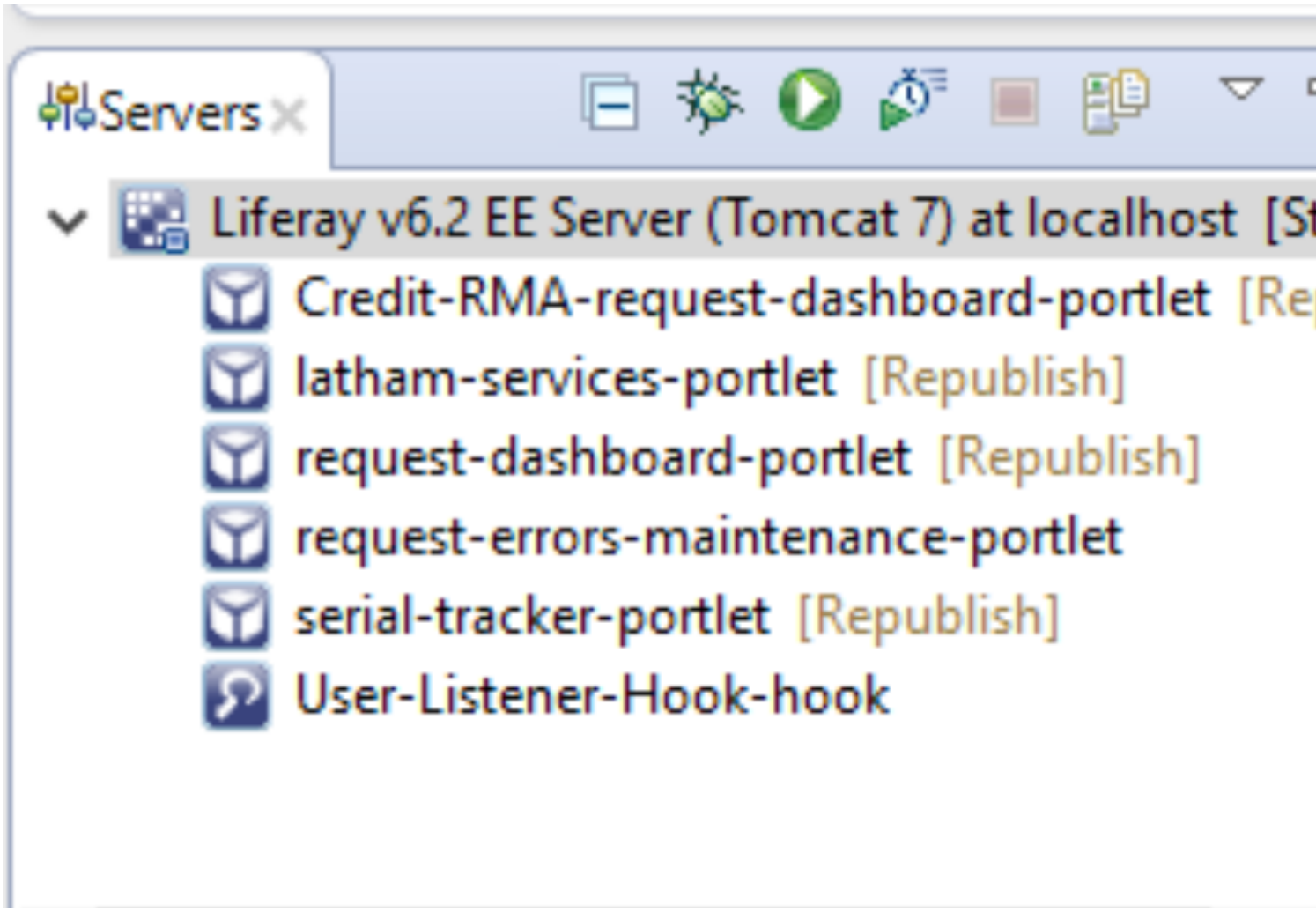
1. Emplacement de votre **fichier build.xml**
2. Dans votre IDE, faites glisser le **fichier build.xml** dans la perspective **Ant**

3. Développez le fichier et exécutez la cible **all**



Dans la vue de votre console, vous devriez voir quelque chose de similaire à ce qui suit

2. Localisez **User-Listener-Hook** sous la sélection **Disponible**
3. Une fois en surbrillance, cliquez sur le bouton **Ajouter** et cliquez sur **OK.**
4. Cliquez sur le bouton **Lire** dans la perspective du **serveur**



S'il vous plaît laissez-moi savoir si vous avez des questions, des commentaires, des préoccupations, etc. Tous les commentaires constructifs sont grandement appréciés!

