JBoss Enterprise Application Platform 6.3

Guide de migration

À utiliser dans Red Hat JBoss Enterprise Application Platform 6

Last Updated: 2017-10-27

JBoss Enterprise Application Platform 6.3 Guide de migration

À utiliser dans Red Hat JBoss Enterprise Application Platform 6

Notice légale

Résumé

Cet ouvrage est un guide pour la migration de votre application en provenance de versions plus anciennes de Red Hat JBoss Enterprise Application Platform 6.

Table des matières

CHAPITRE 1. INTRODUCTION

1.1. RED HAT JBOSS ENTERPRISE APPLICATION PLATFORM 6

Red Hat JBoss Enterprise Application Platform 6 (JBoss EAP 6) est une plate-forme middleware construite sur la base de standards ouverts et compatibles avec Java Enterprise Edition 6. Elle intègre JBoss Application Server 7 avec un clustering de haute disponibilité, une messagerie, une mise en cache distribuée et autres technologies.

JBoss EAP 6 comprend une nouvelle structure modulaire qui permet aux services d'être activés seulement si nécessaire, améliorant ainsi la vitesse de démarrage.

La console de gestion et l'interface CLI rendent la modification des fichiers de configuration XML inutile et rajoutent la capacité d'encoder et d'automatiser des tâches.

En plus, JBoss EAP 6 comprend des frameworks de développement et des API pour développer rapidement des applications de Java EE sécurisées et évolutives.

Rapporter un bogue

1.2. GUIDE DE MIGRATION

JBoss EAP 6 est une version à la fois rapide, légère et puissante de spécification Java Enterprise Edition 6. L'architecture est construite sur le conteneur de services modulaires et autorise des services à la demande lorsque votre application les requiert. En raison de cette nouvelle architecture, les applications qui s'exécutent sur JBoss EAP 5 peuvent avoir besoin de modifications pour s'exécuter sur JBoss EAP 6.

Ce guide vise à documenter les changements requis pour exécuter et déployer avec succès les applications JBoss EAP 5.1 sur JBoss EAP 6. Il fournit des informations sur la façon de résoudre des problèmes de runtime et de déploiement et comment éviter des changements de comportement de l'application. Il s'agit de la première étape du passage à la nouvelle plate-forme. Une fois que l'application sera déployée avec succès et pourra être exécutée, vous pourrez mettre à niveau des composants individuellement pour utiliser les nouvelles fonctions et fonctionalités de JBoss EAP 6.

Rapporter un bogue

CHAPITRE 2. PRÉPARATION À LA MIGRATION

2.1. PRÉPARATION À LA MIGRATION

Étant donné que le serveur d'application est structuré différemment par rapport aux versions précédentes, il est conseillé de se renseigner et de s'organiser à l'avance avant d'essayez de migrer votre application.

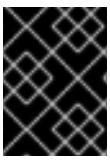
1. **Voir ce qu'il y a de nouveau et de différent dans JBoss EAP 6**
   Il y a un certain nombre de changements dans cette version qui risquent d'avoir un impact dans le déploiement des applications de JBoss EAP 5. Ceux-ci incluent les changements dans la structure des fichiers de répertoire, des scripts, de la configuration du déploiement, du chargement de classes et des recherches JNDI. Voir Section 2.2, « Ce qu'il y a de nouveau et de différent dans JBoss EAP 6 » pour davantage d'informations.

2. **Revue de la documentation du Guide de démarrage**
   Veillez à réviser le chapitre intitulé *Get Started Developing Applications* du *Development Guide* de JBoss EAP 6 dans https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/. Ce chapitre contient des informations importantes sur ce qui suit :

   - Java EE 6
   - Le système de chargement de la nouvelle classe modulaire
   - Changements de la structure de fichiers
   - Comment télécharger et installer JBoss EAP 6
   - Comment télécharger et installer JBoss Developer Studio
   - Comment configurer Maven dans votre environnement de développement.
   - Comment télécharger et exécuter des exemples d'applications Quickstart fournies avec ce produit.

3. **Apprenez à utiliser les dépendances JBoss EAP 6 dans votre projet Maven**
   Veillez à réviser le chapitre intitulé *Maven Guide* du *Development Guide* de JBoss EAP 6 dans https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/. La section *Manage Project Dependencies* contient d'importantes informations sur la façon de configurer votre projet par les artifacts BOM (Bill of Material) de JBoss EAP.

4. **Analyse et compréhension de votre application**
   Chaque application est unique et vous devez comprendre tous les composants et l'architecture de l'application existante avant de tenter la migration.



IMPORTANT

Avant de procéder aux modifications de votre application, veiller à sauvegarder une copie.

Rapporter un bogue

2.2. CE QU'IL Y A DE NOUVEAU ET DE DIFFÉRENT DANS JBOSS EAP 6

Introduction

Voici une liste des différences notables entre JBoss EAP 6, et sa dernière version.

Chargement de classes modulaire

Dans JBoss EAP 5, l'architecture de chargement était hiérarchique. Dans JBoss EAP 6, le chargement de classe est basé sur des modules de JBoss. Cela permet un véritable isolement de l'application, masque également les classes d'implémentation serveur, et charge uniquement les classes nécessaires à votre application. Le chargement de classes est simultané pour améliorer les performances. Les applications écrites pour JBoss EAP 5 doivent être modifiées afin de spécifier les dépendances de modules et dans certains cas, pour reconditionner les archives. Pour plus d'informations, voir la section *Class Loading and Modules*du *Development Guide* de JBoss EAP 6 à l'adresse suivante https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/.

Gestion des domaines

Dans JBoss EAP 6, le serveur peut être exécuté en tant que serveur autonome ou en domaine géré. Dans un domaine géré, vous pouvez configurer des groupes entiers de serveurs à la fois, en gardant les configurations synchronisées dans tout le réseau de serveurs. Cela ne devrait pas avoir d'impact sur les applications générées dans les versions précédentes mais pourra simplifier la gestion des déploiements vers des serveurs multiples. Pour plus d'informations, voir *About Managed Domains* (Domaines gérés) dans le chapitre *Administration and Configuration Guide*(Guide de démarrage pour le développement d'applications) de JBoss Enterprise Application Platform 6 https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/.

Configuration de déploiement

Serveurs autonomes et Domaines gérés

JBoss EAP 5 utilisait une configuration de déploiement basée profil. Ces profils se situaient dans le répertoire ***EAP_HOME***`/server/`. Les applications contenaient souvent des fichiers de configurations multiples pour la sécurité, la base de données, l'adaptateur de ressources, et les autres configurations. Dans JBoss EAP 6, la configuration du déploiement est effectuée par un fichier. Ce fichier est utilisé pour configurer tous les services et sous-systèmes utilisés pour le déploiement. Un serveur autonome est configuré par le fichier ***EAP_HOME***`/standalone/configuration/standalone.xml`. Pour les serveurs qui exécutent dans un domaine géré, le serveur est configuré par le fichier ***EAP_HOME***`/domain/configuration/domain.xml`. Les informations contenues dans les fichiers de configuration de JBoss EAP 5 doivent être migrées dans le nouveau fichier de configuration unique.

Ordre de déploiements

JBoss EAP 6 est pourvue d'une initialisation rapide et instantannée des déploiements qui améliore la performance. Dans la plupart des cas, le serveur d'applications est en mesure de déterminer à l'avance les dépendances automatiquement et de choisir la stratégie de déploiement la plus efficace. Toutefois, les applications de JBoss EAP 5 composées de plusieurs modules déployés comme EAR qui utilisent des recherches JNDI héritées au lieu d'entrées resource-ref ou injection CDI peuvent nécessiter des modifications de configuration.

Structures de répertoires et scripts

Comme nous l'avons déjà expliqué, JBoss EAP 6 n'utilise plus la configuration de déploiement basée profil, donc il n'y a plus de répertoire ***EAP_HOME***`/server/`. Les fichiers de configuration des serveurs autonomes sont maintenant situés dans le répertoire ***EAP_HOME***`/standalone/configuration/` et les déploiements se situent dans le répertoire

autonome ***EAP_HOME*/standalone/deployments/**. Pour les serveurs qui exécutent dans un domaine géré, les fichiers de configuration sont situés dans le répertoire ***EAP_HOME*//domain/configuration/**.

Dans JBoss EAP 5, le script Linux ***EAP_HOME*/bin/run.sh** ou script Windows ***EAP_HOME*/bin/run.bat** était utilisé pour démarrer le serveur. Dans JBoss EAP 6, le script de démarrage du serveur dépend de la façon dont vous exécutez sur le serveur. Le script Linux ***EAP_HOME*/bin/standalone.sh** ou le script Windows ***EAP_HOME*/bin/standalone.bat** est utilisé pour démarrer un serveur autonome. Le script Linux ***EAP_HOME*/bin/domain.sh** ou le script Windows ***EAP_HOME*/bin/domain.bat** est utilisé pour démarrer un domaine géré.

Recherches JNDI

JBoss EAP 6 utilise une syntaxe d'espace-noms JNDI standard portable. Les applications écrites pour JBoss EAP 5 utilisant les recherches JNDI doivent être modifiées afin de suivre la convention de syntaxe d'espace-noms JNDI standard. Pour plus d'informations sur la syntaxe d'appellation JNDI, voir Section 3.1.8.2, « Noms JNDI EJB portables » .

Pour obtenir des informations supplémentaires, voir *New and Changed Features in JBoss EAP 6* (Fonctionnalités Nouvelles ou Modifiées de JBoss EAP 6) du chapitre *Development Guide* (Guide de developpement) de JBoss EAP 6 à https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/.

Rapporter un bogue

2.3. VÉRIFIER LA LISTE DES FONCTIONNALITÉS DÉPRÉCIÉES ET NON PRISES EN CHARGE.

Avant de migrer votre application, vous devez réaliser que certaines fonctionnalités qui étaient disponibles dans les versions précédentes de JBoss EAP risquent d'être dépréciées ou ne seront plus prises en charge. Pour en avoir une liste complète, voir la section *Unsupported Features* de *Release Notes* de JBoss EAP 6 qui se situe sur le Portail clients https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/.

Rapporter un bogue

CHAPITRE 3. MIGRATION DE VOTRE APPLICATION

3.1. CHANGEMENTS REQUIS PAR LA PLUPART DES APPLICATIONS

3.1.1. Revue des changements de migration requis par la plupart des applications

Le chargement des classes et les changements de configuration de JBoss Enterprise Application Platform 6 auront un impact dans presque toutes les applications. JBoss EAP 6 utilise également la nouvelle syntaxe de nommage JNDI standard portable. Ces changements auront un impact sur la plupart des applications, donc nous vous suggérons de consulter ces informations tout d'abord quand vous migrerez votre application.

1. Section 3.1.2.1, « Mise à jour de l'application en raison des changements de chargement de classes »
2. Section 3.1.6.1, « Mise à jour de l'application en raison des changements de configuration »
3. Section 3.1.8.1, « Mise à jour des noms d'espace-noms JNDI d'application »

Rapporter un bogue

3.1.2. Changements au niveau du chargement des classes

3.1.2.1. Mise à jour de l'application en raison des changements de chargement de classes

Le chargement des classes représente un changement important dans JBoss EAP 6 et aura un impact dans presque toutes les applications. Revoir les informations suivantes pour commencer avant de migrer votre application.

1. Veuillez tout d'abord vous pencher sur le packaging de votre application et de ses dépendances. Pour plus d'informations, voir Section 3.1.2.3, « Mise à jour des dépendances d'applications en raison des changements de chargement de classes »
2. Si votre application a une journalisation, vous devez spécifier les dépendances du module qui conviennent. Voir Section 3.1.4.1, « Modifier les Dépendances de Logging »
3. En raison des changements dans le chargement de classe modulaire, vous devrez sans doute modifier la structure de votre EAR ou WAR. Voir Section 3.1.5.1, « Modifier l'empaquetage des EAR et des WAR » pour plus d'informations.

Rapporter un bogue

3.1.2.2. Les Dépendances de modules

Résumé

Un module ne peut accéder qu'à ses propres classes et aux classes de modules sur lesquelles il possède une dépendance implicite ou explicite.

