

Sedona Liferay avec JRebel

Date	04/04/2015
Version	1.0

Référence du document

Auteur	Ling LIN
Version	V1_0
Référence Document	
Nombre de pages	

Liste de diffusion

Nom	Fonctions/Service
Richard Sinelle	Architecte Sedona

Sommaire

1	Introduction	5
1.1	Pré-requis.....	5
1.2	Prélude.....	5
2	Installation	6
2.1	Liferay Developer Studio.....	6
2.1.1	Installation (http://zeroturnaround.com/software/jrebel/quickstart/eclipse/)	6
2.1.2	Configuration du plugin JRebel	8
2.2	Intellij IDEA	10
2.2.1	Installation (http://zeroturnaround.com/software/jrebel/quickstart/intellij/)	10
2.2.2	Configuration du plugin JRebel	10
3	Génération du fichier rebel.xml	12
3.1	Liferay Developer Studio.....	12
3.2	Intellij IDEA	12
3.3	Maven.....	13
3.3.1	Génération	13
3.3.2	Configuration avancée.....	14
4	Plugins Liferay	16
4.1	Portlet.....	16
4.1.1	Préambule.....	16
4.1.2	Classes Java	17
4.1.3	Fichiers d'internationalisation	19
4.1.4	Fichiers Liferay.....	20
4.1.5	Service Builder.....	21
4.1.6	Conclusion.....	21
4.2	Hook.....	22
4.2.1	Préambule.....	22
4.2.2	Ajout, Modification de JSPs, classes Java	22
4.2.3	Conclusion.....	28
4.3	Thème et Layout template	29
4.3.1	Préambule.....	29
4.3.2	Configuration	29
4.3.3	Modification des fichiers	29
4.3.4	Conclusion.....	32
5	Liferay Core.....	33
5.1	Préambule	33
5.2	Import des sources dans Eclipse	34
5.3	Configuration du projet.....	35

5.4	Erreur	37
5.5	Classes Java.....	37
5.6	Fichiers JSPs	37

1 Introduction

Ce document présente l'utilisation de l'outil de développement JRebel avec Liferay.

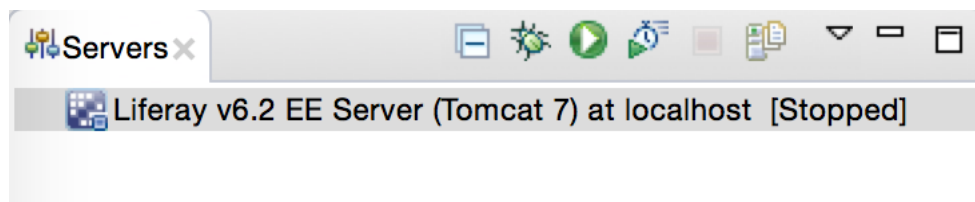
1.1 Pré-requis

Les différents outils utilisés ici sont :

- Liferay Developer Studio 2.2.0 (LDS)
- IntelliJ IDEA 14.1
- JRebel 6.1.1
- Bundle Liferay 6.2 EE SP10 + sources
- Maven 3.2.1
- Ant 1.9.4

On suppose que vous avez déjà installé un environnement de développement Liferay sous l'un des IDE.

PS : On n'ajoutera aucun projet au serveur Liferay.

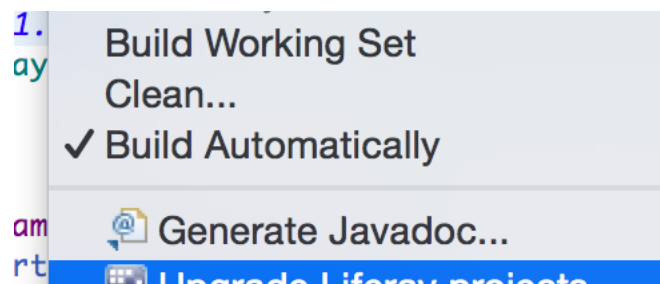


1.2 Prélude

Générons un projet via l'archetype de Sedona qui nous servira durant tout le document.

```
mvn archetype:generate -DarchetypeGroupId=com.sedona.liferay  
-DarchetypeArtifactId=liferay-project-archetype -DgroupId=com.sedona.jrebel  
-DartifactId=jrebel -DinteractiveMode=false -Dversion=6.2-ee-sp10
```

Eclipse ou LDS doit être configuré avec le « build automatically » coché :



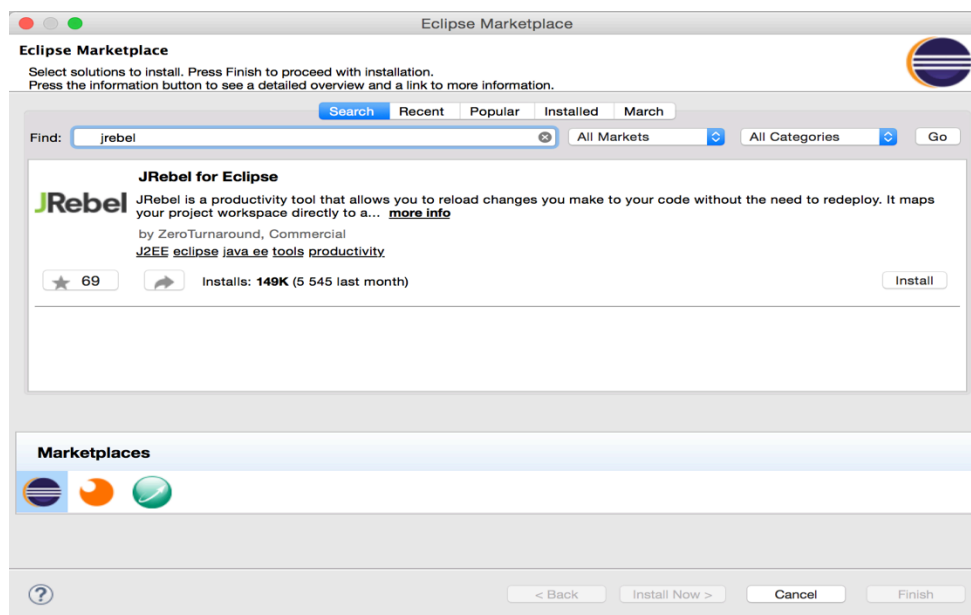
2 Installation

2.1 Liferay Developer Studio

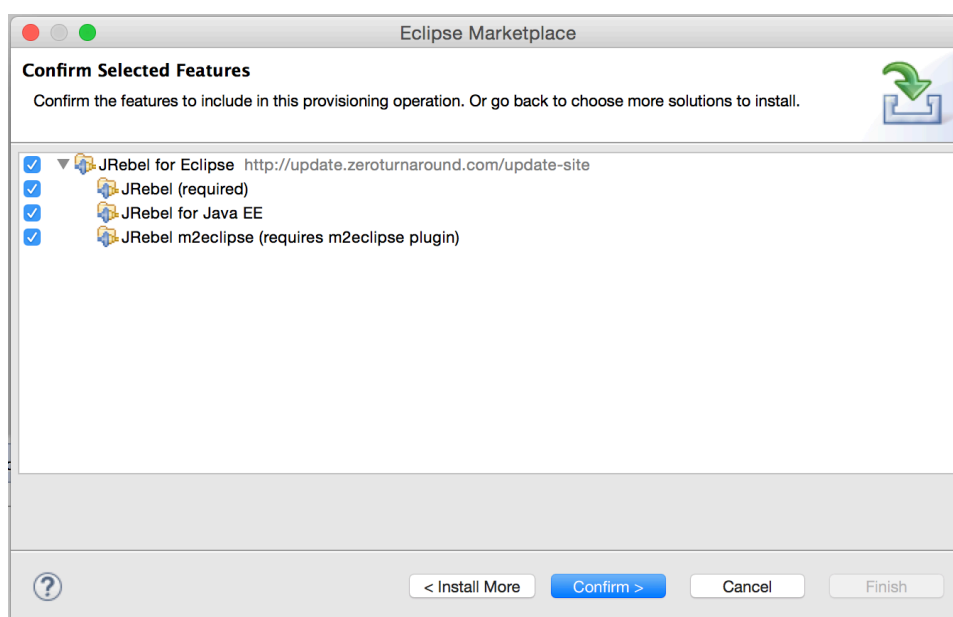
2.1.1 Installation

(<http://zeroturnaround.com/software/jrebel/quickstart/eclipse/>)

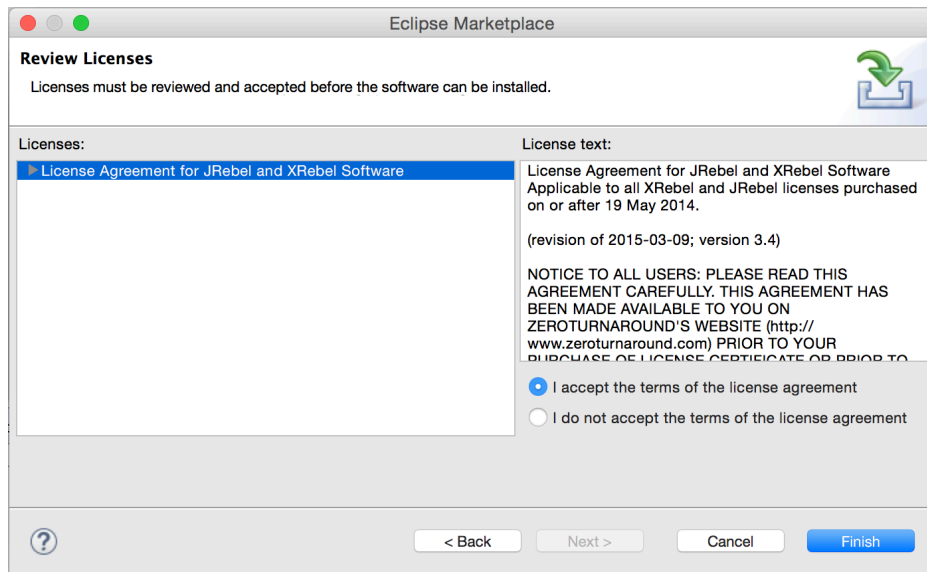
L'installation de JRebel est assez simple. Il suffit d'aller sur le marketplace d'Eclipse (Help -> Eclipse Marketplace) et de faire une recherche sur JRebel :