Lire Crochets à Liferay en ligne: https://riptutorial.com/fr/liferay/topic/3712/crochets-a-liferay

Chapitre 7: Déboguer le serveur Layeray distant via Eclipse

Examples

Déboguer le serveur liferay distant via Eclipse (sans le plugin Eclipse du connecteur IDE Liferay Remote)

Pour déboguer une instance de serveur, démarrez en mode débogage. Pour ce faire, configurez ces paramètres à transmettre au serveur:

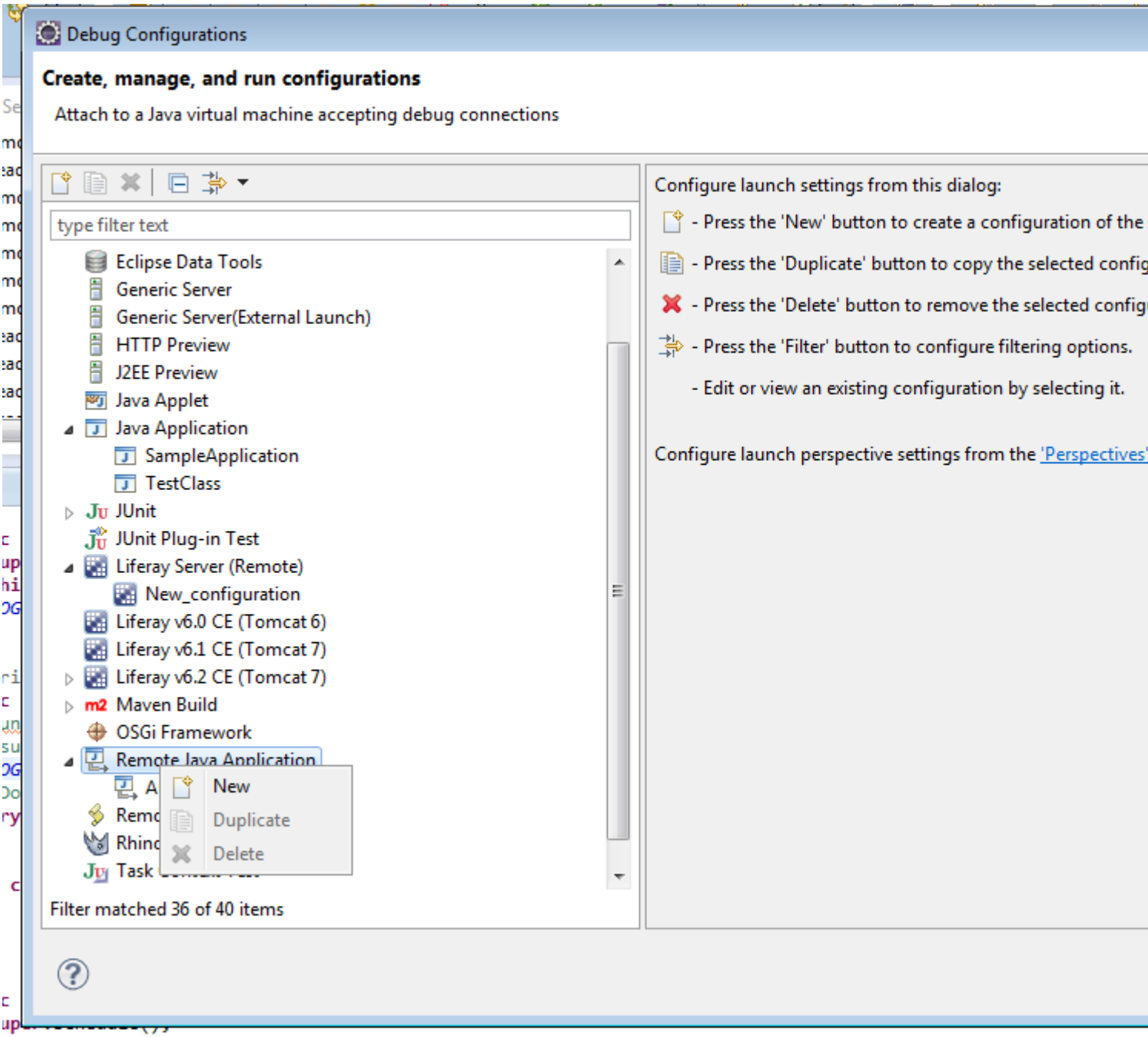
```
-Xdebug -Xrunjdwp:transport=dt_socket,address=8000,server=y,suspend=n
```