Procédure 3.1. Les Dépendances de modules

1. **Comment fonctionnent les dépendances implicites ?**
   Les déployeurs qui se trouvent dans le serveur ajoutent implicitement et automatiquement certaines dépendances de module communément utilisées, comme `javax.api` ou `sun.jdk`.

Cela rend les classes visibles au déploiement au moment de l'exécution et soulage le développeur de la tâche d'ajouter explicitement les dépendances. Quand et comment ces dépendances implicites sont ajoutées est expliqué dans la section *Implicit Module Dependencies* (Dépendances de modules implicites) du chapitre *Class Loading and Modules*(Chargement de classes et Modules) du *Development Guide* (Guide de développement) de JBoss EAP 6 https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/.

2. **Comment fonctionnent les dépendances explicites ?**
   Pour les autres classes, les modules doivent être spécifiés explicitement sinon les dépendances manquantes entraînent des erreurs de déploiement ou de runtime. Si une dépendance est manquante, vous verrez des traces du style **`ClassNotFoundExceptions`** ou **`NoClassDefFoundErrors`** dans le journal du serveur. Si plus d'un module charge le même JAR ou si un module charge une classe qui étend une classe chargée dans un module différent, vous verrez des traces **`ClassCastExceptions`** dans le journal du serveur. Pour spécifier des dépendances explicitement, modifier **`MANIFEST.MF`** ou bien, créer un fichier de descripteur de déploiement spécifique à JBoss **`jboss-deployment-structure.xml`**. Pour plus d'informations sur les dépendances de modules, voir *Overview of Class Loading and Modules* (Aperçu Général des Classes de chargement et Modules) au chapitre *Class Loading and Module* (Guide de démarrage pour le développement d'applications) du *Development Guide* (Guide de développement) de JBoss EAP 6 https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/.

Rapporter un bogue

3.1.2.3. Mise à jour des dépendances d'applications en raison des changements de chargement de classes

Résumé

Le chargement de classes de JBoss EAP 6 est considérablement différent par rapport aux versions précédentes de JBoss Enterprise Application Platform. Le chargement de classes est maintenant basé sur le projet JBoss Modules. Plutôt que d'avoir un chargeur de classe unique, hiérarchique qui charge tous les JAR dans un chemin d'accès de la classe plat, chaque bibliothèque devient un module qui se relie seulement aux modules précis dont elle dépend. Les déploiements JBoss EAP 6 sont également des modules et n'ont pas accès aux classes qui sont définies dans les JAR du serveur d'application, à moins qu'une dépendance explicite sur ces classes soit définie. Certaines dépendances de module définies par le serveur d'applications sont configurées automatiquement pour vous. Par exemple, si vous déployez une application Java EE, une dépendance sur l'API de Java EE sera ajoutée à votre module automatiquement. Pour une liste complète des dépendances qui sont automatiquement ajoutées, voir *Implicit Module Dependencies* dans le chapitre intitulé *Class Loading and Modules*du *Development Guide* de JBoss Enterprise Application Platform 6 à https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/.

Tâches

Quand vous migrez votre application dans JBoss EAP 6, vous devrez sans doute effectuer une des tâches suivantes en raison des changements au niveau du chargement des classes modulaires.

- Section 3.1.2.2, « Les Dépendances de modules »
- Section 4.1.3, « Utiliser Tattletale pour trouver des dépendances d'applications »
- Section 3.1.3.1, « Créer ou modifier des fichiers qui contrôlent le chargement de classes dans JBoss EAP 6 »
- Section 3.1.3.3, « Empaquetage des ressources dans le nouveau système de chargement de classes modulaires »

Rapporter un bogue

3.1.3. Changements dans les fichiers de configuration

3.1.3.1. Créer ou modifier des fichiers qui contrôlent le chargement de classes dans JBoss EAP 6

Résumé

En raison du changement de chargement de classes modulaires dans JBoss EAP 6, vous devrez peut-être créer ou modifier un ou plusieurs fichiers pour ajouter des dépendances ou pour empêcher les dépendances automatiques de charger. Pour plus d'informations sur *Class Loading and Modules*(Class Loading Precedence), voir le chapitre *Development Guide* (Chargement de classes et Modules) du Guide de développement de JBoss EAP 6 https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/.

Les fichiers suivants sont utilisés pour contrôler le chargement de classes dans JBoss EAP 6.

jboss-web.xml

Si vous avez défini un élément **<class-loading>** dans le fichier **jboss-web.xml**, vous devrez le supprimer. Le comportement évoqué dans JBoss EAP 5 est maintenant le comportement de chargement par défaut dans JBoss EAP 6, donc ce n'est plus nécessaire. Si vous ne souhaitez pas supprimer cet élément, vous verrez ParseError et XMLStreamException dans votre log de serveur.

Il s'agit d'un exemple d'élément **<class-loading>** du fichier **jboss-web.xml** qui est dé-commenté.

```
<!DOCTYPE jboss-web PUBLIC
    "-//JBoss//DTD Web Application 4.2//EN"
    "http://www.jboss.org/j2ee/dtd/jboss-web_4_2.dtd">
<jboss-web>
<!--
    <class-loading java2ClassLoadingCompliance="false">
        <loader-repository>
            seam.jboss.org:loader=MyApplication
            <loader-repository-
config>java2ParentDelegation=false</loader-repository-config>
        </loader-repository>
    </class-loading>
 -->
 </jboss-web>
```

MANIFEST.MF

Édité manuellement

Selon les composants ou modules que votre application utilise, vous devrez peut-être ajouter une ou plusieurs dépendances à ce fichier. Vous pouvez les ajouter par l'une des entrées suivantes **Dependencies** ou **Class-Path** .

Voici un exemple de **MANIFEST.MF** édité par un développeur :

```
Manifest-Version: 1.0
Dependencies: org.jboss.logmanager
Class-Path: OrderManagerEJB.jar
```

Si vous modifiez ce fichier, veillez à inclure un caractère de nouvelle ligne en fin de fichier.

Créé par Maven

Si vous utilisez Maven, vous devrez modifier votre fichier **pom.xml** pour créer des dépendances pour le fichier **MANIFEST.MF**. Si votre application utilise EJB 3.0, vous aurez sans doute une section du fichier **pom.xml** qui va ressembler à ce qui suit :

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-ejb-plugin</artifactId>
    <configuration>
        <ejbVersion>3.0</ejbVersion>
    </configuration>
</plugin>
```

Si le code EJB 3.0 utilise **org.apache.commons.log**, vous aurez besoin de cette dépendance dans le fichier **MANIFEST.MF**. Pour créer cette dépendance, ajouter l'élément **<plugin>** au fichier **pom.xml** comme suit :

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-ejb-plugin</artifactId>
    <configuration>
        <ejbVersion>3.0</ejbVersion>
        <archive>
            <manifestFile>src/main/resources/META-
INF/MANIFEST.MF</manifestFile>
        </archive>
    </configuration>
</plugin>
```

Dans l'exemple ci-dessus, le fichier **src/main/resources/META-INF/MANIFEST.MF** ne doit comprendre uniquement la dépendance suivante :

```
Dépendences: org.apache.commons.logging
```

Maven va créer un fichier complet **MANIFEST.MF** :

```
Manifest-Version: 1.0
Dependencies: org.apache.commons.logging
```

jboss-deployment-structure.xml

Ce fichier est un descripteur de déploiement spécifique à JBoss qui peut être utilisé pour contrôler un chargement de classes en toute précision. Comme **MANIFEST.MF**, ce fichier peut être utilisé pour ajouter des dépendances. Peut également empêcher l'ajout automatique de dépendances, définir des modules supplémentaires, modifier un comportement de chargement de classe isolé de déploiement EAR, et ajouter des racines de ressources supplémentaires à un module.

Il s'agit d'un exemple de fichier **jboss-deployment-structure.xml** qui ajoute une dépendance pour le module JSF 1.2 et empêche le chargement automatique du module JSF 2.0.

```
<jboss-deployment-structure xmlns="urn:jboss:deployment-structure:1.0">
  <deployment>
    <dependencies>
      <module name="javax.faces.api" slot="1.2" export="true"/>
      <module name="com.sun.jsf-impl" slot="1.2" export="true"/>
    </dependencies>
  </deployment>
  <sub-deployment name="jboss-seam-booking.war">
    <exclusions>
      <module name="javax.faces.api" slot="main"/>
      <module name="com.sun.jsf-impl" slot="main"/>
    </exclusions>
    <dependencies>
      <module name="javax.faces.api" slot="1.2"/>
      <module name="com.sun.jsf-impl" slot="1.2"/>
    </dependencies>
  </sub-deployment>
</jboss-deployment-structure>
```

Pour obtenir des informations supplémentaires sur ce fichier, voir Section 3.1.3.2, « jboss-deployment-structure.xml ».

application.xml

Dans les versions précédentes de JBoss EAP, vous pouviez contrôler l'ordre des déploiements dans un EAR par le fichier **jboss-app.xml**. Ce n'est plus le cas. Les spécifications de Java EE6 fournissent l'élément **<initialize-in-order>** dans **application.xml** qui permet de contrôler l'ordre dans lequel les modules Java EE sont déployés dans un EAR.

Dans la plupart les cas, vous n'aurez pas besoin d'indiquer l'ordre de déploiement. Si votre application fait des injections de dépendances et des resource-refs pour se référer à des composants de modules externes, dans la plupart des cas, l'élément **<initialize-in-order>** ne sera pas requis car le serveur de l'application sera capable implicitement de déterminer la meilleure façon d'ordonnancer les composants.

SI vous avez une application qui contient un **myBeans.jar** et un **myApp.war** regroupés dans un **myApp.ear**. Un servlet dans **myApp.war** utilise une annotation **@EJB** pour injecter un bean à partir de **myBeans.jar**. Dans ce cas, le serveur d'application est conscient qu'il doit veiller à ce que le composant EJB soit disponible avant que le servlet ne démarre et vous n'êtes pas obligé d'utiliser l'élément **<initialize-in-order>**.

Cependant, si ce servlet utilise l'ancien style de référence JNDI lookup distant comme suit pour accéder au bean, vous devrez spécifier l'ordonnancement des modules.

```
init() {
  Context ctx = new InitialContext();
  ctx.lookup("TheBeanInMyBeansModule");
}
```

Dans ce cas, le serveur n'est pas en mesure de déterminer que le composant EJB se trouve dans **myBeans.jar** et vous devrez enforcer que les composants du **myBeans.jar** soient initialisés et démarrés avant les composants qui se trouvent dans **myApp.war**. Pour cela, définir l'élément

<initialize-in-order> à **true** et indiquer l'ordre des modules **myBeans.jar** et **myApp.war** dans le fichier **application.xml**.

Voici un exemple qui utilise l'élément **<initialize-in-order>** pour contrôler l'ordre de déploiement. Le **myBeans.jar** est déployé avant le fichier **myApp.war**.

```
<application xmlns="http://java.sun.com/xml/ns/javaee"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
version="6"
             xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
             http://java.sun.com/xml/ns/javaee/application_6.xsd">
    <application-name>myApp</application-name>
    <initialize-in-order>true</initialize-in-order>
    <module>
        <ejb>myBeans.jar</ejb>
    </module>
    <module>
        <web>
            <web-uri>myApp.war</web-uri>
            <context-root>myApp</context-root>
        </web>
    </module>
</application>
```

Le schéma du fichier **application.xml** se trouve ici http://java.sun.com/xml/ns/javaee/application_6.xsd.

NOTE

Vous devez savoir que lorsque vous définissez l'élément **<initialize-in-order>** à **true** vous ralentissez le déploiement. Il est préférable de définir les dépendances appropriées à l'aide d'injections de dépendances ou de resource-refs, ce qui donne au contenant une plus grande souplesse en matière d'optimisation des déploiements.

jboss-ejb3.xml

Le descripteur de déploiement **jboss-ejb3.xml** remplace le descripteur de déploiement **jboss.xml** et s'ajoute aux fonctions fournies par le descripteur de déploiement **ejb-jar.xml** fourni par Java Enterprise Edition (EE). Le nouveau fichier est incompatible avec **jboss.xml**, et le fichier **jboss.xml** est maintenant ignoré dans les déploiements.

login-config.xml

Le fichier **login-config.xml** n'est plus utilisé dans la configuration de la sécurité. La sécurité est maintenant configurée dans l'élément **<security-domain>** du fichier de configuration du serveur. Pour le serveur autonome, il s'agit du fichier **standalone/configuration/standalone.xml**. Si vous exécutez votre serveur dans un domaine géré, il s'agit du fichier **domain/configuration/domain.xml**.