Cliquer sur « Install » pour arriver à l'écran suivant :

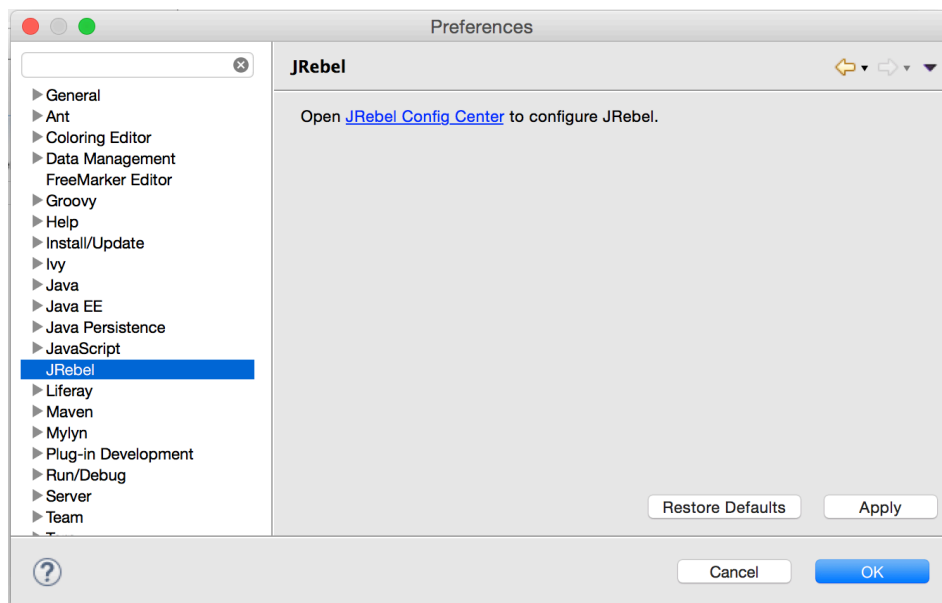


Laisser tout cocher et confirmer en cliquant sur le bouton « Confirm ».



Accepter la licence et cliquer sur « Finish » pour procéder au téléchargement et à l'installation de JRebel dans l'IDE. Après l'installation, redémarrer l'IDE pour prendre en compte l'installation de JRebel.

Une fois l'IDE redémarré, allez dans les préférences de l'IDE et sélectionner dans le menu à gauche JRebel :



Cliquer sur le lien « JRebel Config Center » afin d'ouvrir la fenêtre de configuration de JRebel.

Une fois la fenêtre de configuration ouverte, cliquer sur le lien « Activate/Update Licence » afin d'insérer votre licence.

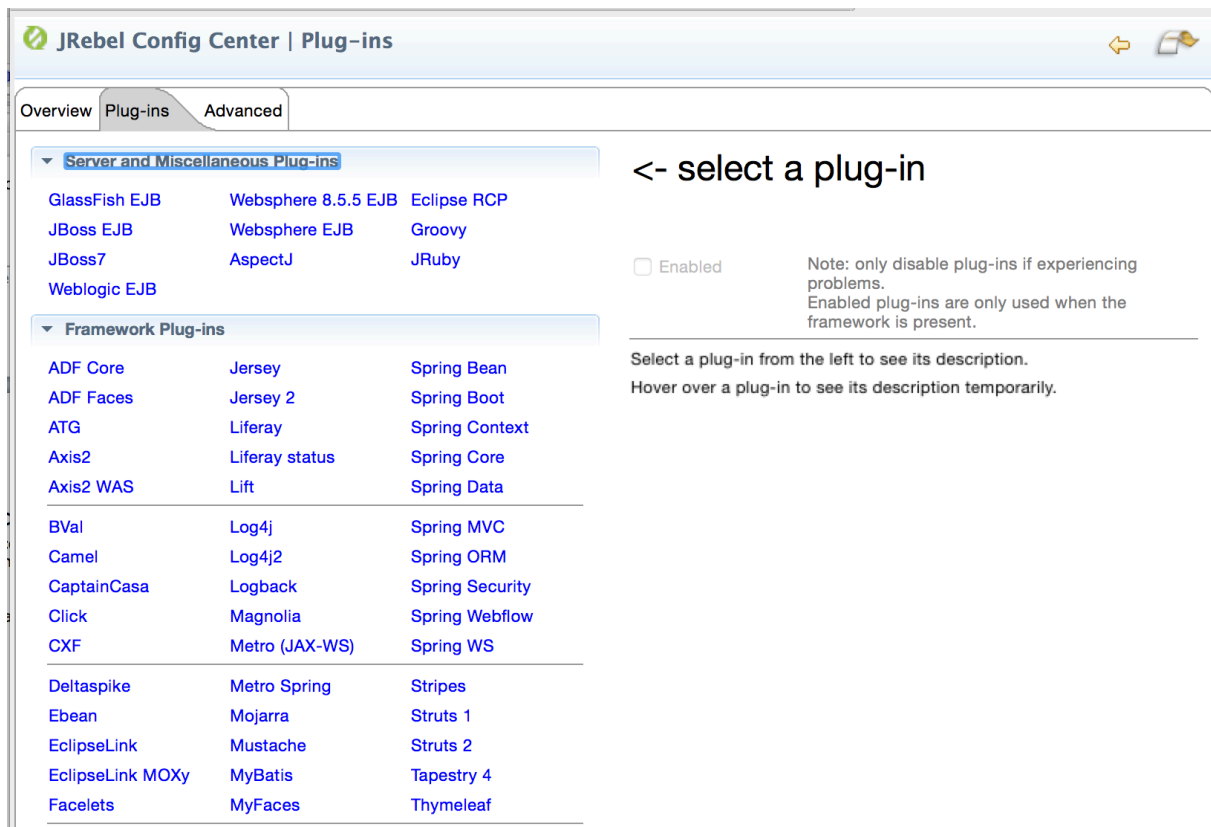
La licence insérée, JRebel va vérifier cette licence auprès de leurs serveurs et va activer votre plugin.

2.1.2 Configuration du plugin JRebel

Pour mettre en œuvre le déploiement direct (live deployment), JRebel installe un agent (javaagent) qui surveillent les classes Java et les ressources du workspace et propage les modifications à l'application en cours de modification.

Avant de lancer le serveur Liferay, il est nécessaire de désactiver certains plugins JRebel. Cette étape n'est pas obligatoire mais cela permet de ne pas occuper de la mémoire inutilement.

Pour cela, il faut ouvrir la fenêtre de configuration de JRebel et aller dans l'onglet « Plug-ins ».



JRebel Config Center | Plug-ins

Overview **Plug-ins** Advanced

Server and Miscellaneous Plug-ins

- GlassFish EJB
- JBoss EJB
- JBoss7
- Weblogic EJB
- Websphere 8.5.5 EJB
- Websphere EJB
- AspectJ
- Eclipse RCP
- Groovy
- JRuby

Framework Plug-ins

ADF Core	Jersey	Spring Bean
ADF Faces	Jersey 2	Spring Boot
ATG	Liferay	Spring Context
Axis2	Liferay status	Spring Core
Axis2 WAS	Lift	Spring Data
BVal	Log4j	Spring MVC
Camel	Log4j2	Spring ORM
CaptainCasa	Logback	Spring Security
Click	Magnolia	Spring Webflow
CXF	Metro (JAX-WS)	Spring WS
Deltaspike	Metro Spring	Stripes
Ebean	Mojarra	Struts 1
EclipseLink	Mustache	Struts 2
EclipseLink MOXy	MyBatis	Tapestry 4
Facelets	MyFaces	Thymeleaf

<- select a plug-in

Enabled

Note: only disable plug-ins if experiencing problems. Enabled plug-ins are only used when the framework is present.

Select a plug-in from the left to see its description.
Hover over a plug-in to see its description temporarily.

Les plug-ins à activer obligatoirement pour Liferay sont :

- Liferay
- Liferay status
- La suite Spring
- Axis2
- Hibernate
- Hibernate Validator
- Jackson 1.x et Jackson 2.x
- Log4j et Log4j2
- Logback
- Struts 1
- Tiles 1
- Velocity

Dans le dernier onglet « Advanced » de la fenêtre de configuration JRebel, il est possible de :

- Modifier les logs de JRebel
- Sélectionner un agent JRebel (Legacy agent ou JRebel 6 agent). La différence peut se trouver sur le site de JRebel (<http://zeroturnaround.com/software/jrebel/features/comparison-matrix/>). En résumé, le nouvel agent « JRebel 6 agent » permet de prendre en compte les nouvelles classes. Les nouvelles classes Java sont initialisées sans à avoir à redémarrer le serveur. Cependant, ce nouvel agent requiert Java 6 tandis que le Legacy requiert Java 4.
- Etc.

Il est possible d'utiliser l'agent « JRebel 6 agent » seulement si Liferay core n'est pas configuré avec JRebel.

Lorsque Liferay core est configuré avec le nouvel agent, la page d'accueil ne s'affiche pas et tourne en rond sans qu'il y ait une erreur. Cependant, avec les plugins Liferay, cela ne pose pas de problème.