à setenv.bat (Windows) ou setenv.sh (Unix)

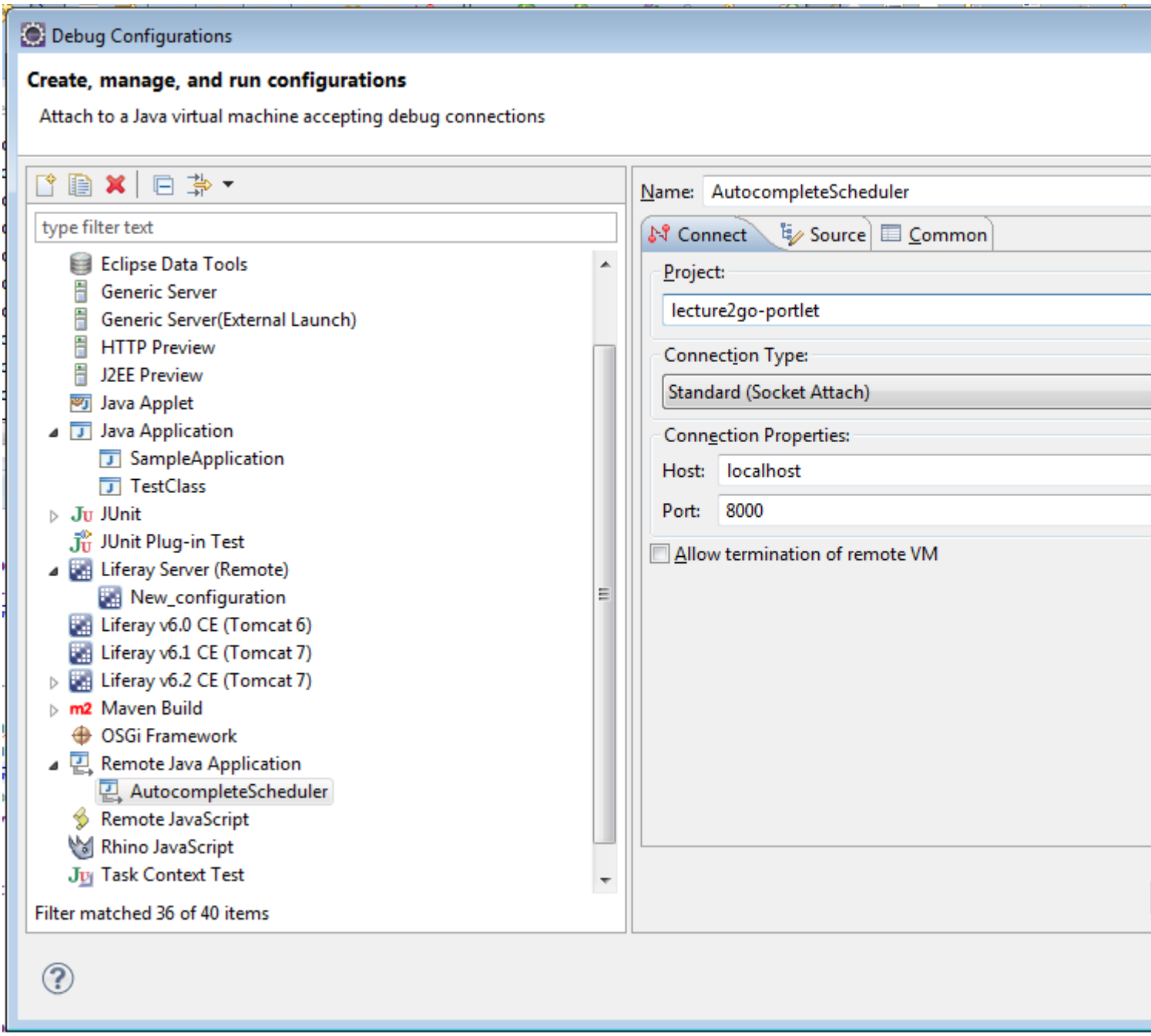
Celles-ci initialisent le serveur en mode débogage et écoutent les requêtes de débogage sur le port donné. Démarrez le serveur et publiez la configuration.