Rapporter un bogue

3.1.3.2. jboss-deployment-structure.xml

jboss-deployment-structure.xml est un descripteur de déploiement optionnel de JBoss EAP 6. Ce descripteur de déploiement fournit un contrôle pour le chargement de classes dans le déploiement.

Le schéma XML de ce descripteur de déploiement se trouve dans ***EAP_HOME*/docs/schema/jboss-deployment-structure-1_2.xsd**

Rapporter un bogue

3.1.3.3. Empaquetage des ressources dans le nouveau système de chargement de classes modulaires

Résumé

Dans les versions précédentes de JBoss EAP, toutes les ressources qui se trouvaient à l'intérieur du répertoire **WEB-INF/** étaient ajoutées au chemin WAR. Dans JBoss EAP 6, les artefacts d'applications web ne sont chargées qu'à partir des répertoires **WEB-INF/classes** et **WEB-INF/lib**. Les erreurs d'empaquetage d'artifacts d'applications dans les emplacements spécifiés peuvent résulter en **ClassNotFoundException**, **NoClassDefError**, ou autres erreurs de runtime.

Pour résoudre ces erreurs de chargement de classes, vous devez soit modifier la structure de votre archive d'application, ou bien définir un module personnalisé.

Modifier l'empaquetage des ressources

Pour que les ressources ne soient disponibles qu'aux applications, vous devez grouper les fichiers de propriétés, les JAR, ou autres artifacts avec le WAR en les déplaçant dans le répertoire **WEB-INF/classes/** ou **WEB-INF/lib/**. Cette approche est décrite davantage en détails dans : Section 3.1.3.4, « Changer la location des propriétés de ResourceBundle »

Créer un module personnalisé

Si vous souhaitez rendre les ressources personnalisées disponibles auprès de toutes les applications exécutant sur le serveur JBoss EAP 6, vous devrez créer un module personnalisé. Cette approche est décrite plus en détails dans : Section 3.1.3.5, « Créer un module personnalisé »

Rapporter un bogue

3.1.3.4. Changer la location des propriétés de ResourceBundle

Résumé

Dans les versions précédentes de JBoss EAP, le répertoire ***EAP_HOME*/server/*SERVER_NAME*/conf/** se trouvait dans le chemin de classe disponible à l'Application. Pour rendre les propriétés disponibles sur le chemin de classe de l'application dans JBoss EAP 6, vous devez les empaqueter dans votre application.

Procédure 3.2. Changer la location des propriétés de ResourceBundle

1. Si vous déployez une archive WAR, vous devez empaqueter ces propriétés dans le dossier **WEB-INF/classes/** du WAR.

2. Si vous voulez que ces propriétés soient accessibles à toutes les composantes d'un EAR, alors vous devrez les empaqueter dans un JAR, en tant que root, puis mettre le JAR dans le dossier **lib /** du EAR.

Rapporter un bogue

3.1.3.5. Créer un module personnalisé

La procédure suivante décrit comment créer un module personnalisé pour rendre les fichiers de propriétés et les autres ressources disponibles à toutes les applications qui exécutent sur le serveur JBoss EAP.

Procédure 3.3. Créer un module personnalisé

1. Créer et compléter la structure de répertoire **module/**.

   a. Créer une structure de répertoire sous le répertoire ***EAP_HOME*/module** qui contiendra les fichiers et les JAR.

   ```
   $ cd EAP_HOME/modules/
   $ mkdir -p myorg-conf/main/properties
   ```

   b. Déplacer les fichiers de propriétés dans le répertoire ***EAP_HOME*/modules/myorg-conf/main/properties/** que vous avez créé à l'étape précédente.

   c. Créer un fichier **module.xml** dans le répertoire ***EAP_HOME*/modules/myorg-conf/main/** qui devra contenir l'XML suivant:

   ```
   <module xmlns="urn:jboss:module:1.1" name="myorg-conf">
       <resources>
           <resource-root path="properties"/>
       </resources>
   </module>
   ```

2. Modifier le sous-système **ee** du fichier de configuration du serveur. Vous pourrez utiliser JBoss CLI, ou bien, vous pourrez éditer le fichier manuellement.

   - Suivez ces étapes pour modifier le fichier de configuration du serveur par le JBoss CLI.

     a. Démarrez le serveur et connectez-vous au Management CLI.

     - Dans Linux, saisir ce qui suit au niveau de la ligne de commande :

     ```
     $ EAP_HOME/bin/jboss-cli.sh --connect
     ```

     - Dans Windows, saisir ce qui suit au niveau de la ligne de commande :

     ```
     C:\>EAP_HOME\bin\jboss-cli.bat --connect
     ```

     Vous devriez voir apparaître le résultat suivant :

     ```
     Connected to standalone controller at localhost:9999
     ```

     b. Pour créer l'élément <global-modules> **myorg-conf** dans le sous-système **ee**, tapez ce qui suit au niveau de la ligne de commande :

     ```
     /subsystem=ee:write-attribute(name=global-modules, value=
     [{"name"=>"myorg-conf","slot"=>"main"}])
     ```

Vous devriez voir apparaître le résultat suivant :

```
{"outcome" => "success"}
```

- Suivre ces étapes si vous préférez éditer manuellement le fichier de configuration du serveur.

  a. Arrêter le serveur et ouvrir le fichier de configuration du serveur dans un éditeur de texte. Si vous exécutez sur un serveur autonome, il s'agira du fichier ***EAP_HOME*/standalone/configuration/standalone.xml**. Il s'agira du fichier ***EAP_HOME*/domain/configuration/domain.xml** si vous exécutez sur un domaine géré.

  b. Chercher le sous-système **ee** et ajouter le module global de **myorg-conf**. Ce qui suit est un exemple d'élément du sous-système **ee** modifié pour inclure l'élément **myorg-conf** :

```
<subsystem xmlns="urn:jboss:domain:ee:1.0" >
    <global-modules>
        <module name="myorg-conf" slot="main" />
    </global-modules>
</subsystem>
```

3. Si vous copiez un fichier nommé **my.properties** dans l'emplacement de module qui convient, vous pourrez alors charger les fichiers de propriétés en utilisant un code qui ressemble à ceci :

```
Thread.currentThread().getContextClassLoader().getResource("my.properties");
```

Rapporter un bogue

3.1.4. Changements au niveau du logging

3.1.4.1. Modifier les Dépendances de Logging

Résumé

JBoss LogManager supporte les serveurs frontaux pour tous les frameworks de journalisation, donc vous pouvez conserver votre code de logging actuel ou passer à la nouvelle infrastructure de journalisation de JBoss. Quelle que soit votre décision, à cause des changements au niveau du chargement de classes modulaires, vous devrez probablement modifier votre application pour ajouter les dépendances nécessaires.

Procédure 3.4. Mise à jour du code de logging d'application

1. Section 3.1.4.2, « Mise à jour du Code d'application pour les frameworks de Logging (journalisation) de tierce partie »

2. Section 3.1.4.3, « Modifier le code pour utiliser le nouveau framework de JBoss Logging »

Rapporter un bogue

3.1.4.2. Mise à jour du Code d'application pour les frameworks de Logging (journalisation) de tierce partie

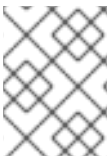
Résumé

Dans JBoss EAP 6, les dépendances de logging des frameworks de tierce partie connues comme Apache Commons Logging, Apache log4j, SLF4J, et Java Logging sont ajoutées par défaut. Dans la plupart des cas, il est préférable d'utiliser le framework de journalisation fourni dans le conteneur JBoss EAP. Toutefois, si vous avez besoin de fonctionnalités spécifiques fournies par des frameworks de tierce partie, vous devrez exclure le module JBoss EAP correspondant de votre déploiement. Notez que même si votre déploiement utilise un framework de journalisation de tierce partie, les logs du serveur continueront d'utiliser la configuration de sous-système de journalisation de JBoss EAP.

Les procédures suivantes montrent comment exclure le module **`org.apache.log4j`** de JBoss EAP 6 de votre déploiement. La première procédure fonctionne sur n'importe quelle version de JBoss EAP 6. La seconde procédure s'applique uniquement à JBoss EAP 6.3 ou version supérieure.

Procédure 3.5. Configurer JBoss EAP 6 pour utiliser log4j.properties ou le fichier log4j.xml

Cette procédure fonctionne pour toutes les versions de JBoss EAP 6.



NOTE

Comme cette méthode utilise un fichier de configuration log4j, vous ne pourrez plus changer la configuration de journalisation log4j en cours d'exécution.

1. Créer un **`jboss-deployment-structure.xml`** avec le contenu suivant :

```
<jboss-deployment-structure>
    <deployment>
        <!-- Exclusions allow you to prevent the server from
automatically adding some dependencies -->
        <exclusions>
            <module name="org.apache.log4j" />
        </exclusions>
    </deployment>
</jboss-deployment-structure>
```

2. Mettez le fichier **`jboss-deployment-structure.xml`** dans le répertoire **`META-INF/`** ou **`WEB-INF/`** si vous déployez un WAR, ou dans **`META-INF/`** si vous déployez un EAR. Si votre déploiement inclut des déploiements dépendants supplémentaires, vous devrez également exclure le module de chaque sous-déploiement.

3. Inclure **`log4j.properties`** ou le fichier **`log4j.xml`** dans le répertoire **`lib/`** de votre EAR, ou dans le répertoire **`WEB-INF/classes/`** de votre déploiement WAR. Si vous souhaitez mettre le fichier dans le répertoire **`lib/`** de votre WAR, vous devrez spécifier le chemin **`<resource-root>`** dans le fichier **`jboss-deployment-structure.xml`**.

```
<jboss-deployment-structure>
    <deployment>
        <!-- Exclusions allow you to prevent the server from
automatically adding some dependencies -->
        <exclusions>
            <module name="org.apache.log4j" />
```

```
        </exclusions>
        <resources>
            <resource-root path="lib" />
        </resources>
    </deployment>
</jboss-deployment-structure>
```

4. Démarrer le serveur JBoss EAP 6 avec l'argument de runtime suivant pour éviter de voir apparaître une exception **`ClassCastException`** de la console quand vous déployez une application :

```
-Dorg.jboss.as.logging.per-deployment=false
```

5. Déployer votre application.

Procédure 3.6. Configurer les dépendances de journalisation de JBoss EAP 6.3 ou version supérieure

Dans JBoss EAP 6.3 et versions supérieures, vous pouvez utiliser le nouvel attribut système de journalisation **`add-logging-api-dependencies`** pour exclure des dépendances de framework de journalisation. Les étapes suivantes montrent comment modifier cet attribut de journalisation sur un serveur autonome JBoss EAP.

1. Démarrer le serveur JBoss EAP 6 avec l'argument de runtime suivant pour éviter de voir apparaître une exception **`ClassCastException`** de la console quand vous déployez une application :

```
-Dorg.jboss.as.logging.per-deployment=false
```

2. Ouvrir un terminal et se connecter au Management CLI.
   - Dans Linux, saisir ce qui suit au niveau de la ligne de commande :

```
$ EAP_HOME/bin/jboss-cli.sh --connect
```

   - Dans Windows, saisir ce qui suit au niveau de la ligne de commande :

```
C:\>EAP_HOME\bin\jboss-cli.bat --connect
```

3. Modifier l'attribut **`add-logging-api-dependencies`** du sous-système de journalisation.

   Cet attribut contrôle si le conteneur ajoute des dépendances d'API de journalisation implicites à votre déploiement.

   - Si défini à **`true`**, qui est la valeur par défaut, toutes les dépendances d'API de journalisation implicites seront ajoutées.
   - Si défini à **`false`**, les dépendances ne seront pas ajoutées à vos déploiements.