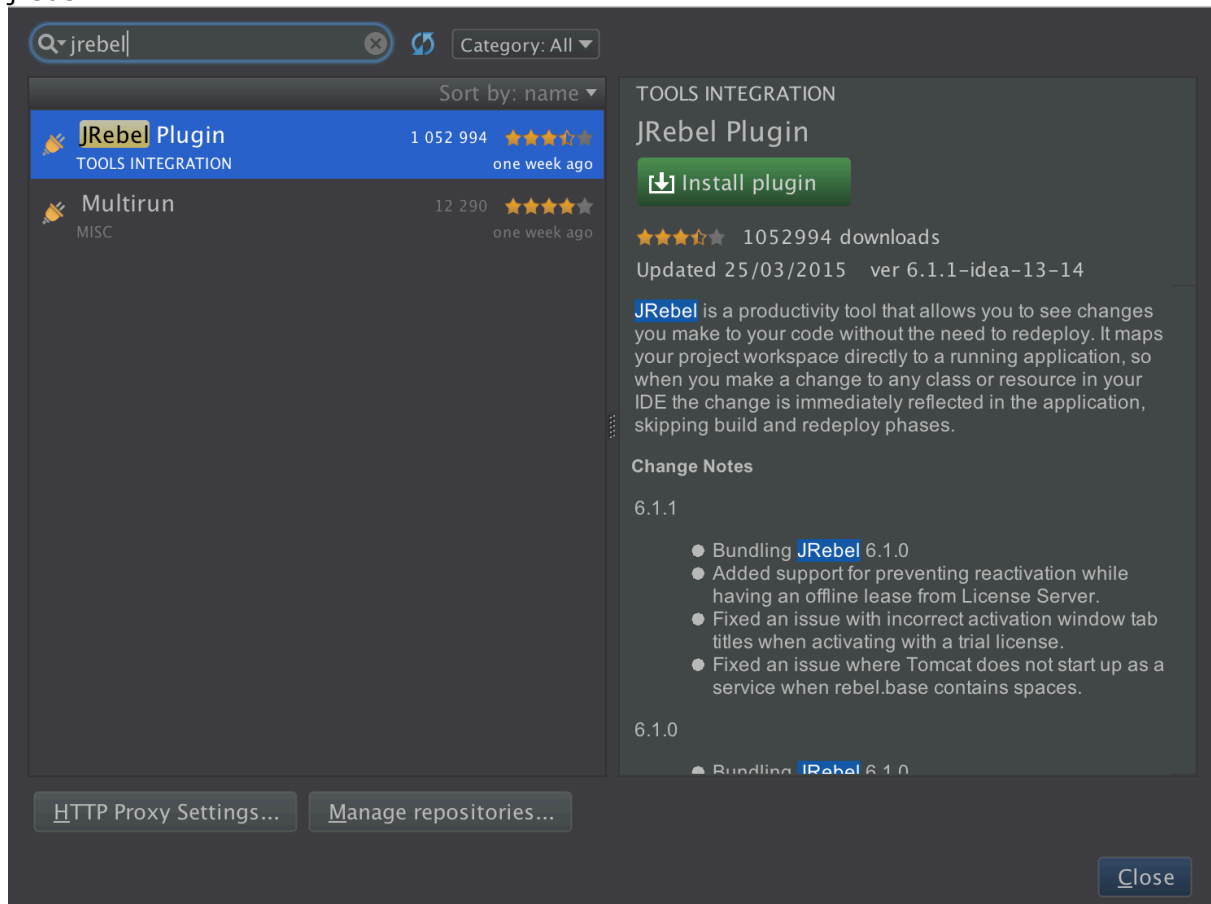
Liferay démarre plus rapidement avec l'agent « Legacy » qu'avec le nouvel agent.

2.2 IntelliJ IDEA

2.2.1 Installation

(<http://zeroturnaround.com/software/jrebel/quickstart/intellij/>)

JRebel s'installe facilement sous IntelliJ IDEA. Aller dans les préférences puis cliquer dans le menu à gauche « Plugins » puis sur « Browse repositories... » et enfin rechercher jrebel :



Cliquer sur « Install plugin » et une fois installé, redémarrer l'IDE.

Après le redémarrage de l'IDE, aller dans les préférences puis dans « Other settings » et enfin JRebel. Cliquer sur « Activate » et entrer votre licence pour activer le plugin sous IntelliJ IDEA.

2.2.2 Configuration du plugin JRebel

Avant de lancer le serveur Liferay, il est nécessaire de désactiver certains plugins JRebel. Cette étape n'est pas obligatoire mais cela permet de ne pas occuper de la mémoire inutilement.

Pour cela, il faut aller dans les préférences de l'IDE puis « Others settings » puis « JRebel » et enfin « Plugins ». Décocher les plugins JRebel qui ne sont pas utiles à Liferay.

Les plugins JRebel nécessaires à Liferay sont :

- Liferay
- Liferay status
- La suite Spring

- Axis2
- Hibernate
- Hibernate Validator
- Jackson 1.x et Jackson 2.x
- Log4j et Log4j2
- Logback
- Struts 1
- Tiles 1
- Velocity

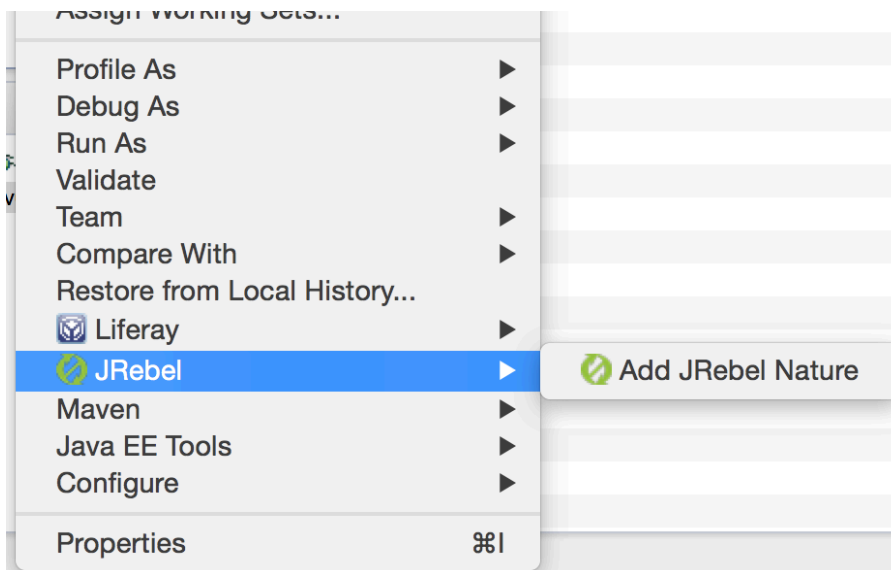
3 Génération du fichier rebel.xml

Comme vu précédemment, pour que JRebel puisse fonctionner, JRebel a besoin du fichier « rebel.xml » pour chaque module à déployer et à recharger.

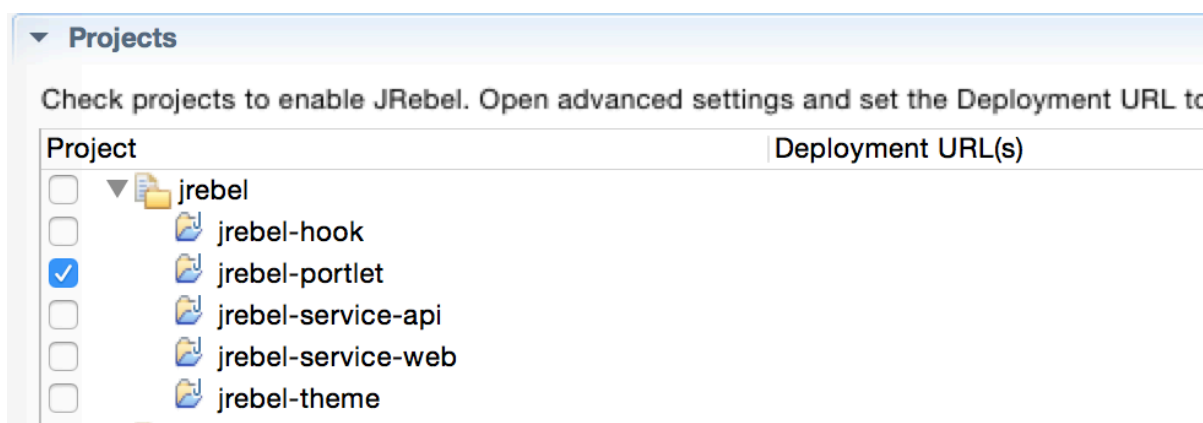
3.1 Liferay Developer Studio

Le fichier « rebel.xml » se génère facilement et de 2 manières :

- En faisant un clic droit sur le projet dont JRebel doit suivre et cliquer sur « JRebel -> Add JRebel Nature »



- En allant dans la fenêtre de configuration de JRebel et cocher la case à coté du projet dont JRebel doit suivre.

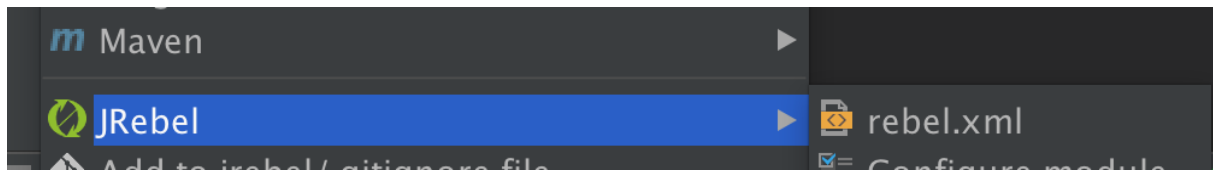


Après la génération du fichier, il est possible de vérifier l'existence du fichier dans « src/main/resources ». Redéployer ensuite l'application dans Liferay.