Dans eclipse, la configuration de débogage à distance doit être configurée pour attacher la source au serveur distant. Suivez les étapes indiquées:

1. Allez dans *Exécuter-> Configurations de débogage-> Application Java distante* :



2. Créez une nouvelle configuration à partir de l'application Java distante:



3. Entrez les détails donnés:

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

4. Cliquez sur *Déboguer* pour attribuer des pièces jointes à l'instance du serveur.

Lire Déboguer le serveur Layeray distant via Eclipse en ligne:
https://riptutorial.com/fr/liferay/topic/7891/deboguer-le-serveur-layeray-distant-via-eclipse

Chapitre 8: Déploiement d'un plugin

Examples

Déploiement sur Glassfish

Donc, vous créez d'abord un fichier .war, disons un portlet nommé `<YOUR PLUGIN>.war` . Vous voulez le faire fonctionner sur un domaine Glassfish sous le portail Liferay.

Les étapes du succès:

1. Naviguez vers le panneau de configuration -> installation de plugins sur Liferay
2. Hit **Installer de nouveaux portlets**
3. Hit **Configuration**
4. Remplissez le répertoire de déploiement un nouvel endroit pour le déploiement, disons `<YOUR DOMAIN>/autodeploy2`
5. Vérifiez que la cible de la ligne suivante est `<YOUR DOMAIN>/autodeploy` (c'est le répertoire de déploiement par défaut de Glassfish)
6. Hit **Enregistrer**

Maintenant, le déploiement se fera en copiant les fichiers de collage dans ce nouveau répertoire `<YOUR DOMAIN>/autodeploy2` . Le reste est géré automatiquement. Le réglage prend des mesures immédiatement.

Fait avec le déploiement: Faire un gabarit de victoire et en profiter :)

..vous arrêtez de danser et faites face à un bug. Vous voulez une nouvelle révision à déployer. Dans ce cas, continuez à lire.

Ainsi, vous avez de nouveau construit votre guerre et souhaitez vous redéployer. Procédez comme suit:

1. `<YOUR DOMAIN>/autodeploy` le `<YOUR DOMAIN>/autodeploy` anciens éléments du dossier `<YOUR DOMAIN>/autodeploy` en supprimant le fichier war. Ne supprimez aucun autre fichier.
2. Le résultat est que le fichier `<YOUR PLUGIN>.war_UnDeployed` apparaîtra.
3. déployer un nouveau fichier en copiant la nouvelle guerre dans le dossier `<YOUR DOMAIN>/autodeploy2` .
4. Le résultat est que `<YOUR PLUGIN>.war_deployed` apparaîtra dans le dossier `<YOUR DOMAIN>/autodeploy` .