   Pour exclure les dépendances de framework de journalisation, vous devez définir cet attribut à **`false`** à l'aide de la commande suivante :

```
/subsystem=logging:write-attribute(name=add-logging-api-
dependencies, value=false)
```

-

Cette commande ajoute l'élément **<add-logging-api-dependencies>** au sous-système **logging** du fichier de configuration **standalone.xml**.

```
<subsystem xmlns="urn:jboss:domain:logging:1.4">
    <add-logging-api-dependencies value="false"/>
    ....
</subsystem>
```

4. Déployer votre application.

Rapporter un bogue

3.1.4.3. Modifier le code pour utiliser le nouveau framework de JBoss Logging

Résumé

Pour utiliser le nouveau framework, vous devrez modifier vos importations et votre code comme suit :

Procédure 3.7. Modifier le code et les dépendances pour utiliser le nouveau framework de JBoss Logging

1. **Modifier vos importations et votre code de logging**
   Voici un exemple de code qui utilise le framework de JBoss Logging :

```
import org.jboss.logging.Level;
import org.jboss.logging.Logger;

private static final Logger logger =
Logger.getLogger(MyClass.class.toString());

if(logger.isTraceEnabled()) {
    logger.tracef("Starting...", subsystem);
}
```

2. **Ajouter la dépendance de logging**
   Le JAR qui contient les classes de JBoss Logging se trouve dans le module intitulé **org.jboss.logging**. Votre fichier **MANIFEST-MF** devrait ressembler à ceci :

```
Manifest-Version: 1.0
Dependencies: org.jboss.logging
```

   Pour plus d'informations sur la façon de trouver la dépendance de module, consulter Section 3.1.2.3, « Mise à jour des dépendances d'applications en raison des changements de chargement de classes » et Section 4.2.1, « Déboguer et Résoudre les problèmes de migration ».

Rapporter un bogue

3.1.5. Changements au niveau packaging d'applications

3.1.5.1. Modifier l'empaquetage des EAR et des WAR

Résumé

Quand vous migrez votre application, vous devez modifier la structure d'empaquetage de votre EAR ou WAR en raison du changement au niveau du chargement de classe modulaire. Les dépendances de modules sont chargées dans l'ordre spécifique suivant :

1. Dépendances Système
2. Dépendances Utilisateur
3. Ressources locales
4. Dépendances d'inter-déploiement

Procédure 3.8. Modifier l'empaquetage des archives

1. **Empaqueter un WAR**
   Un WAR est un module simple et toutes les classes du WAR sont chargées par le même chargeur de classes. Cela signifie que les classes packagées dans le répertoire **`WEB-INF/lib/`** sont traitées de la même façon que les classes qui se trouvent dans le répertoire **`WEB-INF/classes`**.

2. **Empaqueter un EAR**
   Un EAR se compose de plusieurs modules. Le répertoire **`EAR/lib/`** est un module unique et chaque sous-déploiement de jar WAR ou EJB du EAR est un module séparé. Les classes n'ont pas accès aux classes dans d'autres modules du EAR, à moins que des dépendances explicites n'aient été définies. Les sous-déploiements ont toujours une dépendance automatique sur le module parent qui leur donne accès aux classes dans le répertoire **`EAR/lib/`**. Toutefois, les sous-déploiements n'ont pas toujours une dépendance automatique pour leur permettre de passer de l'un à l'autre. Ce comportement est contrôlé en définissant l'élément **`<ear-subdeployments-isolated>`** dans la configuration du sous-système **`ee`** comme suit :

   ```
   <subsystem xmlns="urn:jboss:domain:ee:1.0" >
     <ear-subdeployments-isolated>false</ear-subdeployments-isolated>
   </subsystem>
   ```

   Défini par défaut à false, ce qui permet aux sous-déploiements de voir les classes qui appartiennent aux autres sous-déploiements du EAR.

   Pour obtenir des informations supplémentaires sur le chargement de classes, voir le chapitre *Class Loading and Modules*(Chargement de classes et modules) du *Development Guide* (Guide de développement) de JBoss EAP 6 https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/.

Rapporter un bogue

3.1.6. Changements au niveau de la configuration d'adaptateur de ressources et de source de données

3.1.6.1. Mise à jour de l'application en raison des changements de configuration

Dans JBoss EAP 5, les sous-systèmes et les services ont été configurés dans plusieurs fichiers différents. Dans JBoss EAP 6, la configuration est maintenant effectuée principalement dans un seul fichier. Si votre application utilise les ressources ou les services suivants, des modifications de configuration peuvent être nécessaires.

1. Si votre application utilise une source de données, voir Section 3.1.6.2, « Mise à jour de la Configuration de la Source de données ».

2. Si votre application utilise JPA et regroupe actuellement les JARS Hibernate, voir Section 3.1.6.4, « Configurer la source de données d'Hibernate ou JPA » pour vos options de migration.

3. Si votre application utilise un adaptateur de ressources, voir Section 3.1.6.5, « Mise à jour de la Configuration de l'adaptateur de ressources ».

4. Pour obtenir plus d'informations sur la façon de configurer les changements pour augmenter le niveau de sécurité: Section 3.1.7.1, « Configurer les changements de sécurité d'applications » .

Rapporter un bogue

3.1.6.2. Mise à jour de la Configuration de la Source de données

Résumé

Dans les versions précédentes de JBoss EAP, la configuration de source de données JCA était définie dans un fichier avec un suffixe `*-ds.xml`. Ce fichier était ensuite déployé sur le répertoire du serveur `deployer/` ou fourni avec l'application. Le pilote JDBC était copié sur le répertoire du serveur `server/lib/` ou fourni dans le répertoire `WEB-INF/lib/` de l'application. Malgré que cette méthode de configuration de source de données soit toujours prise en charge pour le développement, il n'est pas recommandé pour la production parce qu'il n'est pas soutenu par les outils de gestion et administratifs de JBoss.

Dans JBoss EAP 6, la source de données est configurée dans le fichier de configuration du serveur. Si l'instance JBoss EAP 6 exécute dans un domaine géré, la source de données est configurée dans le `domain/configuration/domain.xml` du fichier. Si l'instance de la JBoss Enterprise Application Platform exécute comme un serveur autonome, la source de données est configurée dans le fichier `standalone/configuration/standalone.xml`. Les sources de données configurées de cette façon peuvent être gérées et contrôlées par les interfaces de gestion de JBoss, y compris par la Web Management Console et une interface de ligne de commande (CLI). Ces outils permettent de gérer des déploiements et de configurer plusieurs serveurs exécutant dans un domaine géré.

La section suivante décrit comment modifier votre configuration de source de données pour qu'elle puisse être gérée et supportée par les outils de gestion disponibles.

Migration vers une configuration de sources de données gérable pour JBoss EAP 6

Un pilote compatible JDBC 4.0 peut être installé sous forme de déploiement ou sous forme d'un module de base. Un pilote compatible JDBC 4.0 contient un fichier `META-INF/services/java.sql.Driver` qui indique le nom de classe du pilote. Un pilote qui n'est pas compatible JDBC 4.0 requiert des étapes supplémentaires. Pour plus d'informations sur la façon de rendre un pilote JDBC 4.0 compatible et comment mettre à jour votre configuration de source de données actuelle pour qu'elle soit gérable par la Web Management Console et la CLI, voir Section 3.1.6.3, « Installer et Configurer le Pilote JDBC ».

Si votre application utilise Hibernate ou JPA, elle a sans doute besoin de changements supplémentaires. Voir Section 3.1.6.4, « Configurer la source de données d'Hibernate ou JPA » pour plus d'informations.

Utiliser l'outil de migration IronJacamar pour convertir des données de configuration

Vous pouvez utiliser IronJacamar pour migrer les configurations de l'adaptateur de ressources et de source de données. Cet outil convertit les fichiers de configuration de style `*-ds.xml` en formats

familiers à JBoss EAP 6. Pour plus d'informations, consulter : Section 4.1.6, « Utilisation de l'outil IronJacamar pour migrer les Configurations d'aptateurs de ressources et de sources de données ».

Rapporter un bogue

3.1.6.3. Installer et Configurer le Pilote JDBC

Résumé

Le pilote JDBC peut être installé dans le conteneur d'une des manières suivantes :

- Sous forme de déploiement
- Sous forme de module core

Les avantages et les inconvénients de chaque approche sont notés ci-dessous.

Dans JBoss EAP 6, la source de données est configurée dans le fichier de configuration du serveur. Si l'instance JBoss EAP 6 exécute dans un domaine géré, la source de données est configurée dans le **`domain/configuration/domain.xml`** du fichier. Si l'instance JBoss EAP 6 exécute comme un serveur autonome, la source de données est configurée dans le fichier **`standalone/configuration/standalone.xml`**. Les informations de référence schéma, qui sont identiques pour les deux modes, se trouvent dans le répertoire **`doc/`** de JBoss EAP 6. Dans le but de cette discussion, assumez que le serveur exécute de façon autonome et que la source de données est configurée dans le fichier **`standalone.xml`**.

Procédure 3.9. Installer et Configurer le Pilote JDBC

1. **Installer le pilote JDBC.**

   a. **Installer le Pilote JDBC sous forme de déploiement.**
   C'est la méthode recommandée pour installer le pilote. Lorsque le pilote JDBC est installé comme un déploiement, il est déployé en tant que JAR ordinaire. Si l'instance de JBoss EAP 6 exécute en serveur autonome, copiez le JAR compatible JDBC 4.0 dans le répertoire ***`EAP_HOME`*`/standalone/deployments/`**. Si le serveur exécute en domaine géré, vous devez utiliser la Console de gestion ou le Management CLI pour déployer le JAR dans les groupes de serveurs.

   Voici un exemple de pilote MySQL JDBC installé sous forme de déploiement de serveur autonome :

   ```
   $cp mysql-connector-java-5.1.15.jar
   EAP_HOME/standalone/deployments/
   ```

   Tout pilote conforme JDBC 4.0 est automatiquement reconnu et installé dans le système avec son nom et sa version. Un JAR conforme JDBC 4.0 contient un fichier-texte nommé **`META-INF/services/java.sql.Driver`** qui indique les nom(s) de classe de pilote. Si le pilote n'est pas conforme JDBC 4.0, il peut être rendu déployable par l'un des moyens suivants :

   - Créer et ajouter un fichier **`java.sql.Driver`** au JAR dans le chemin **`META-INF/services/`**. Ce fichier doit contenir le nom de classe du pilote, par exemple :

     ```
     com.mysql.jdbc.Driver
     ```

- Créer un fichier **`java.sql.Driver`** dans le répertoire de déploiement. Pour une instance de JBoss EAP 6 exécutant en serveur autonome, le fichier devra aller à l'emplacement suivant : ***`EAP_HOME`*`/standalone/deployments/META-INF/services/java.sql.Driver`**. Si le serveur est en domaine géré, vous devrez utiliser la Console de gestion ou le Management CLI pour déployer le fichier.

Les avantages de cette approche :

- Il s'agit de la méthode la plus aisée car il n'y a nul besoin de définir un module.
- Quand le serveur exécute dans un domaine géré, les déploiements qui utilisent cette approche sont automatiquement propagés dans tous les serveurs du domaine. Cela signifie que l'administrateur n'a nul besoin de distribuer le JAR Pilote manuellement.

Les inconvénients de cette approche :

- Si le pilote JDBC est constitué de plus d'un JAR, par exemple le JAR Pilote, plus un JAR Licence dépendant ou un JAR Localisation, vous ne pouvez pas installer le pilote comme déploiement. Vous devrez alors installer le pilote JDBC comme un module de base.
- Si le pilote n'est pas conforme à JDBC 4.0, un fichier doit être créé, contenant les noms de classe de pilote et doit être importé dans le JAR ou bien être superposé dans le répertoire **`deployments/`**.

b. **Installer un Pilote JDBC comme Core Module.**
Pour installer un pilote JDBC comme module principal, vous devez installer une structure de chemin de fichier sous le répertoire ***`EAP_HOME`*`/modules/`**. Cette structure contiendra le JAR Pilote JDBC, tout JAR de localisation ou licence supplémentaire, et un fichier **`module.xml`** pour définir le module.