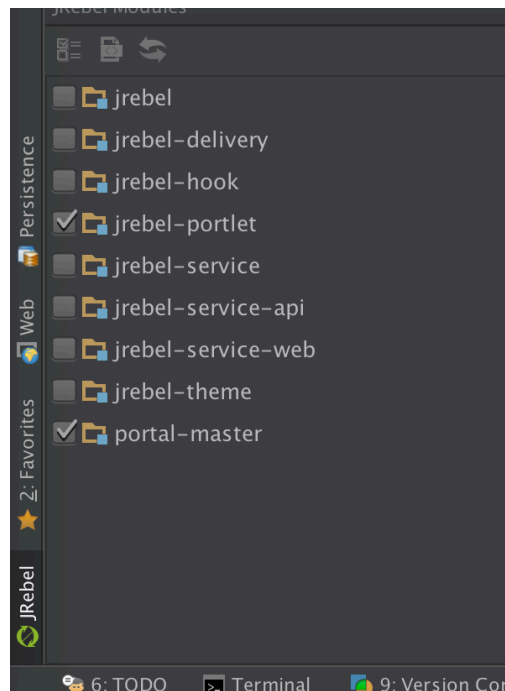
3.2 IntelliJ IDEA

Le fichier « rebel.xml » se génère facilement et de 2 manières :

- En faisant un clic droit sur le projet dont JRebel doit suivre et cliquer sur « JRebel -> rebel.xml »



- En ouvrant la fenêtre JRebel et en cochant sur le projet dont JRebel doit suivre



Une fois le fichier généré, il est possible de vérifier l'existence du fichier dans « src/main/resources ». Il faut ensuite redéployer l'application.

3.3 Maven

3.3.1 Génération

Il est possible de générer le fichier « rebel.xml » via le plugin Maven. Pour cela, ajouter dans le fichier pom.xml la configuration suivante :

```
<plugin>
  <groupId>org.zeroturnaround</groupId>
  <artifactId>jrebel-maven-plugin</artifactId>
  <version>1.1.5</version>
  <executions>
    <execution>
      <id>generate-rebel-xml</id>
      <phase>process-resources</phase>
      <goals>
        <goal>generate</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

```
</executions>
</plugin>
```

Puis :

- Lancer la commande pour générer le fichier « rebel.xml »

```
mvn jrebel:generate
```

- Lancer la commande pour générer le fichier et le war à déployer

```
mvn package
```

- Lancer la commande pour générer le fichier, le war et le déployer dans Liferay

```
mvn package liferay:deploy
```

3.3.2 Configuration avancée

```
<plugin>
  <groupId>org.zeroturnaround</groupId>
  <artifactId>jrebel-maven-plugin</artifactId>
  <version>1.1.5</version>
  <configuration>
    <!-- If your project uses custom packaging that is not recognized set this to jar
or war. -->
    <packaging>war</packaging>
    <classpath>
      <fallback>default</fallback>
    <resources>
      <resource>
        <!-- A relative path. -->
        <directory>target/special-classes </directory>
        <!-- You may use includes and excludes as with any other resource. -->
        <includes>
          <include>com/yourapp/include/package1/**</include>
          <include>com/yourapp/include/package2/**</include>
        </includes>
        <excludes>
          <exclude>com/yourapp/exclude/package1/**</exclude>
          <exclude>com/yourapp/exclude/package2/**</exclude>
        </excludes>
      </resource>
      <resource>
        <!-- Empty resource element marks default configuration. By default it is
placed first in generated configuration. -->
      </resource>
      <resource>
        <!-- An absolute path is used here. -->
        <jar>c:\projects\myProject\3rdpartyLibs\myLibrary.jar</jar>
      </resource>
      <resource>
        <jarset>app/3rd-party-lib</jarset>
        <excludes>
          <exclude>apache*.jar</exclude>
```

```
</excludes>
</resource>
<resource>
  <dirset>c:\projects\project1Root</dirset>
  <excludes>
    <exclude>**\build\classes</exclude>
  </excludes>
</resource>
</resources>
</classpath>
<war>
  <path>c:\projects\myProject\dist\myProject.war</path>
</war>
<web>
  <resources>
    <resource>
      <target>gfx</target>
      <directory>c:\projects\myProject\static\gfx
      </directory>
    </resource>
    <resource>
      <!-- Empty resource element marks default configuration. By default it is
placed first in generated configuration. -->
      </resource>
    </resources>
  </web>

  <!--
addResourcesDirToRebelXml - default is false
Required if the resource directories are to be added to rebel.xml
-->
<addResourcesDirToRebelXml>true</addResourcesDirToRebelXml>

  <!--
alwaysGenerate - default is false
If 'false' - rebel.xml is generated if timestamps of pom.xml and the current
rebel.xml file are not equal.
If 'true' - rebel.xml will always be generated
-->
<alwaysGenerate>true</alwaysGenerate>

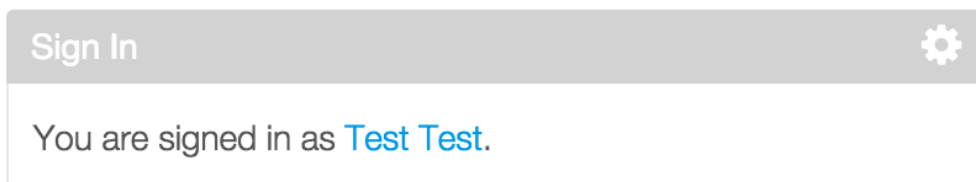
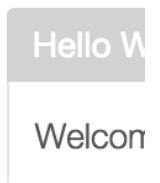
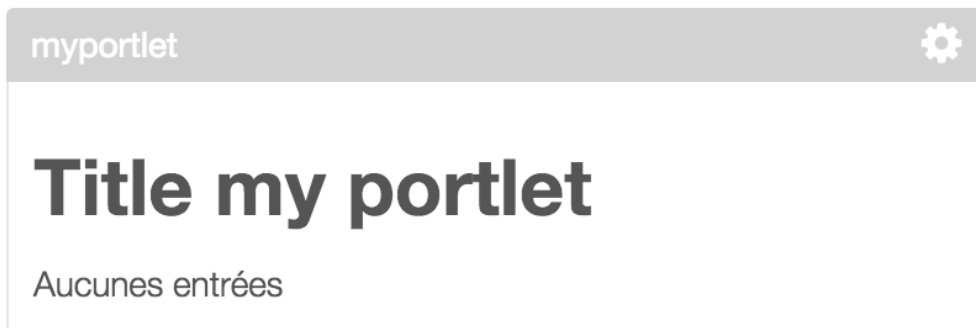
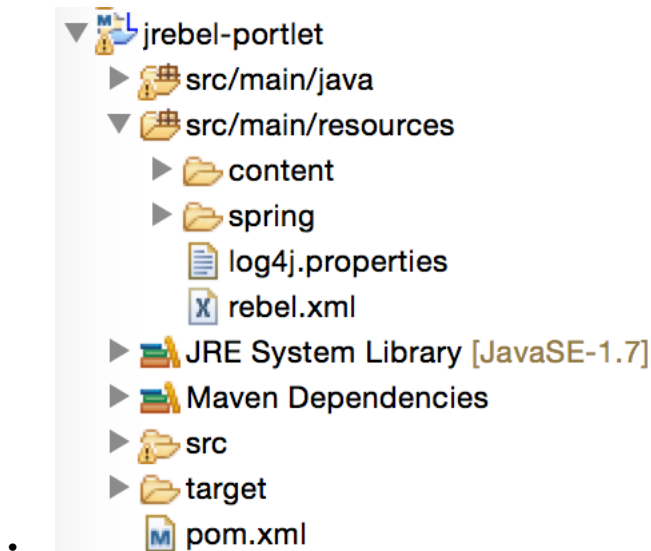
</configuration>
</plugin>
```

4 Plugins Liferay

4.1 Portlet

4.1.1 Préambule

La portlet doit être configurée avec le fichier « jrebel.xml » et déployé dans Liferay :



4.1.2 Classes Java

4.1.2.1 Modification d'une classe

Modifier le code la classe « MyPortletController » en ajoutant le bout de code à l'entrée de la méthode « show » :

```
System.out.println("Hello JRebel");
```

On a donc :

```
public String show(Model model, PortletPreferences portletPreferences, RenderRequest request) {
    System.out.println("Hello JRebel");
    String url = PrefsParamUtil.getString(portletPreferences, request, MyPortletConstant.Preferences.URL, StringPool.BLANK);
    int num = PrefsParamUtil.getInteger(portletPreferences, request, MyPortletConstant.Preferences.NUM_OF_ENTRIES, MyPortletConstant.DEFAULT_NUM_OF_ENTRIES);

    List<MyPortletBean> myPortletBeans = null;
    if (!Strings.isNullOrEmpty(url)) {
        myPortletBeans = myPortletService.getEntries(url, num);
    }
    model.addAttribute("myPortletBeans", myPortletBeans);
    return "view";
}
```

Sauvegarder et en regardant la console, on obtient :

```
JRebel: Reloading class
'com.sedona.jrebel.portlet.myportlet.controller.MyPortletController'.
```

Rafraichissons la page et dans la console on peut voir apparaître le bout de code inséré :

```
Hello JRebel
```

4.1.2.2 Ajout d'une nouvelle classe

Créer une classe Java dans le package « com.sedona.jrebel.portlet.myportlet.service » :

```
package com.sedona.jrebel.portlet.myportlet.service;

import org.springframework.stereotype.Service;

@Service
public class JrebelService {

    public String getString() {
        return this.getClass().getName();
    }
}
```

Puis modifier la classe « MyPortletController » afin d'injecter ce nouveau service :

```

package com.sedona.jrebel.portlet.myportlet.controller;

import com.google.common.base.Strings;

import java.util.List;

import javax.inject.Inject;
import javax.portlet.PortletPreferences;
import javax.portlet.RenderRequest;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.portlet.bind.annotation.RenderMapping;

import com.liferay.portal.kernel.util.PrefsParamUtil;
import com.liferay.portal.kernel.util.StringPool;
import com.sedona.jrebel.portlet.myportlet.model.MyPortletBean;
import com.sedona.jrebel.portlet.myportlet.service.JrebelService;
import com.sedona.jrebel.portlet.myportlet.service.MyPortletService;
import com.sedona.jrebel.portlet.myportlet.utils.MyPortletConstant;

/**
 * MyPortletController
 */
@Controller
@RequestMapping("VIEW")
public class MyPortletController {

    @Inject
    private MyPortletService myPortletService;