Faites une nouvelle danse :)

Lire Déploiement d'un plugin en ligne: https://riptutorial.com/fr/liferay/topic/1708/deploiement-d-un-plugin

Chapitre 9: Utilisation d'une requête SQL dynamique et personnalisée dans Liferay

Introduction

Il existe des scénarios dans le traitement de la couche de service dans liferay, lorsque nous devons interroger la base de données avec trop de clauses ou traiter plusieurs tables. Dans de tels cas, nous utilisons soit:

1) Requête dynamique (wrapper sur l'API critères Hibernate)

2) Requêtes SQL personnalisées

Remarques

Les références:

1. SQL personnalisé
2. Requête dynamique

Examples

Utilisation de la requête dynamique dans Liferay

Pour la plupart des scénarios impliquant des entités de la couche de service, nous pouvons nous débrouiller avec les appels de service par défaut, avec l'aide des chercheurs. Pour les scénarios simples impliquant plusieurs entités, nous allons utiliser l'API de requête dynamique. pour l'API Criteria utilisée dans Hibernate.Il peut être utilisé pour les cas où nous devons générer une requête dynamique, qui n'est pas très complexe par nature, en utilisant plusieurs constructions de l'API. Pour commencer, certaines des constructions les plus couramment utilisées sont: `DynamicQueryFactoryUtil` -Utilisé pour la construction de la requête

`RestrictionsFactoryUtil` Utilisé pour fournir des restrictions sur les champs pour la comparaison avec une certaine valeur pour limiter les résultats correspondant à une certaine valeur ou dans une plage, etc.

`ProjectionFactoryUtil` -Utilisé pour fournir des projections pour obtenir des champs qui feront partie du résultat de la recherche, au lieu de fournir l'entité entière, fournira uniquement certains champs ou appliquer la fonction d'agrégation (telle que min.max, avg) sur le même.

`PropertyFactoryUtil` -Utilisé pour la comparaison de certaines propriétés de la classe d'entité pour effectuer la comparaison avec d'autres champs d'une requête

L'implémentation de ces classes est présente dans le package dao.orm.jpa avec toutes les

méthodes disponibles

Lire Utilisation d'une requête SQL dynamique et personnalisée dans Liferay en ligne: https://riptutorial.com/fr/liferay/topic/10863/utilisation-d-une-requete-sql-dynamique-et-personnalisee-dans-liferay