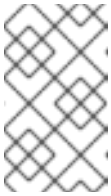
- **Installer un Pilote MySQL JDBC comme Core Module**

  i. Créer la structure de répertoire de ***`EAP_HOME`*`/modules/com/mysql/main/`**

  ii. Dans le sous-répertoire **`main/`**, créer un fichier **`module.xml`** qui contient la définition de module suivante pour le pilote MySQL JDBC :

```
<?xml version="1.0" encoding="UTF-8"?>
<module xmlns="urn:jboss:module:1.0" name="com.mysql">
<resources>
    <resource-root path="mysql-connector-java-5.1.15.jar"/>
</resources>
<dependencies>
    <module name="javax.api"/>
</dependencies>
</module>
```

Le nom du module "com.mysql" doit correspondre à la structure du répertoire du module. L'élément **`<dependencies>`** est utilisé pour spécifier les dépendances de ce module sur les autres modules. Dans un tel cas, comme pour tous les cas comprenant des sources de données JDCB, il sera dépendant des API JDBC qui sont définis dans un autre module nommé **`javax.api`**. Ce module se trouve sous le répertoire **`modules/system/layers/base/javax/api/main/`**.



NOTE

Veillez à ne PAS laisser d'espace au début du fichier **module.xml** sinon, vous aurez l'erreur "New missing/unsatisfied dependencies" pour ce pilote.

iii. Copier le pilote JAR MySQL JDBC dans le répertoire ***EAP_HOME*/modules/com/mysql/main/** :

```
$ cp mysql-connector-java-5.1.15.jar
EAP_HOME/modules/com/mysql/main/
```

- **Installer le pilote IBM DB2 JDBC et le JAR Licence en tant que module principal.** Cet exemple est là pour vous démontrer comment déployer les pilotes qui requièrent des JAR en plus du JAR de pilote JDBC.

  i. Créer la structure de répertoire de ***EAP_HOME*/modules/com/ibm/db2/main/**

  ii. Dans le sous-répertoire **main/**, créer un fichier **module.xml** qui contient la définition de module suivante pour le pilote et la licence DB2 JDBC :

```
<?xml version="1.0" encoding="UTF-8"?>
<module xmlns="urn:jboss:module:1.1" name="com.ibm.db2">
  <resources>
    <resource-root path="db2jcc.jar"/>
    <resource-root path="db2jcc_license_cisuz.jar"/>
  </resources>
  <dependencies>
    <module name="javax.api"/>
    <module name="javax.transaction.api"/>
  </dependencies>
</module>
```

NOTE

Veillez à ne PAS laisser d'espace au début du fichier **module.xml** sinon, vous aurez l'erreur "New missing/unsatisfied dependencies" pour ce pilote.

  iii. Copier le pilote JDBC et le JAR de licence dans le sous-répertoire ***EAP_HOME*/modules/com/ibm/db2/main/**

```
$ cp db2jcc.jar EAP_HOME/modules/com/ibm/db2/main/
$ cp db2jcc_license_cisuz.jar
EAP_HOME/modules/com/ibm/db2/main/
```

Les avantages de cette approche :

- Il s'agit de la seule approche qui fonctionne quand le pilote JDBC consiste en un JAR au moins.
- Avec cette approche, les pilotes qui ne sont pas conformes à JDBC 4.0 peuvent être installés sans modifier le JAR Pilote ou créer un fichier superposé.

Les inconvénients de cette approche :

- Il est plus difficile de définir un module.
- Le module doit être copié manuellement dans chaque serveur exécutant dans un domaine géré.

2. **Configurer la source de données.**

   a. **Ajouter le pilote de base de données.**
   Ajouter l'élément **`<driver>`** à l'élément **`<drivers>`** du même fichier. Là aussi, il contiendra les mêmes informations de source de données que celles préalablement définies dans le fichier **`*-ds.xml`**.

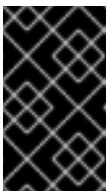
   Déterminer tout d'abord si le pilote JAR est conforme JDBC 4.0. Un JAR qui est conforme JDBC 4.0 contient un fichier **`META-INF/services/java.sql.Driver`** qui indique le nom de la classe du pilote. Le serveur utilise ce fichier pour trouver le nom des classe(s) de pilote du JAR. Un pilote qui est conforme à JDBC 4.0 n'a pas besoin d'élément **`<driver-class>`** puisqu'il est déjà spécifié dans le JAR. Voici un exemple d'élément de pilote pour un pilote MySQL conforme JDBC 4.0 :

```
<driver name="mysql-connector-java-5.1.15.jar"
module="com.mysql"/>
```

   Un pilote qui est conforme JDBC 4.0 a besoin d'un attribut **`<driver-class>`** pour identifier la classe du pilote puisqu'il n'y a pas de fichier **`META-INF/services/java.sql.Driver`** qui spécifie le nom de classe du pilote. Il s'agit d'un exemple d'élément de pilote non conforme à JDBC 4.0 :

```
<driver name="mysql-connector-java-5.1.15.jar"
module="com.mysql">
<driver-class>com.mysql.jdbc.Driver</driver-class></driver>
```

   b. **Créer la source de données.**
   Créer un élément **`<datasource>`** dans la section **`<datasources>`** du fichier **`standalone.xml`**. Ce fichier contient plus ou moins les mêmes informations de source de données que celles préalablement définies dans le fichier **`*-ds.xml`**.



IMPORTANT

Vous devez interrompre le serveur avant de modifier le fichier de configuration du serveur pour que votre changement puisse être persisté au redémarrage du serveur.

Ce qui suit est un exemple d'élément de source de données MySQL du fichier **`standalone.xml`** :

```
<datasource jndi-name="java:/YourDatasourceName" pool-
name="YourDatasourceName">
  <connection-
url>jdbc:mysql://localhost:3306/YourApplicationURL</connection-
url>
  <driver>mysql-connector-java-5.1.15.jar</driver>
  <transaction-isolation>TRANSACTION_READ_COMMITTED</transaction-
```

```
isolation>
  <pool>
    <min-pool-size>100</min-pool-size>
    <max-pool-size>200</max-pool-size>
  </pool>
  <security>
    <user-name>USERID</user-name>
    <password>PASSWORD</password>
  </security>
  <statement>
    <prepared-statement-cache-size>100</prepared-statement-cache-
size>
    <share-prepared-statements/>
  </statement>
</datasource>
```

3. **Mettre à jour les références JNDI dans le code d'application.**
   Vous devez remplacer les noms de recherche JNDI obsolètes dans le code source d'application pour utiliser les nouveaux noms de source de données standardisés que vous aurez définis. Pour plus d'informations, consulter : Section 3.1.8.4, « Modifier l'application pour qu'elle puisse suivre les nouvelles règles d'espace noms JNDI ».

   Vous devrez également remplacer toute annotation **@Resource** existante qui accède à la source de données avec le nouveau nom JNDI. Par exemple :

   ```
   @Resource(name = "java:/YourDatasourceName").
   ```

Rapporter un bogue

3.1.6.4. Configurer la source de données d'Hibernate ou JPA

Si votre application utilise JPA et regroupe actuellement les JAR d'Hibernate, vous pouvez utiliser l'Hibernate qui est inclus dans JBoss EAP 6. Pour utiliser cette version d'Hibernate, vous devez supprimer l'ancien Hibernate de votre application.

Procédure 3.10. Supprimer le lot Hibernate

1. Supprimer les JAR Hibernate de vos dossiers de bibliothèque d'applications.

2. Supprimer et décommenter l'élément **<hibernate.transaction.manager_lookup_class>** de votre fichier **persistence.xml** car cet élément n'est pas utile.

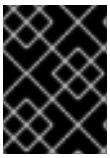
Rapporter un bogue

3.1.6.5. Mise à jour de la Configuration de l'adaptateur de ressources

Résumé

Dans les dernières versions du serveur d'applications, la configuration de l'adaptateur de ressources était définie dans un fichier ayant pour suffixe ***-ds.xml**. Dans JBoss EAP 6, un adaptateur de ressources est configuré dans le fichier de configuration du serveur. Si vous exécutez un domaine géré, le fichier de configuration est le fichier ***EAP_HOME*/domain/configuration/domain.xml**. Si vous exécutez en serveur autonome, configurez l'adaptateur de ressources dans le fichier

***EAP_HOME*/standalone/configuration/standalone.xml**. Vous pourrez trouver les informations de référence schéma, identiques pour les deux modes, sous *Schemas* dans le site web IronJacamar à l'adresse suivante : http://www.ironjacamar.org/documentation.html.



IMPORTANT

Vous devez interrompre le serveur avant de modifier le fichier de configuration du serveur pour que votre changement puisse être persisté au redémarrage du serveur.

Définir l'adaptateur de ressources

Les informations sur le descripteur d'adaptateur de ressources se trouvent dans le fichier de configuration du serveur, sous l'élément de sous-système suivant :

```
<subsystem xmlns="urn:jboss:domain:resource-adapters:1.1"/>
```

Vous allez utiliser certaines des informations qui étaient auparavant définies dans le fichier d'adaptateur de ressources ***-ds.xml**.

Ce qui suit est un exemple d'élément d'adaptateur de ressource du fichier de configuration du serveur :

```
<resource-adapters>
  <resource-adapter>
    <archive>multiple-full.rar</archive>
    <config-property name="Name">ResourceAdapterValue</config-property>
    <transaction-support>NoTransaction</transaction-support>
    <connection-definitions>
      <connection-definition
      class-
name="org.jboss.jca.test.deployers.spec.rars.multiple.MultipleManagedConne
ctionFactory1"
      enabled="true" jndi-name="java:/eis/MultipleConnectionFactory1"
      pool-name="MultipleConnectionFactory1">
    <config-property name="Name">MultipleConnectionFactory1Value</config-
property>
      </connection-definition>
      <connection-definition
      class-
name="org.jboss.jca.test.deployers.spec.rars.multiple.MultipleManagedConne
ctionFactory2"
      enabled="true" jndi-name="java:/eis/MultipleConnectionFactory2"
      pool-name="MultipleConnectionFactory2">
    <config-property name="Name">MultipleConnectionFactory2Value</config-
property>
      </connection-definition>
    </connection-definitions>
    <admin-objects>
      <admin-object
      class-
name="org.jboss.jca.test.deployers.spec.rars.multiple.MultipleAdminObject1
Impl"
      jndi-name="java:/eis/MultipleAdminObject1">
    <config-property name="Name">MultipleAdminObject1Value</config-
property>
      </admin-object>
```

```
      <admin-object class-
name="org.jboss.jca.test.deployers.spec.rars.multiple.MultipleAdminObject2
Impl"
      jndi-name="java:/eis/MultipleAdminObject2">
    <config-property name="Name">MultipleAdminObject2Value</config-
property>
      </admin-object>
      </admin-objects>
  </resource-adapter>
</resource-adapters>
```

Rapporter un bogue

3.1.7. Changements au niveau sécurité

3.1.7.1. Configurer les changements de sécurité d'applications

Configurer la sécurité pour l'authentification de base

Dans les versions précédentes de JBoss EAP, les fichiers de propriétés qui se trouvent dans le répertoire ***EAP_HOME*/server/*SERVER_NAME*/conf/** étaient sur un chemin de classe et on pouvait les trouver facilement avec **UsersRolesLoginModule**. Dans JBoss EAP 6, la structure du répertoire a changé. Les fichiers de propriétés doivent être empaquetés dans l'application pour les rendre disponibles par le chemin de classe.

IMPORTANT

Vous devez interrompre le serveur avant de modifier le fichier de configuration du serveur pour que votre changement puisse être persisté au redémarrage du serveur.

Pour configurer la sécurité de l'authentification de base, ajouter un nouveau domaine de sécurité sous **security-domains** au **standalone/configuration/standalone.xml** ou au fichier de configuration du serveur **domain/configuration/domain.xml**:

```
<security-domain name="example">
    <authentication>
        <login-module code="UsersRoles" flag="required">
            <module-option name="usersProperties"
                    value="${jboss.server.config.dir}/example-
users.properties"/>
            <module-option name="rolesProperties"
                    value="${jboss.server.config.dir}/example-
roles.properties"/>
        </login-module>
    </authentication>
</security-domain>
```

Si l'instance de JBoss EAP 6 exécute en tant que serveur autonome, **${jboss.server.config.dir}** fait référence au répertoire ***EAP_HOME*/standalone/configuration/**. Si l'instance exécute en domaine géré, **${jboss.server.config.dir}** fait référence au répertoire ***EAP_HOME*/domain/configuration/**.