    @Inject
    private JrebelService jrebelService;

    /**
     * Show string.
     *
     * @param model the model
     * @param portletPreferences the portlet preferences
     * @param request the request
     * @return the string
     */
    @RenderMapping
    public String show(Model model, PortletPreferences portletPreferences, RenderRequest
request) {
        System.out.println("Hello JRebel");
        System.out.println(jrebelService.getString());
        String url = PrefsParamUtil.getString(portletPreferences, request,
MyPortletConstant.Preferences.URL,
StringPool.BLANK);
        int num = PrefsParamUtil.getInteger(portletPreferences, request,
MyPortletConstant.Preferences.NUM_OF_ENTRIES,
MyPortletConstant.DEFAULT_NUM_OF_ENTRIES);
    }
}

```

```

List<MyPortletBean> myPortletBeans = null;
if (!Strings.isNullOrEmpty(url)) {
    myPortletBeans = myPortletService.getEntries(url, num);
}
model.addAttribute("myPortletBeans", myPortletBeans);
return "view";
}
}

```

Après sauvegarde du controller et rafraichissement de la page, on obtient dans la console :

```

Hello JRebel
com.sedona.jrebel.portlet.myportlet.service.JrebelService

```

4.1.3 Fichiers d'internationalisation

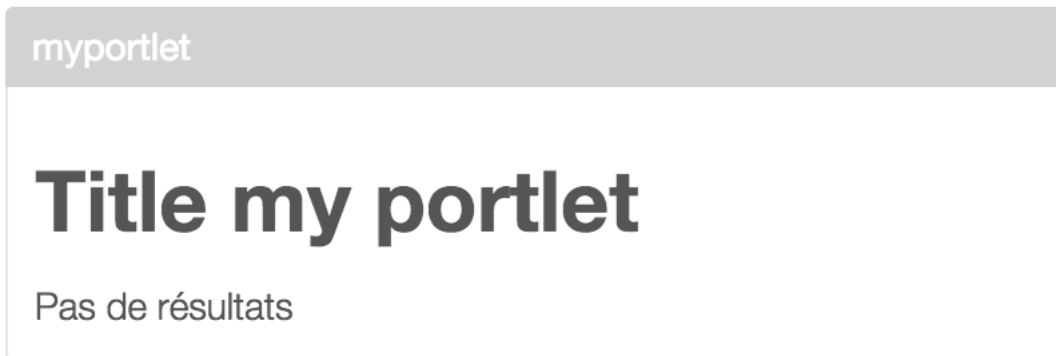
Ouvrir le fichier « myportlet.properties » et modifier la valeur de la propriété « portlet.myportlet.no.entries » par :

```

Pas de r\u00e9sultats

```

Après sauvegarde du fichier et rafraichissement de la page :



Ajouter à la fin du fichier « view.jsp » (qui se trouve dans « src/main/webapp/html/portlet/myportlet ») :

```

<spring:message code="portlet.myportlet.footer"/>

```

et ajouter également à la fin du fichier myportlet.properties :

```

portlet.myportlet.footer=Footer

```

Rafraichissez la page pour obtenir :

myportlet

Title my portlet

Pas de résultats

Footer

4.1.4 Fichiers Liferay

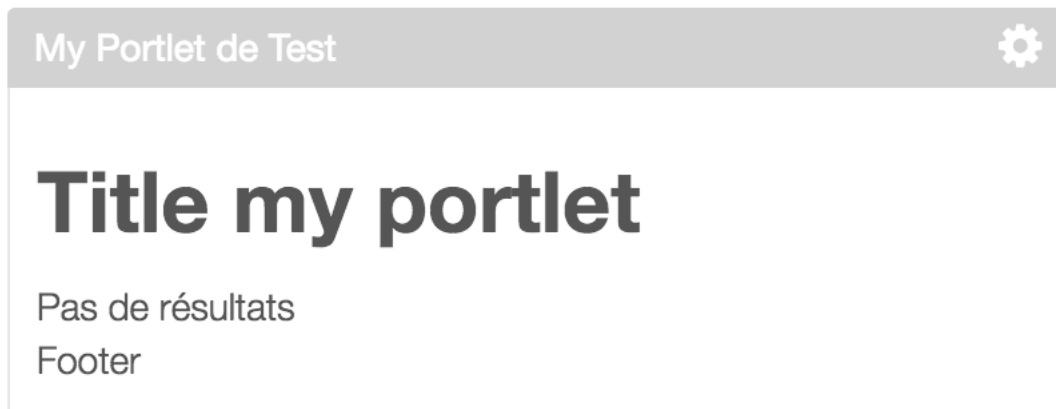
Il est possible de modifier à chaud les fichiers de « Liferay » tels que :

- liferay-display.xml
- liferay-portlet.xml
- portlet.xml

Par exemple, modifier le fichier « portlet.xml » en changeant le champ « title » par exemple :

```
<portlet>
  <portlet-name>myportlet</portlet-name>
  <display-name>myportlet</display-name>
  <portlet-class>org.springframework.web.portlet.DispatcherPortlet</portlet-class>
  <init-param>
    <name>contextConfigLocation</name>
    <value>/WEB-INF/classes/spring/portlet-myportlet.xml</value>
  </init-param>
  <supports>
    <mime-type>text/html</mime-type>
    <portlet-mode>view</portlet-mode>
  </supports>
  <resource-bundle>content/myportlet</resource-bundle>
  <portlet-info>
    <title>My Portlet de Test</title>
    <short-title>myportlet</short-title>
    <keywords>myportlet</keywords>
  </portlet-info>
</portlet>
```

On obtient :



4.1.5 Service Builder

JRebel ne permet pas le rechargement à chaud des nouvelles classes créées par le Service Builder de Liferay. De même les scripts SQL ne sont pas exécutés. Il faut obligatoirement déployer la portlet.

Cependant, toute modification faite par la suite sera prise en compte via JRebel.

4.1.6 Conclusion

Bien que LDS permette de déployer les portlets automatiquement, Liferay les redéploye à chaque fois. Alors si on a un projet avec une dizaine de portlet par exemple, Liferay les redéploye tous même si on a fait une modification mineure sur un controller par exemple.

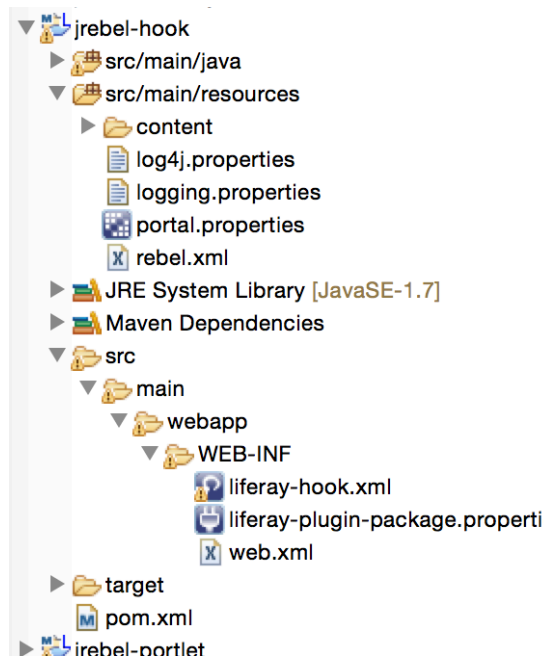
De plus, après un certain nombre de déploiement, Liferay passe en Permgem. Il faut alors redémarrer le serveur, ce qui peut être une perte de temps.

Avec JRebel, aucun redéploiement n'est effectué. De plus, à chaque sauvegarde d'une modification d'une classe ou fichier, celui-ci est automatiquement et rapidement pris en compte par Liferay.

4.2 Hook

4.2.1 Préambule

Le hook doit être configuré avec le fichier « rebel.xml » et déployé dans Liferay :



Il est possible que JRebel n'ait pas généré le fichier « rebel.xml » correctement et ait ajouté une ligne supplémentaire :

```
<link target="/">
  <dir name="/.../jrebel/jrebel-hook/target/m2e-wtp/web-resources">
    <exclude name="/">
  </dir>
</link>
```

Supprimer le bout de code et déployer le hook.

4.2.2 Ajout, Modification de JSPs, classes Java

4.2.2.1 JSP

Créer les dossiers « html/portlet/login » dans le dossier « WEB-INF » du projet Hook.

Copier la JSP « login.jsp » des sources de Liferay dans le dossier « login » puis modifier le fichier en ajoutant en haut de la JSP :

Modification du Hook via JRebel

Sauvegarder puis rafraichir la page pour obtenir le résultat suivant :

Sign In

Modification du Hook via JRebel

Screen Name

Password

Remember Me

[Sign In](#)

PS : Vérifier que la propriété « custom-jsp-dir » a été définie dans le fichier « liferay-hook.xml ». Celle-ci doit pointer vers le dossier « WEB-INF ».

4.2.2.2 Indexer Post Processor

Créer une classe « CustomUserIndexer » dans le package « com.sedona.jrebel.indexer » :

```

package com.sedona.jrebel.indexer;

import com.liferay.portal.kernel.search.BaseIndexerPostProcessor;
import com.liferay.portal.kernel.search.BooleanQuery;
import com.liferay.portal.kernel.search.Document;
import com.liferay.portal.kernel.search.Field;
import com.liferay.portal.kernel.search.SearchContext;
import com.liferay.portal.kernel.search.Summary;
import com.liferay.portal.model.User;

import java.util.Locale;

import javax.portlet.PortletURL;

public class CustomUserIndexer extends BaseIndexerPostProcessor {

    public void postProcessContextQuery(BooleanQuery booleanQuery, SearchContext searchcontext) throws Exception {
        System.out.println(" postProcessContextQuery()");
    }

    public void postProcessDocument(Document document, Object object) throws Exception {
        System.out.println("postProcessDocument()");
        User userEntity = (User) object;
    }
}

```