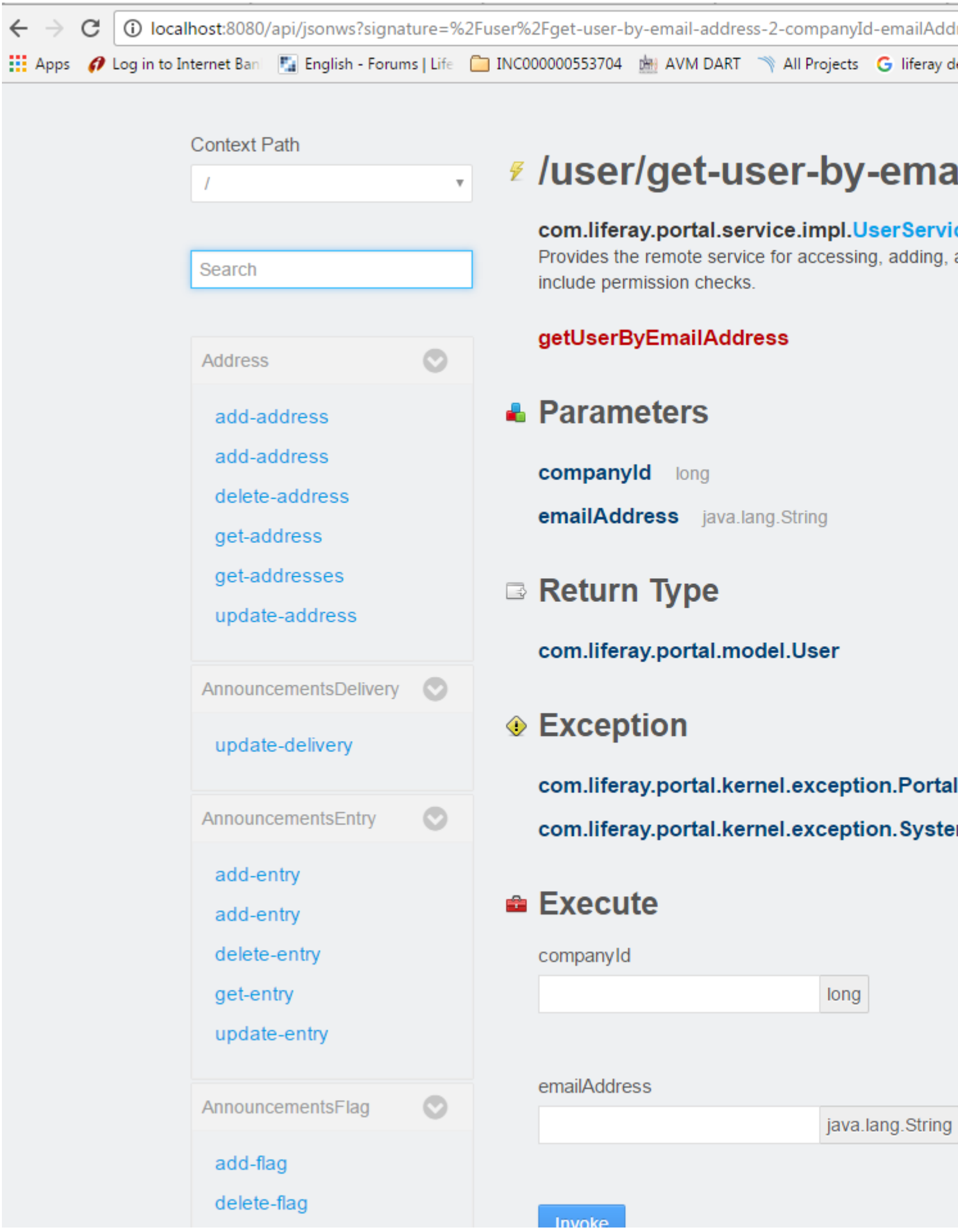
Chapitre 10: Utiliser le service Web Restful dans Liferay

Examples

Consommez le service Jifer Liferay pour les requêtes GET

Liferay expose de nombreux services par défaut et personnalisés disponibles sur d'autres systèmes via JSON. Pour explorer des services sur une instance liferay particulière, utilisez une URL donnée - Une instance locale dans ce cas:

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



Sélectionnez le service requis, utilisez le service avec la syntaxe et les paramètres indiqués:

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

Utilisez `companyId and emailAddress` pour récupérer l'utilisateur avec les types de données attendus, ainsi que les éventuelles exceptions à gérer par le consommateur.

L'exemple suivant consomme ce service à partir d'un portlet. La méthode de classe d'utilitaire donnée appelle le service Web en transmettant les arguments nécessaires:

```
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
fetech 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 autheticate 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");

}

}
```

Lire Utiliser le service Web Restful dans Liferay en ligne:
https://riptutorial.com/fr/liferay/topic/7821/utiliser-le-service-web-restful-dans-liferay

Crédits

S. No	Chapitres	Contributeurs
1	Commencer avec liferay	brandizzi, Community, Pier Paolo Ramon, Pierpaolo Cira, rp.
2	Communication inter portlet	a_horse_with_no_name, Shivam Aggarwal
3	Configuration de SSL	El0din, rp., Tofik Sahraoui
4	Configurer le gestionnaire de tags Google (GTM) dans liferay	Shivam Aggarwal
5	Créer un planificateur Quartz en mode de vie	Shivam Aggarwal
6	Crochets à Liferay	Chris Maggiulli, El0din, KLajdPaja, Pier Paolo Ramon, rp.
7	Déboguer le serveur Layeray distant via Eclipse	4444, Shivam Aggarwal
8	Déploiement d'un plugin	mico, Pier Paolo Ramon, rp.
9	Utilisation d'une requête SQL dynamique et personnalisée dans Liferay	Shivam Aggarwal
10	Utiliser le service Web Restful dans Liferay	4444, Shivam Aggarwal