Modifier les noms de domaine de sécurité

Dans JBoss EAP 6, les domaines de sécurité n'utilisent plus le préfixe **`java:/jaas/`** dans leurs noms.

- Pour les applications web, vous devez supprimer ce préfixe des configurations du domaine de sécurité dans le fichier **`jboss-web.xml`**.
- Dans les applications Enterprise, vous devez supprimer ce préfixe des configurations du domaine de sécurité dans le fichier **`jboss-ejb3.xml`**. Ce fichier remplace le fichier **`jboss.xml`** de JBoss EAP 6.

Rapporter un bogue

3.1.7.2. Mettre à jour les applications utilisant PicketLink STS et les Services Web

Résumé

Si votre applicationJBoss EAP 6.1 utilise PicketLink STS et les services Web, vous devrez sans doute effectuer des changements quand vous migrerez vers JBoss EAP 6.2 ou version supérieure. Un correctif s'appliquant à JBoss EAP pour solutionner CVE-2013-2133 oblige le conteneur à vérifier les autorisations avant d'exécuter un gestionnnaire JAXWS qui serait attaché aux points de terminaison WS basés-EJB3. De ce fait, certaines fonctionnalités PicketLink STS peuvent être affectées car le **`SAML2Handler`** de PicketLink établit un principal de sécurité conçu pour rêtre utilisé plus tard dans le processus. Vous apercevrez sans doute une exception **`NullPointerException`** dans le journal du serveur parce que le principal est **`NULL`** quand le **`HandlerAuthInterceptor`** accède au **`SAML2Handler`**. Vous devrez déssactiver ce contrôle de sécurité pour régler ce problème.

Procédure 3.11. Désactiver les contrôles d'autorisation supplémentaires

- Vous pouvez désactiver le contrôles d'autorisation supplémentaires et continuer d'utiliser les déploiements PicketLink existants par l'une des méthodes suivantes.
  - **Définir une propriété système**
    Vous pouvez désactiver des contrôles d'autorisation supplémentaires au niveau serveur en définissant la valeur de la propriété système **`org.jboss.ws.cxf.disableHandlerAuthChecks`** à **`true`**. Cette méthode affecte tout développement effectué dans le serveur d'applications.

    Pour obtenir des informations sur la façon de définir une propriété système, voir la section *Configure System Properties Using the Management CLI*du guide *Administration and Configuration Guide* de JBoss EAP.
  - **Créer une propriété dans le fichier de descripteur de services Web du déploiement**
    Vous pouvez désactiver des contrôles d'authorisation supplémentaires au niveau du déploiement en définissant la valeur de la propriété système **`org.jboss.ws.cxf.disableHandlerAuthChecks`** à **`true`** dans le fichier **`jboss-webservices.xml`**. Cette méthode n'affectera que le déploiement dont il s'agit.

    a. Créer un fichier **`jboss-webservices.xml`** dans le répertoire **`META-INF`** du déploiement dans lequel vous souhaitez désactiver les contrôles d'authorisation supplémentaires.

    b. Ajouter le contenu suivant :

```
<?xml version="1.1" encoding="UTF-8"?>
<webservices xmlns="http://www.jboss.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  version="1.2"
```

```
xsi:schemaLocation="http://www.jboss.com/xml/ns/javaee">
  <property>
    <name>org.jboss.ws.cxf.disableHandlerAuthChecks</name>
    <value>true</value>
  </property>
</webservices>
```

NOTE

En activant la propriété **`org.jboss.ws.cxf.disableHandlerAuthChecks`**, on rend le système vulnérable à CVE-2013-2133. Si l'application s'attend à des restrictions de sécurité déclarées sur les méthodes EJB à appliquer et ne les applique pas indépendemment du gestionnaire JAX-WS, alors la propriété ne doit pas être activée. La propriété ne doit être utilisée qu'à but de rétro-compatibilité quand c'est utile afin d'éviter d'interrompre l'application.

Rapporter un bogue

3.1.8. Changements JNDI

3.1.8.1. Mise à jour des noms d'espace-noms JNDI d'application

Résumé

EJB 3.1 introduit un espace-noms JNDI global standardisé et une série d'espace-noms connexes qui mappent vers les différents scopes d'une application Java EE. Les noms EJB portables ne peuvent se rattacher qu'à trois d'entre eux : **`java:global`**, **`java:module`**, et **`java:app`**. Les applications avec EJBs qui utilisent JNDI doivent être mises à jour à la nouvelle convention d'espace-noms JNDI standardisée.

Procédure 3.12. Modifier les recherches JNDI

1. Pour en savoir plus Section 3.1.8.2, « Noms JNDI EJB portables »
2. Section 3.1.8.3, « Revue des règles d'espace-noms JNDI »
3. Section 3.1.8.4, « Modifier l'application pour qu'elle puisse suivre les nouvelles règles d'espace noms JNDI »

Exemple de mappings JNDI

Des exemples d'espace-noms JNDI de versions précédentes et des informations sur la façon dont ils peuvent être spécifiés dans JBoss EAP 6 peuvent être consultés ici : Section 3.1.8.5, « Exemples d'espace noms JNDI de versions antérieures et la façon dont ils sont spécifiés dans JBoss EAP 6 »

Rapporter un bogue

3.1.8.2. Noms JNDI EJB portables

Résumé

La spécification Java EE6 définit quatre espace-noms logiques, ayant chacun son propre scope mais les noms EJB ne sont rattachés qu'à trois d'entre eux. La table suivante explique quand et comment utiliser chaque espace-nom.

Tableau 3.1. Espace-nom JNDI portable

Espace-nom JNDI	Description
java:global	Les noms sont partagés par toutes les applications déployées dans une instance de serveur d'applications. Utiliser les noms de cet espace-nom pour rechercher des EJBs d'archives externes déployées dans le même serveur. Ce qui suit est un exemple d'espace-nom java:global namespace: **`java:global/jboss-seam-booking/jboss-seam-booking-jar/HotelBookingAction`**
java:module	Dans cet espace-nom, les noms sont partagés par tous les composants d'un module, c'est à dire et par exemple, tous les beans Enterprise d'un module EJB unique ou tous les composants d'un module web. Ce qui suit est un exemple d'espace-nom java:global : **`java:module/HotelBookingAction!org.jboss.seam.example.booking.HotelBooking`**
java:app	Les noms sont partagés par tous les composants dans tous les modules d'une simple application. Ainsi, un WAR ou fichier jar EJB du même fichier EAR aura accès à toutes les ressources dans l'espace-nom java:app. Ce qui suit est un exemple d'espace-nom java:app : **`java:app/jboss-seam-booking-jar/HotelBookingAction`**

Vous trouverez des informations supplémentaires sur les contextes de nommage JNDI dans la section EE.5.2.2, "Application Component Environment Namespaces" dans "JSR 316: JavaTM Platform, Enterprise Edition (Java EE) Specification, v6". Vous pouvez télécharger la spécification à partir de : http://jcp.org/en/jsr/detail?id=316

Rapporter un bogue

3.1.8.3. Revue des règles d'espace-noms JNDI

Résumé

JBoss EAP 6 s'est améliorée au niveau des noms d'espace-noms JNDI, non seulement afin de fournir des règles prévisibles et cohérentes pour chaque nom lié dans le serveur d'applications, mais aussi pour éviter des problèmes de compatibilité à venir. Cela signifie que vous pouvez vous heurter à des problèmes d'espaces-noms dans votre application, s'ils ne suivent pas les nouvelles règles.

Les espace-noms doivent suivre les règles suivantes :

1. Les noms relatifs non qualifiés tels que **`DefaultDS`** ou **`jdbc/DefaultDS`** doivent être qualifiés par rapport à **`java:comp/env`**, **`java:module/env`**, ou **`java:jboss/env`**, suivant le contexte.

2. Les noms non qualifiés tels que les noms **`absolute`** comme **`/jdbc/DefaultDS`** doivent être qualifiés par rapport à un nom **`java:jboss/root`**.

3. Les noms qualifiés **`absolute`** tels que **`java:/jdbc/DefaultDS`** doivent être qualifiés de la même façon que les noms **`absolute`** non qualifiés ci-dessus.

4. L'espace-nom **`java:jboss`** en particulier est partagé par toute l'instance de serveur AS.

5. Tout nom **`relative`** ayant pour préfixe **`java:`** doit correspondre à l'un des cinq espace-noms : **`comp`**, **`module`**, **`app`**, **`global`**, ou **`jboss`** propriétaire. Tout nom commençant par **`java:xxx`** et dont *xxx* ne correspond pas à l'un des cinq espace-noms ci-dessous, résulterait en une erreur de nom non valide.

Rapporter un bogue

3.1.8.4. Modifier l'application pour qu'elle puisse suivre les nouvelles règles d'espace noms JNDI

- Voici un exemple de recherche JNDI dans JBoss EAP 5.1. On trouve normalement ce code dans la méthode d'initialisation.

```
private ProductManager productManager;
try {
    context = new InitialContext();
    productManager = (ProductManager)
context.lookup("OrderManagerApp/ProductManagerBean/local");
} catch(Exception lookupError) {
    throw new ServletException("Unable to find the ProductManager
bean", lookupError);
}
```

  Notez que le nom de recherche est **`OrderManagerApp/ProductManagerBean/local`**.

- Ce qui suit est un exemple qui montre comment une même recherche serait codifiée dans JBoss EAP 6.

```
@EJB(lookup="java:app/OrderManagerEJB/ProductManagerBean!services.ej
b.ProductManager")
private ProductManager productManager;
```

  Les valeurs de recherche sont maintenant définies comme variables de membre et utilisent le nouveau nom d'espace nom JNDI portable **`java:app`** **`java:app/OrderManagerEJB/ProductManagerBean!services.ejb.ProductManage r`**.

- Si vous préférez ne pas utiliser une injeciton de dépendance, vous pouvez toujours créer le nouvel InitialContext comme ci-dessus et modifier la recherche pour utiliser le nouvel espace nom JNDI.

```
private ProductManager productManager;
try {
    context = new InitialContext();
    productManager = (ProductManager)
context.lookup("java:app/OrderManagerEJB/ProductManagerBean!services
.ejb.ProductManager");
} catch(Exception lookupError) {
    throw new ServletException("Unable to find the ProductManager
bean", lookupError);
}
```

Rapporter un bogue

3.1.8.5. Exemples d'espace noms JNDI de versions antérieures et la façon dont ils sont spécifiés dans JBoss EAP 6

Tableau 3.2. Table de mappage d'espace noms JNDI

Espace noms dans JBoss EAP 5.x	Espace noms dans JBoss EAP 6	Commentaires supplémentaires
OrderManagerApp/ProductManagerBean/local	java:module/ProductManagerBean!services.ejb.ProductManager	Liaison standard Java EE6, étendue au module en cours et accessible uniquement dans le même module
OrderManagerApp/ProductManagerBean/local	java:app/OrderManagerEJB/ProductManagerBean!services.ejb.ProductManager	Liaison standard Java EE6, étendue à l'application en cours et accessible uniquement dans la même application.
OrderManagerApp/ProductManagerBean/local	java:global/OrderManagerApp/OrderManagerEJB/ProductManagerBean!services.ejb.ProductManager	Liaison standard Java EE6, étendue au serveur d'application en cours et accessible globalement.
java:comp/UserTransaction	java:comp/UserTransaction	Espace nom étendu au composant en cours. Non Accessible pour les threads non EE, comme des threads que votre application crée directement.
java:comp/UserTransaction	java:jboss/UserTransaction	Accessible globalement. À utiliser si java:comp/UserTransaction n'est pas disponible
java:/TransactionManager	java:jboss/TransactionManager	
java:/TransactionSynchronizationRegistry	java:jboss/TransactionSynchronizationRegistry	

Rapporter un bogue

3.2. CHANGEMENTS QUI DÉPENDENT DE VOTRE ARCHITECTURE D'APPLICATION ET DE SES COMPOSANTS

3.2.1. Vérification des changements de migration qui dépendent de l'architecture de votre application et de ses composants

Si votre application utilise les technologies ou les composants suivants, vous devrez sans doute modifier votre application lorsque vous migrerez vers JBoss EAP 6.