```

String indexerUserTitle = "";
try {
    indexerUserTitle = userEntity.getJobTitle();
} catch (Exception e) {
}
if (indexerUserTitle.length() > 0)
    document.addText(Field.TITLE, indexerUserTitle);
}

public void postProcessFullQuery(BooleanQuery fullQuery, SearchContext
searchcontext) throws Exception {
    System.out.println(" postProcessFullQuery()");
}

public void postProcessSearchQuery(BooleanQuery searchquery, SearchContext
searchcontext) throws Exception {
    System.out.println(" postProcessSearchQuery()");
}

public void postProcessSummary(Summary summary, Document document, Locale
locale, String snippet,
    PortletURL portletURL) {
    System.out.println("postProcessSummary()");
}
}

```

Ajouter le code suivant dans le fichier « liferay-hook.xml » après la balise « custom-jsp-dir » :

```

<indexer-post-processor>
  <indexer-class-name>com.liferay.portal.model.User</indexer-class-name>
  <indexer-post-processor-
impl>com.sedona.jrebel.indexer.CustomUserIndexer</indexer-post-processor-impl>
</indexer-post-processor>

```

Une fois le fichier sauvegardé, faites une ré-indexation pour voir dans la console :

```
postProcessDocument()
```

Chaque modification faite (après sauvegarde fichier) sera directement prise en compte dans Liferay.

4.2.2.3 Services

Créer une classe « CustomUserLocalServiceImpl » dans le package « com.sedona.jrebel.service » :

```

package com.sedona.jrebel.service;

import com.liferay.portal.kernel.exception.PortalException;
import com.liferay.portal.kernel.exception.SystemException;
import com.liferay.portal.model.User;
import com.liferay.portal.service.UserLocalService;

```



```
import com.liferay.portal.service.UserLocalServiceWrapper;

public class CustomUserLocalServiceImpl extends UserLocalServiceWrapper {

    public CustomUserLocalServiceImpl(UserLocalService userLocalService) {
        super(userLocalService);
        // TODO Auto-generated constructor stub
    }

    public User getUserById(long userId) throws PortalException, SystemException {
        System.out.println("## MyUserLocalServiceImpl.getUserById(" + userId + ")");
        return super.getUserById(userId);
    }
}
```

Ajouter le code suivant dans le fichier « liferay-hook.xml » après la balise « indexer-post-processor » :

```
<service>
  <service-type>com.liferay.portal.service.UserLocalService</service-type>
  <service-impl>com.sedona.jrebel.service.CustomUserLocalServiceImpl</service-impl>
</service>
```

Une fois le fichier sauvegardé, aller dans la page des utilisateurs pour que la console affiche :

```
## MyUserLocalServiceImpl.getUserById(...)
```

Toute modification du fichier « CustomUserLocalServiceImpl » sera directement prise en compte dans Liferay.

4.2.2.4 Servlet-filter

Créer une classe « CustomFilter » dans le package « com.sedona.jrebel.filter » :

```
package com.sedona.jrebel.filter;

import java.io.IOException;

import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;

public class CustomFilter implements Filter {

    @Override
    public void destroy() {
        System.out.println("Custom filter destroy");
    }
}
```

```

@Override
public void doFilter(ServletRequest arg0, ServletResponse arg1,
    FilterChain arg2) throws IOException, ServletException {
    System.out.println("Custom filter doFilter");
    arg2.doFilter(arg0, arg1);
}

@Override
public void init(FilterConfig arg0) throws ServletException {
    System.out.println("Custom filter init");
}
}

```

Ajouter le code suivant dans le fichier « liferay-hook.xml » après la balise « service » :

```

<servlet-filter>
  <servlet-filter-name>Custom Filter</servlet-filter-name>
  <servlet-filter-impl>com.sedona.jrebel.filter.CustomFilter</servlet-filter-impl>
</servlet-filter>
<servlet-filter-mapping>
  <servlet-filter-name>Custom Filter</servlet-filter-name>
  <url-pattern>/*</url-pattern>
</servlet-filter-mapping>

```

Une fois le fichier sauvegardé, rafraichir la page pour observer dans la console :

```
Custom filter doFilter
```

Tout modification du filter sera prise en compte directement sans déployant le hook.

4.2.2.5 Struts-action

Créer une classe « CustomLoginAction » dans le package « com.sedona.jrebel.action » :

```

package com.sedona.jrebel.action;

import com.liferay.portal.kernel.struts.BaseStrutsPortletAction;
import com.liferay.portal.kernel.struts.StrutsPortletAction;
import com.liferay.portal.theme.ThemeDisplay;
import com.liferay.portal.kernel.util.WebKeys;
import javax.portlet.ActionRequest;
import javax.portlet.ActionResponse;
import javax.portlet.PortletConfig;
import javax.portlet.RenderRequest;
import javax.portlet.RenderResponse;
import javax.portlet.ResourceRequest;
import javax.portlet.ResourceResponse;

public class CustomLoginAction extends BaseStrutsPortletAction {

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

```

```

throws Exception {
    ThemeDisplay          themeDisplay          =
(ThemeDisplay)actionRequest.getAttribute(WebKeys.THEME_DISPLAY);
    Long currentuser = themeDisplay.getUserId();
    if (currentuser != null) {
        System.out.println("Process action de CustomLoginAction");
    }
    originalStrutsPortletAction.processAction(
        originalStrutsPortletAction, portletConfig, actionRequest,
        actionResponse);
}

public String render(
    StrutsPortletAction originalStrutsPortletAction,
    PortletConfig portletConfig, RenderRequest renderRequest,
    RenderResponse renderResponse)
throws Exception {
    System.out.println("Render de CustomLoginAction");
    return originalStrutsPortletAction.render(
        null, portletConfig, renderRequest, renderResponse);
}

public void serveResource(
    StrutsPortletAction originalStrutsPortletAction,
    PortletConfig portletConfig, ResourceRequest resourceRequest,
    ResourceResponse resourceResponse)
throws Exception {
    System.out.println("Process Resource de CustomLoginAction");
    originalStrutsPortletAction.serveResource(
        originalStrutsPortletAction, portletConfig, resourceRequest,
        resourceResponse);
}
}

```

Ajouter ensuite le code suivant dans le fichier « liferay-hook.xml » après la balise « servlet-filter-mapping » :

```

<struts-action>
  <struts-action-path>/login/login</struts-action-path>
  <struts-action-impl>com.sedona.jrebel.action.CustomLoginAction</struts-action-impl>
</struts-action>

```

Après sauvegarde du fichier, connecter vous pour observer dans la console le message suivant :

```
Process action de CustomLoginAction
```

Les modifications des classes actions de Struts seront prises en compte instantané.

4.2.2.6 Fichier « portal.properties »

Les propriétés du fichier « portal.properties » ne sont pas prises en compte directement par JRebel.

Par contre les modifications des classes Java définies dans le fichier sont prises en compte par JRebel.

4.2.3 Conclusion

Le rechargement à chaud par JRebel a permis de modifier :

- Les JSPs
- Les « indexer-post-processor »
- Les « services » (classes surchargeant les services de Liferay)
- Les « servlet-filter »
- Les « struts-action »

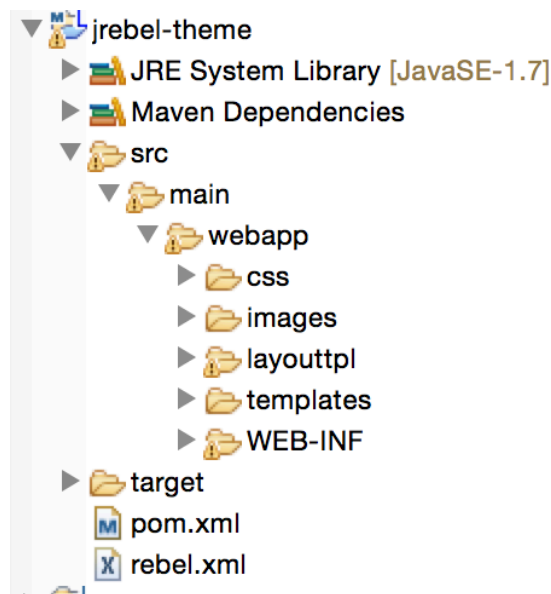
Chaque ajout et/ou modification de fichiers et/ou de classes java doit s'accompagner d'un message de JRebel dans la console.

Grâce à JRebel, plus besoin de déployer de hook, celui-ci est pris en compte à chaud.

4.3 Thème et Layout template

4.3.1 Préambule

Le thème avec les layouts templates doit être configuré avec le fichier « rebel.xml » et déployé dans Liferay :



Jrebel a généré le fichier « rebel.xml » au mauvais endroit. Déplacer le fichier rebel.xml dans « src/main/resources ».

De plus, Jrebel n'a pas généré le fichier correctement. Le code suivant est à supprimer :

```
<link target="/">
  <dir name="/.../jrebel/jrebel-hook/target/m2e-wtp/web-resources">
    <exclude name="/">
  </dir>
</link>
```

Redéployer le thème une fois ces configurations effectuées.

4.3.2 Configuration

Afin que le thème puisse se recharger à chaud grâce à JRebel, il faut que Liferay inclue le fichier portal-developper.properties.

4.3.3 Modification des fichiers

4.3.3.1 Templates

Modifier le fichier « portal_normal.vm » en ajoutant après la balise « body » :

Modification du template via JRebel

Après sauvegarde du fichier, sélectionner le thème pour voir la modification apparaître.

4.3.3.2 CSS

Créer un fichier « custom.css » dans « src/main/webapp/css » (si le fichier n'existe pas) et ajouter le code suivant :

```
@import "compass";

$color_breadcrumb: red;

.aui {
  .breadcrumb {
    background-color: $color_breadcrumb;
  }
}
```

Après sauvegarde du fichier, rafraichir la page pour voir que la couleur de fond de la navigation a changé de couleur.

4.3.3.3 Images

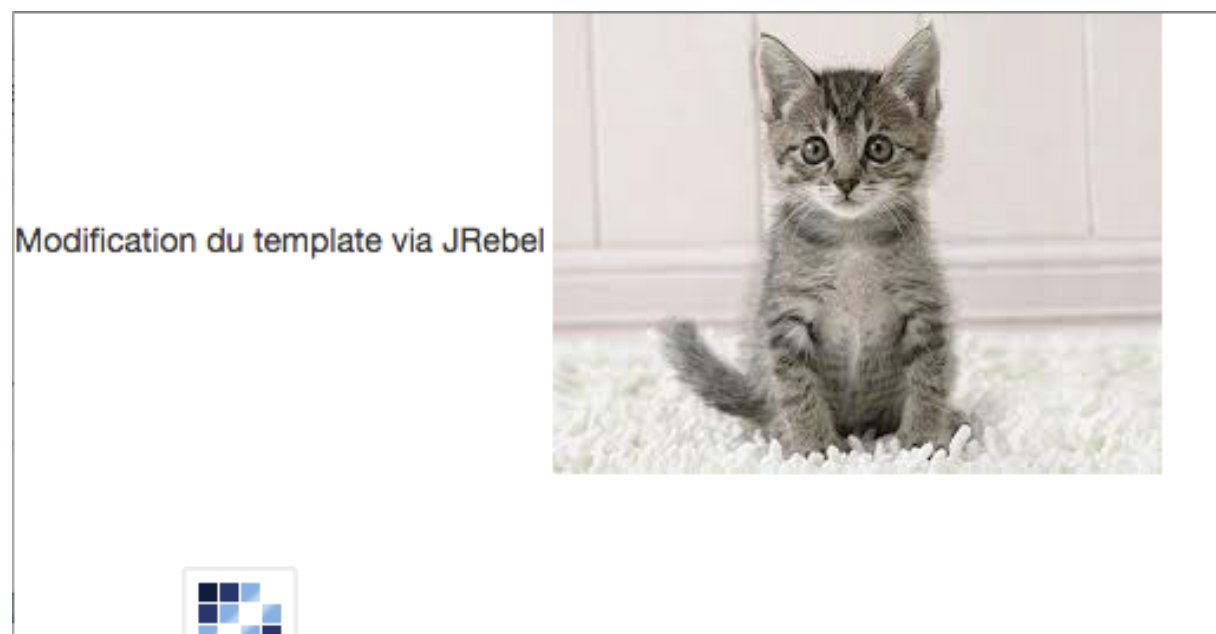
Ajouter une image dans le répertoire « images » et ajouter dans le fichier « portal_normal.vm » le code suivant :

```

```

« cat.jpeg » étant le nom de l'image

Après avoir rafraichi la page, on obtient :



4.3.3.4 Javascript

Créer un fichier « main.js » dans « src/main/webapp/js » et ajouter :

```
AUI().ready(
  /*
  This function gets loaded when all the HTML, not including the portlets, is
  loaded.
  */
  function() {
    alert('Javascript from JRebel');
  }
);

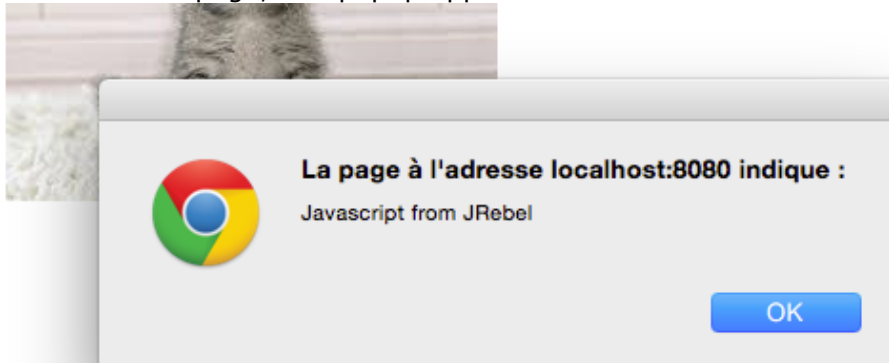
Liferay.Portlet.ready(
  /*
  This function gets loaded after each and every portlet on the page.

  portletId: the current portlet's id
  node: the Alloy Node object of the current portlet
  */
  function(portletId, node) {
  }
);

Liferay.on(
  'allPortletsReady',

  /*
  This function gets loaded when everything, including the portlets, is on
  the page.
  */
  function() {
  }
);
```

Après avoir rafraîchi la page, une popup apparaît :



4.3.3.5 Fichier liferay-look-and-feel.xml

Modifier le fichier « liferay-look-and-feel.xml » en ajoutant :

```
<setting configurable="false" key="jrebel" value="Value from liferay-look-and-feel.xml file" />
```

Ajouter également dans le fichier « portal_normal.vm » :

```
$theme_display.getThemeSetting('jrebel')
```

Rafraîchissez la page pour voir le résultat.

4.3.4 Conclusion

Voici la liste dont JRebel a permis de recharger à chaud :

- Templates
- CSS en SASS
- Images
- Javascript
- Propriétés du fichier liferay-look-and-feel.xml

LDS permet déjà de recharger à chaud certains éléments du thème tels que les « templates ».

Cependant il ne gère pas le SASS ou les propriétés du fichier « liferay-look-and-feel.xml » tandis que JRebel le permet.

5 Liferay Core

5.1 Préambule

Afin que JRebel puisse fonctionner avec les sources de Liferay, il est nécessaire de configurer les sources avant.

Dans un premier temps, copier le fichier ZIP des sources dans votre workspace. Puis extraire le fichier pour obtenir le dossier contenant toutes les sources de Liferay.

Dans le dossier des sources, il manque des fichiers qui sont des fichiers permettant de builder Liferay. La version EE ne fournit pas les fichiers car le build est fait par le support de Liferay. Cependant, pour JRebel, il est nécessaire de pouvoir faire des builds grâce à ant.

Les fichiers peuvent être facilement retrouver dans la version CE. Voici la liste des fichiers à récupérer et à installer dans le dossier des sources de Liferay :

- build-common-java.xml
- build-common-web.xml
- build-common.xml
- build-dist.xml
- build-maven.xml
- build-test-cluster.xml
- build-test-db-failover.xml
- build-test-db-sharding.xml
- build-test-db-upgrade.xml
- build-test-geronimo.xml
- build-test-glassfish.xml
- build-test-jboss-eap.xml
- build-test-jboss.xml
- build-test-jetty.xml
- build-test-jonas.xml
- build-test-ldap.xml
- build-test-plugins.xml
- build-test-tck.xml
- build-test-themes.xml
- build-test-tomcat.xml
- build-test-unit.xml
- build-test-weblogic.xml
- build-test-websphere-6.1.xml
- build-test-websphere-7.0.xml
- build-test-websphere-8.0.xml
- build-test.xml
- build.xml

Une fois les fichiers copiés, créer un fichier avec le nom « app.server.\${user.name}.properties » avec user.name votre identifiant OS.

Insérer dans le fichier :

```
app.server.parent.dir = %LIFERAY_HOME%
```

« LIFERAY_HOME » est le chemin vers le Liferay.

Lancer un terminal ou une fenêtre de ligne de commande puis aller dans le dossier des sources. Enfin lancer la commande suivante :

```
ant
```

Cette commande va compiler toutes les classes.

Ensuite, lancer la commande suivante :