Hibernate et JPA

Si votre application utilise Hibernate ou JPA, votre application devra être modifiée. Voir Section 3.2.2.1, « Mise à jour d'applications qui utilisent Hibernate et/ou JPA » pour plus d'informations.

REST

Si votre application utilise JAX-RS, vous devez savoir que JBoss EAP 6 installe automatiquement RESTEasy, donc vous n'aurez plus besoin de l'installer vous-même. Pour plus d'informations, voir Section 3.2.5.1, « Configurer les changements de JAX-RS and RESTEasy »

LDAP

Le domaine de sécurité LDAP est configuré différemment dans JBoss EAP 6. Si votre application utilise LDAP, voir Section 3.2.6.1, « Configurer les changements de domaine de sécurité LDAP » pour plus d'informations.

Messagerie

JBoss Messaging n'est plus inclus dans JBoss EAP 6. Si votre application utilise JBoss Messaging comme fournisseur de messagerie, vous devrez remplacer le code de Messagerie JBoss par celui d'HornetQ. Section 3.2.7.4, « Migrer votre Application pour qu'elle utilise HornetQ comme JMS Provider » décrit ce que vous devez faire.

Clustering

La façon dont vous activez le clustering a changé dans JBoss EAP 6. Pour plus d'informations, voir Section 3.2.8.1, « Changements à votre application pour le clustering » .

Déploiement style Service

Malgré que JBoss EAP 6 n'utilise plus de descripteurs de style Service, le conteneur prend en charge ces déploiements de style Service sans changement dans la mesure du possible. Pour plus d'informations, consultez Section 3.2.9.1, « Mise à jour des Applications qui utilisent les Déploiements Style-Service »

Invocations à distance

Si votre application effectue des invocations à distance, vous pourrez toujours utiliser JNDI pour chercher un proxy pour votre bean et invoquer sur ce proxy de renvoi. Voir Section 3.2.10.1, « Migrer des Applications dépoyées dans JBoss EAP 5 qui font des invocations dans JBoss EAP 6 » pour la syntaxe requise et les changements d'espace noms.

Seam 2.2

Si votre application utilise Seam 2.2, voir Section 3.2.13.1, « Migrer les Archives Seam 2.2 dans JBoss EAP 6 » pour les changements à effectuer.

Spring

Si votre application utilise Spring, voir Section 3.2.14.1, « Migrer les Applications Spring » .

Autres changements qui pourraient avoir un impact sur votre migration

Pour obtenir des informations sur les changements de JBoss EAP 6 qui risquent d'affecter votre application, voir : Section 3.2.15.1, « Familiarisez-vous avec les autres changements qui pourraient avoir un impact sur votre migration ».

Rapporter un bogue

3.2.2. Changements Hibernate et JPA

3.2.2.1. Mise à jour d'applications qui utilisent Hibernate et/ou JPA

Résumé

Si votre application utilise Hibernate ou JPA, lire les sections suivantes et faîtes les changements nécessaires pour migrer vers JBoss EAP 6.

- Section 3.2.2.2, « Configuration des changements des applications qui utilisent Hibernate et JPA »
- Section 3.2.2.4, « Mettre votre application Hibernate 3 à jour pour utiliser Hibernate 4 »
- Section 3.2.2.9, « Mettez à jour votre application pour qu'elle puisse respecter la Specification JAP 2.0 »
- Section 3.2.2.10, « Remplacer le Cache de Second Niveau JPA/Hibernate par Infinispan »
- Section 3.2.2.12, « Migrer dans Hibernate Validator 4 »

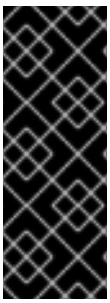
Rapporter un bogue

3.2.2.2. Configuration des changements des applications qui utilisent Hibernate et JPA

Résumé

Si votre application contient un fichier `persistence.xml` ou que le code utilise les annotations `@PersistenceContext` ou `@PersistenceUnit`, JBoss EAP 6 le détectera pendant le déploiement et assumera que l'application utilise JPA. Elle ajoutera Hibernate 4 et quelques autres dépendances à votre chemin de classe d'application.

Si votre application utilise actuellement les bibliothèques Hibernate 3, vous pourrez, la plupart du temps, passer à Hibernate 4 et exécuter. Si vous apercevez `ClassNotFoundExceptions` quand vous déployez votre application, vous pourrez essayer de la résoudre d'une des façons suivantes.



IMPORTANT

Les applications qui utilisent Hibernate directement avec Seam 2.2 utilisent sans doute une version d'Hibernate 3 empaquetée dans l'application. Hibernate 4, fourni par le module org.hibernate de JBoss EAP 6, n'est pas pris en charge par Seam 2.2. Cet exemple a pour but de vous aider à exécuter votre application dans JBoss EAP 6 comme première étape. Notez qu'empaqueter Hibernate 3 avec une application Seam 2.2 n'est pas une configuration prise en charge.

Procédure 3.13. Configurer l'application

1. **Copier les 3 JAR Hibernate dans votre bibliothèque d'applications.**
   Vous pourrez sans doute résoudre ce problème en copiant les 3 JAR Hibernate particulières qui contiennent les classes manquantes dans le répertoire de votre application `lib/` ou en les ajoutant au chemin de classe par une autre méthode. Dans certains cas, cela peut résulter en `ClassCastExceptions` ou autre problème de chargement de classe en raison de l'usage mixte de versions Hibernate. Si cela se produit, vous devrez utiliser l'approche suivante.

2. **Ordonnez au serveur de n'utiliser que les bibliothèques Hibernate 3.**
   JBoss EAP 6 vous permet d'empaqueter les jars de fournisseurs de persistance Hibernate 3.5 (ou version supérieure) dans l'application. Pour instruire le serveur à n'utiliser que les bibiothèques Hibernate 3 et pour exclure les bibliothèques Hibernate 4, vous devrez définir le **`jboss.as.jpa.providerModule`** à **`hibernate3-bundled`** dans le fichier **`persistence.xml`** comme suit :

   ```
   <?xml version="1.0" encoding="UTF-8"?>
   <persistence xmlns="http://java.sun.com/xml/ns/persistence"
   version="1.0">
       <persistence-unit name="plannerdatasource_pu">
           <description>Hibernate 3 Persistence Unit.</description>
           <jta-data-source>java:jboss/datasources/PlannerDS</jta-data-
   source>
           <properties>
               <property name="hibernate.show_sql" value="false" />
               <property name="jboss.as.jpa.providerModule"
   value="hibernate3-bundled" />
           </properties>
       </persistence-unit>
   </persistence>
   ```

   Le déployeur JPA (Java Persistence API) détectera la présence d'un fournisseur de persistences dans l'application et utilisera les bibliothèques d'Hibernate 3. Pour plus d'informations sur les propriétés de la persistence JPA, voir Section 3.2.2.3, « Propriétés d'unité de persistence ».

3. **Désactiver le cache Hibernate de second niveau**
   Le cache de second niveau d'Hibernate 3 n'a pas le même comportement avec JBoss EAP 6 que dans les versions précédentes. Si vous utilisez un cache Hibernate de second niveau avec votre application, vous devrez le désactiver jusqu'à ce que vous puissiez le mettre à niveau à Hibernate 4. Pour désactiver un cache de second niveau, définir **`<hibernate.cache.use_second_level_cache>`** à **`false`** dans le fichier **`persistence.xml`**.

Rapporter un bogue

3.2.2.3. Propriétés d'unité de persistence

Propriétés de configuration Hibernate 4.x

JBoss EAP 6 définit les propriétés de configuration Hibernate 4.x suivantes :

Tableau 3.3. Propriétés Hibernate JPA Persistence Unit

Nom de propriété	Valeur par défaut	But

Nom de propriété	Valeur par défaut	But
hibernate.id.new_generator_mappings	true	Cette configuration s'applique si vous utilisez **@GeneratedValue(AUTO)** pour générer des valeurs de clé d'indexation uniques pour les nouvelles entités. Les nouvelles applications devront garder la valeur par défaut **true**. Les applications existantes qui utilisent Hibernate 3.3.x devront sans doute modifier cette valeur à **false** pour continuer à utiliser un objet de séquence ou un générateur basé-table et pour maintenir la compatibilité rétro-active. L'application peut remplacer cette valeur dans le fichier **persistence.xml**. Des informations supplémentaires sur ce comportement sont fournies ci-dessous.
hibernate.transaction.jta.platform	Instance de l'interface **org.hibernate.service.jta.platform.spi.JtaPlatform**	Cette classe fait passer les gestionnaires de transaction, transactions utilisateur, et registres de synchronisation de transaction dans Hibernate.
hibernate.ejb.resource_scanner	Instance de l'interface **org.hibernate.ejb.packaging.Scanner**	Cette classe sait comment utiliser l'indexateur d'annotations de la plate-forme JBoss EAP pour faciliter un déploiement plus rapide.
hibernate.transaction.manager_lookup_class		Cette propriété sera supprimée si on la trouve dans persistence.xml car il y a risque de conflit avec **hibernate.transaction.jta.platform**
hibernate.session_factory_name	*QUALIFIED_PERSISTENCE_UNIT_NAME*	Défini au Nom de l'application + Nom de l'unité de persistance. L'application peut indiquer une valeur différente, mais celle-ci doit être unique pour tous les déploiements d'applications dans l'instance de JBoss EAP 6.
hibernate.session_factory_name_is_jndi	false	Défini uniquement si l'application n'indique pas de valeur pour le **hibernate.session_factory_name**.
hibernate.ejb.entitymanager_factory_name	*QUALIFIED_PERSISTENCE_UNIT_NAME*	Défini au Nom de l'application + Nom de l'unité de persistance. L'application peut indiquer une valeur différente, mais celle-ci doit être unique pour tous les déploiements d'applications dans l'instance de JBoss EAP 6.

Dans Hibernate 4.x, si **new_generator_mappings** est défini à **true**:

- **@GeneratedValue(AUTO)** correspond à **org.hibernate.id.enhanced.SequenceStyleGenerator**.

- `@GeneratedValue(TABLE)` correspond à `org.hibernate.id.enhanced.TableGenerator`.
- `@GeneratedValue(SEQUENCE)` correspond à `org.hibernate.id.enhanced.SequenceStyleGenerator`.

Dans Hibernate 4.x, si `new_generator_mappings` est défini à `false`:

- `@GeneratedValue(AUTO)` correspond à Hibernate "native".
- `@GeneratedValue(TABLE)` correspond à `org.hibernate.id.MultipleHiLoPerTableGenerator`.
- `@GeneratedValue(SEQUENCE)` correspond à Hibernate "seqhilo".

Pour obtenir plus d'informations sur ces propriétés, consulter http://www.hibernate.org/docs et consulter Hibernate 4.1 Developer Guide.

Propriétés de persistence JPA

Les propriétés suivantes sont prises en charge dans la définition de l'unité de persistence dans le fichier `persistence.xml`:

Tableau 3.4. Propriétés d'unité de persistence JPA

Nom de propriété	Valeur par défaut	But
`jboss.as.jpa.providerModule`	`org.hibernate`	Le nom du module de fournisseur de persistences La valeur doit correspondre à `hibernate3-bundled` si 3 JAR Hibernate sont dans l'archive de l'application. Si un fournisseur de persistences est empaqueté dans l'application, cette valeur doit être `application`.
`jboss.as.jpa.adapterModule`	`org.jboss.as.jpa.hibernate:4`	Le nom des classes d'intégration qui aident JBoss EAP à fonctionner avec un fournisseur de persistences. Les valeurs correctes sont les suivantes : • `org.jboss.as.jpa.hibernate:4`: pour les classes d'intégration d'Hibernate 4 • `org.jboss.as.jpa.hibernate:3`: pour les classes d'intégration d'Hibernate 3

Rapporter un bogue

3.2.2.4. Mettre votre application Hibernate 3 à jour pour utiliser Hibernate 4

Résumé

Quand vous mettez à jour votre application pour qu'elle utilise Hibernate 4, certaines mises à jour sont d'ordre général et s'appliquent quelle que soit la version d'Hibernate en cours d'utilisation par l'application. Pour les autres mises à jour, vous devez déterminer quelle version l'application utilise actuellement.