```
ant setup-jrebel
```

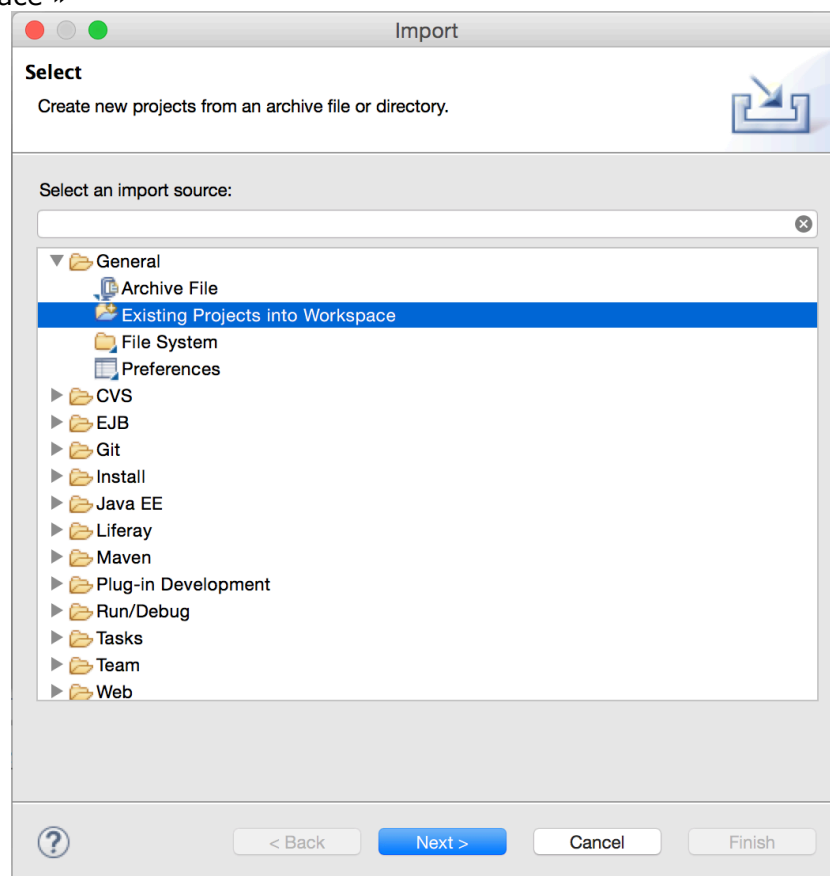
Cette commande va générer les 2 fichiers « rebel.xml » dans les dossiers suivants :

- %LIFERAY_HOME%/tomcat-7.0.42/lib/ext
- %LIFERAY_HOME%/tomcat-7.0.42/webapps/ROOT/WEB-INF/classes

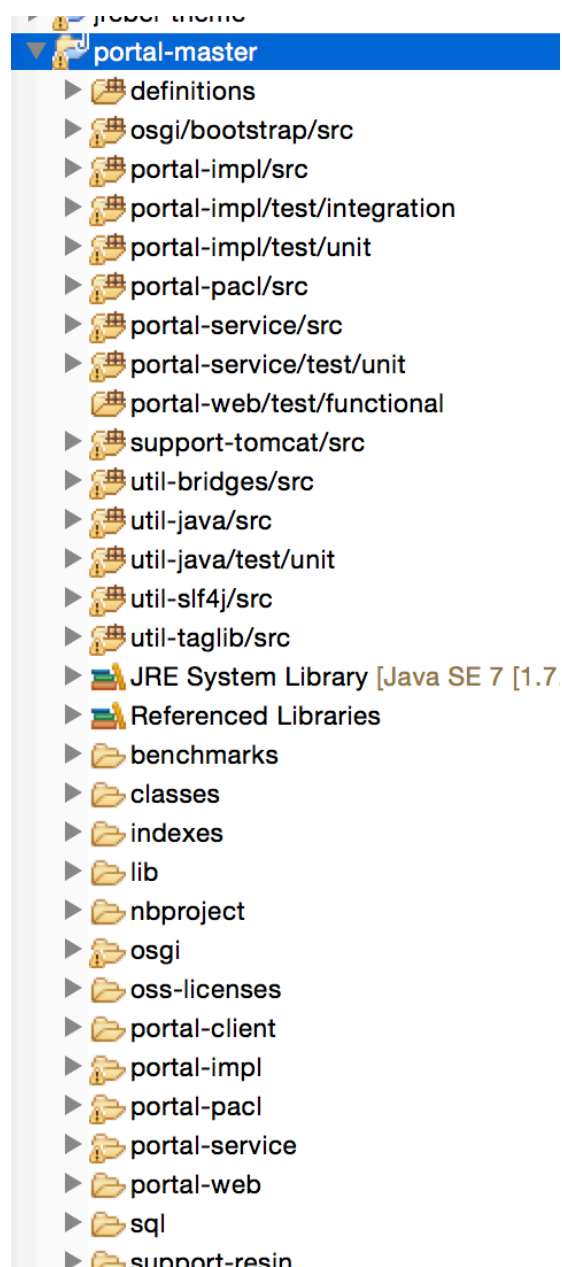
Cette configuration est suffisante et n'a point besoin de faire des modifications.

5.2 Import des sources dans Eclipse

Importer les sources par « File -> Import » puis choisir « General -> Existing Projects into Workspace into Workspace »



Faire « Next ». Cliquer sur « Browse » et pointer le chemin vers les sources Liferay. Une fois sélectionnée, cliquer sur « Finish ».

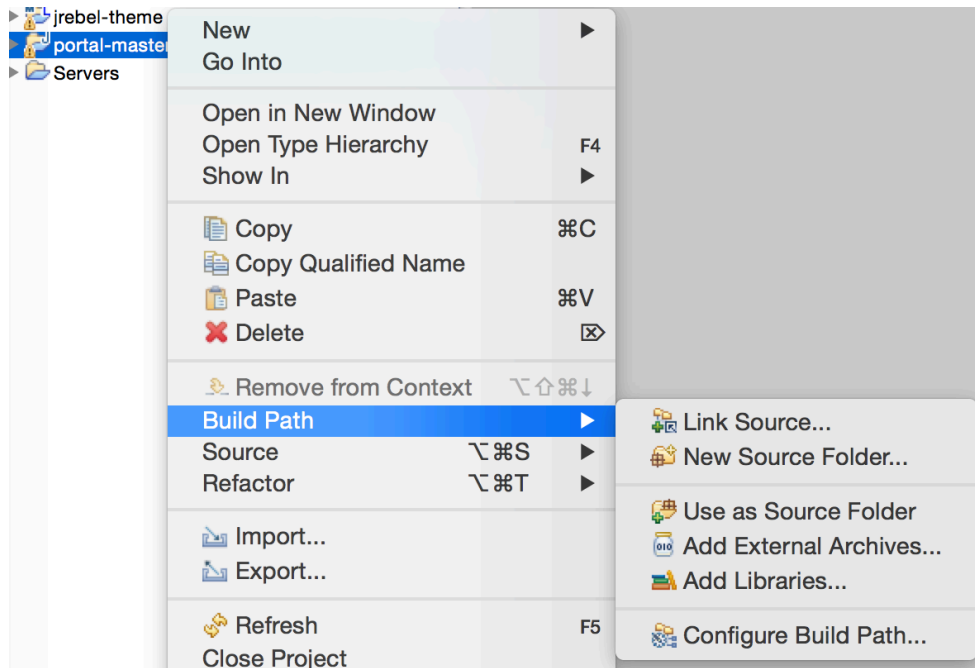


5.3 Configuration du projet

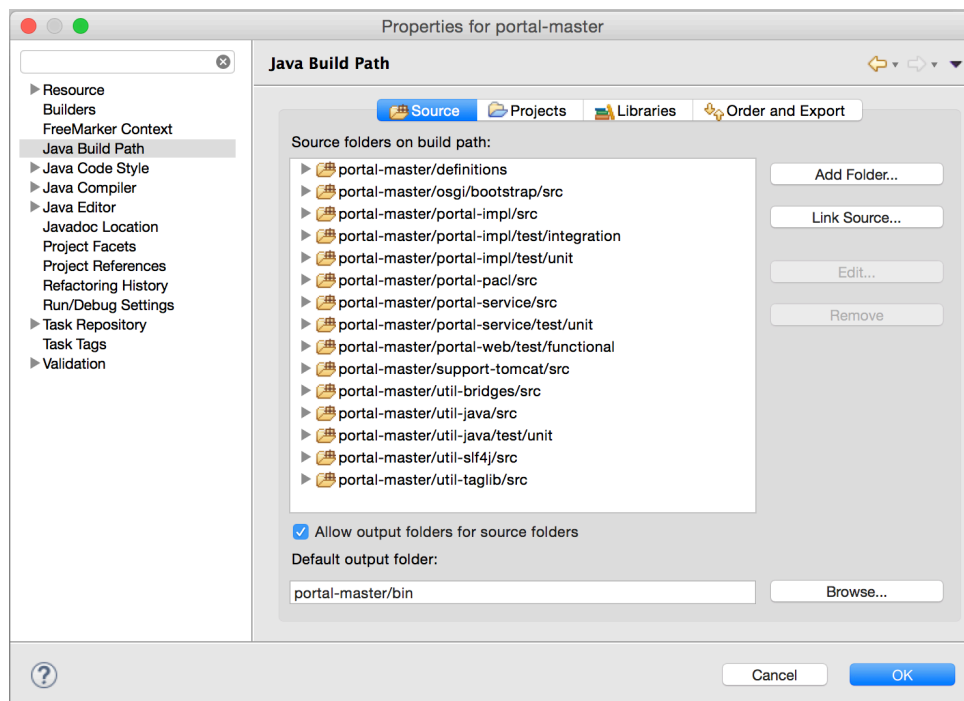
Lorsqu'elles ont été importées, la configuration Eclipse n'est pas la bonne. Les classes Java sont compilées dans le dossier « bin ».

Il faut donc reconfigurer le projet Liferay :

- Faire un clic droit sur le projet
- Aller « Build Path »
- Aller dans « Configure Build Path »



- Cliquer sur l'onglet « Source » :



- Cocher la case « Allow output folder for source folders »
- Modifier le dossier de destination des dossiers suivants :
 - portal-master/osgi/bootstrap/src => portal-master/osgi/bootstrap/classes
 - portal-master/portal-impl/src => portal-master/portal-impl/classes
 - portal-master/portal-pacl/src => portal-master/portal-pacl/classes
 - portal-master/portal-service/src => portal-master/portal-service/classes
 - portal-master/util-bridges/src => portal-master/util-bridges/classes
 - portal-master/util-java/src => portal-master/util-java/classes
 - portal-master/util-slf4j/src => portal-master/util-slf4j/classes
 - portal-master/util-taglib/src => portal-master/util-taglib/classes

- Cliquer sur « OK » pour valider le tout

5.4 Erreur

A l'affichage de la page d'accueil et après le premier démarrage de Liferay, une erreur survient :

```
Failed to load resource: the server responded with a status of 404 (Not Found)
Uncaught TypeError: Cannot read property 'init' of undefined
```

Une librairie Javascript ne se charge pas (erreur HTML 404).

Ce n'est pas un bug Liferay. Avec JRebel activé, Liferay ne démarre plus dans son dossier habituel mais dans le workspace du développeur. Ainsi, l'erreur rencontrée ici provient du fait que Liferay recherche le fichier dans le répertoire du développeur.

Pour corriger cela, ouvrir la classe « ComboServlet » et chercher la méthode « getResourceURL(ServletContext servletContext, String rootPath, String path) ». A la dernière ligne, remplacer « return null » par « return url ».

Sauvegarder, vous pouvez également voir si JRebel est correctement activé en voyant les lignes suivantes dans la console :

```
JRebel: Reloading class 'com.liferay.portal.servlet.ComboServlet'.
JRebel: Reloading class 'com.liferay.portal.servlet.ComboServlet$FileContentBag'.
```

5.5 Classes Java

Modifier n'importe quel classes des dossiers « portal-service », « portal-impl », « util-bridges », « util-java », « util-slf4j » et « util-taglib ».

Chaque modification d'une classe Java est validée par un message dans la console.

Par exemple, en modifiant la classe « com.liferay.portlet.login.action.LoginAction », on a :

```
JRebel: Reloading class 'com.liferay.portlet.login.action.LoginAction'.
```

5.6 Fichiers JSPs

Modifier, par exemple, la JSP « login.jsp » en ajoutant par exemple un texte puis rafraichir la page pour obtenir :

Sign In

Modification par JRebel

Screen Name

Password

Remember Me

[Sign In](#)

Hello

Welc

FIN DU DOCUMENT