Procédure 3.14. Mettre à jour l'application pour qu'elle utilise Hibernate 4

1. Le comportement par défaut du générateur de séquence d'auto incrémentation a changé dans JBoss EAP 6. Pour davantage d'informations, consulter Section 3.2.2.5, « Préserve le comportement existant de la valeur de l'identité Hibernate auto-générée ».
2. Déterminer la version d'Hibernate actuellement utilisée par l'application et choisir la procédure de mise à jour qui convient ci-dessous.
   - Section 3.2.2.6, « Migrer votre application Hibernate 3.3.x vers Hibernate 4.x »
   - Section 3.2.2.7, « Migrer votre application Hibernate 3.5.x vers Hibernate 4.x »
3. Voir Section 3.2.2.8, « Modifier les propriétés de persistance pour les applications Seam ou Hibernate migrées qui exécutent dans un environnement clusterisé. » si vous avez l'intention d'exécuter votre application dans un environnement clusterisé.

Rapporter un bogue

3.2.2.5. Préserve le comportement existant de la valeur de l'identité Hibernate auto-générée

Hibernate 3.5 a introduit une propriété core nommée **`hibernate.id.new_generator_mappings`** qui indique comment les colonnes de séquences ou d'identités sont créees quand on utilise **`@GeneratedValue`**. Dans JBoss EAP 6, la valeur par défaut de cette propriété est définie ainsi :

- Quand vous déployez une application Hibernate native, la valeur par défaut est **`false`**.
- Quand vous déployez une application JPA, la valeur par défaut est **`true`**.

Instructions pour les nouvelles applications

Les nouvelles applications qui utilisent l'annotation **`@GeneratedValue`** doivent définir la valeur de la propriété **`hibernate.id.new_generator_mappings`** à **`true`**. Il s'agit de la configuration favorite car plus portable à travers les bases de données diverses. Dans la plupart des cas, cette configuration est plus efficace, et dans certains cas, elle résout les problèmes de compatibilité avec la spécification JPA 2.

- Pour les nouvelles applications JPA, JBoss EAP 6 définit la propriété **`hibernate.id.new_generator_mappings`** par défaut à **`true`** et cela ne doit pas être modifié.
- Pour les nouvelles applications natives Hibernate, JBoss EAP 6 définit la propriété **`hibernate.id.new_generator_mappings`** par défaut à **`false`**. Vous devrez définir cette propriété à **`true`**.

Instructions pour les applications JBoss EAP 5 existantes

Les applications existantes qui utilisent l'annotation **`@GeneratedValue`** doivent vérifier que le même générateur est utilisé pour créer les valeurs de clés primaires des nouvelles entités quand l'application est migrée dans JBoss EAP 6.

- Pour les applications JPA existantes, JBoss EAP 6 détermine la valeur par défaut de la propriété **`hibernate.id.new_generator_mappings`** à **`true`**. Vous devrez définir cette propriété à **`false`** dans le fichier **`persistence.xml`**.

- Pour les applications natives Hibernate existantes, JBoss EAP 6 définit la propriété **`hibernate.id.new_generator_mappings`** par défaut à **`false`** et vous ne devez pas la modifier.

Pour plus d'informations sur la configuration de ces propriétés, voir Section 3.2.2.3, « Propriétés d'unité de persistence ».

Rapporter un bogue

3.2.2.6. Migrer votre application Hibernate 3.3.x vers Hibernate 4.x

1. **Mapper le type `text` à `JDBC LONGVARCHAR`**
   Dans les versions d'Hibernate antérieures à 3.5, le type **`text`** était mappé à **`JDBC CLOB`**. Un nouveau type Hibernate, **`materialized_clob`**, a été ajouté à Hibernate 4 pour mapper les propriétés **`String`** Java à **`JDBC CLOB`**. Si votre application a des propriétés configurées **`type="text"`** qui devraient être mappées à **`JDBC CLOB`**, vous devrez faire une des choses suivantes :

   a. Si votre application utilise les fichiers de mappage hbm, modifier la propriété à **`type="materialized_clob"`**.

   b. Si votre application utilise des annotations, vous devrez remplacer **`@Type(type = "text")`** par **`@Lob`**.

2. **Vérifier le code pour trouver les changements de types de valeurs retournées**
   Les projections de critères agrégés numériques renvoient maintenant le même type de valeur que leurs équivalents HQL. De ce fait, les types de renvois des projections suivantes de **`org.hibernate.criterion`** ont changé.

   a. En raison de changements dans **`CountProjection`**, **`Projections.rowCount()`**, **`Projections.count(propertyName)`**, et **`Projections.countDistinct(propertyName)`**, les projections de **`count`** et **`count distinct`** renvoient maintenant une valeur **`Long`**.

   b. En raison de changement dans **`Projections.sum(propertyName)`**, les projections de **`sum`** renvoient maintenant un type de valeur qui dépend du type de propriété.

   NOTE

   Un manquement à modifier votre code d'application pourrait résulter en une exception **`java.lang.ClassCastException`**.

      i. Pour les propriétés mappées de type Long, Short, Integer, ou Integer primitifs, une valeur Long est retournée.

      ii. Pour les propriétés mappées de type Float, Double, ou Floating point primitive, une valeur Double est retournée.

Rapporter un bogue

3.2.2.7. Migrer votre application Hibernate 3.5.x vers Hibernate 4.x

1. Merger AnnotationConfiguration dans la Configuration.

Malgré que **AnnotationConfiguration** est maintenant déprécié, il ne doit pas compromettre votre migration.

Si vous utilisez encore un fichier **hbm.xml**, vous devez savoir que la plateforme JBoss EAP 6 utilise maintenant la stratégie de nommage **org.hibernate.cfg.EJB3NamingStrategy** dans **AnnotationConfiguration** à la place de **org.hibernate.cfg.DefaultNamingStrategy** utilisé dans les versions précédentes. Cela peut résulter par des erreurs de nommage. Si vous vous fiez à la stratégie de nommage pour donner un nom de tableau d'association (many-to-many et ensemble d'éléments) par défaut, vous rencontrerez sans doute ce problème. Pour résoudre ceci, vous devez ordonner à Hibernate d'utiliser l'ancienne stratégie **org.hibernate.cfg.DefaultNamingStrategy** en appelant **Configuration#setNamingStrategy** et en le passant à **org.hibernate.cfg.DefaultNamingStrategy#INSTANCE**.

2. Modifier les espace noms pour qu'ils soient conformes aux noms de fichiers DTD Hibernate comme dans le tableau suivant.

Tableau 3.5. Table de mappage d'espace noms DTD

Ancien Espace-nom DTD	Nouvel Espace-nom DTD
http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd	http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd
http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd	http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd

3. Modifier les variables d'environnement.

   a. Si vous utilisez Oracle avec les propriétés **materialized_clob** ou **materialized_blob**, la variable d'environnement globale **hibernate.jdbc.use_streams_for_binary** devra être définie à true.

   b. Si vous utilisez PostgreSQL avec les propriétés **CLOB** ou **BLOB**, la variable d'environnement globale **hibernate.jdbc.use_streams_for_binary** devra être définie à false.

Rapporter un bogue

3.2.2.8. Modifier les propriétés de persistance pour les applications Seam ou Hibernate migrées qui exécutent dans un environnement clusterisé.

Si vous migrez une application gérée-conteneur JPA, les propriétés de persistance correctes qui influencent la sérialisation sont passées automatiquement au conteneur.

Cependant, en raison de changements dans Hibernate, vous risquez de vous retrouver avec des problèmes de sérialisation si vous exécutez votre application migrée Seam ou Hibernate dans un environnement clusterisé. Vous risquez d'apercevoir des messages de journalisation qui ressemblent à ce qui suit :

```
javax.ejb.EJBTransactionRolledbackException: JBAS010361: Failed to
deserialize
....
Caused by: java.io.InvalidObjectException: could not resolve session
factory during session deserialization
```

```
[uuid=8aa29e74373ce3a301373ce3a44b0000, name=null]
```

Pour résoudre ces erreurs, vous devrez modifier les propriétés dans le fichier de configuration. Dans la plupart des cas, il s'agit du fichier **`persistence.xml`**. Pour les application AI Hibernate Native, il s'agit du fichier **`hibernate.cfg.xml`**.

Procédure 3.15. Définir les propriétés de persistance pour exécuter dans un environnement clusterisé.

1. Définir la valeur de **`hibernate.session_factory_name`** à un nom unique. Ce nom doit être unique pour tous les déploiements d'applications sur l'instance JBoss EAP 6. Par exemple :

```
<property name="hibernate.session_factory_name" value="jboss-seam-booking.ear_session_factory"/>
```

2. Définir la valeur de **`hibernate.ejb.entitymanager_factory_name`** à un nom unique. Ce nom doit être unique pour tous les déploiements d'applications sur l'instance JBoss EAP 6. Par exemple :

```
<property name="hibernate.ejb.entitymanager_factory_name"
value="seam-booking.ear_PersistenceUnitName"/>
```

Pour plus d'informations sur la configuration de la propriété Hibernate JPA Persistence Unit, voir Section 3.2.2.3, « Propriétés d'unité de persistence ».

Rapporter un bogue

3.2.2.9. Mettez à jour votre application pour qu'elle puisse respecter la Specification JAP 2.0

Résumé

La spécification JPA 2.0 exige qu'un contexte de persistance ne puisse pas être propagé à l'extérieur d'une transaction JTA. Si votre application utilise uniquement des contextes de persistance niveau transaction, le comportement sera le même que dans JBoss EAP 6 comme c'était le cas dans les versions précédentes du serveur d'applications et aucune modification ne sera requise. Toutefois, si votre application utilise un contexte de persistance étendu (XPC), pour permettre la mise en file d'attente ou de traitement par lot des modifications de données, vous devrez sans doute modifier votre application.

Comportement de propagation en contexte de persistance

Si votre application comprend un stateful session bean, **Bean1**, qui utilise un contexte de persistance étendu, et qu'elle appelle un stateless session bean, **Bean2**, qui utilise un contexte de persistance niveau transaction, vous devrez vous attendre à ce genre de comportement :

- Si **Bean1** démarre une transaction JTA et effectue une invocation de méthode **Bean2** avec une transaction JTA active, le comportement de JBoss EAP 6 sera le même que pour les versions précédentes et aucun changement ne sera requis.
- Si **Bean1** ne démarre pas une transaction JTA et fait une invocation de méthode **Bean2**, JBoss EAP 6 ne propagera pas le contexte de persistance dans **Bean2**. Ce comportement diffère par rapport aux versions précédentes, qui propageaient le contexte de persistance étendu dans

Bean2. Si votre application s'attend à ce que le contexte de persistance étendu soit propagé dans le bean par le gestionnaire d'entités transactionnelles, vous devrez modifier votre application pour effectuer l'invocation dans une transaction JTA active.

Rapporter un bogue

3.2.2.10. Remplacer le Cache de Second Niveau JPA/Hibernate par Infinispan

Résumé

JBoss Cache a été remplacé par Infinispan pour les caches de second niveau (2LC). Cela requiert un fichier `persistence.xml`. La syntaxe est légèrement différente, selon que vous utilisez le cache de second niveau de JPA ou Hibernate. Les exemples suivantes partent du principe que vous utilisez Hibernate.

Voici un exemple sur la façon dont les propriétés de cache de second niveau ont été spécifiées dans le fichier `persistence.xml` de JBoss EAP 5.x.

```
<property name="hibernate.cache.region.factory_class"

value="org.hibernate.cache.jbc2.JndiMultiplexedJBossCacheRegionFactory"/>
<property name="hibernate.cache.region.jbc2.cachefactory"
value="java:CacheManager"/>
<property name="hibernate.cache.use_second_level_cache" value="true"/>
<property name="hibernate.cache.region.jbc2.cfg.entity" value="mvcc-
entity"/>
<property name="hibernate.cache.region_prefix" value="services"/>
```

Suivre les étapes suivantes pour configurer Infinispan dans JBoss EAP 6

Procédure 3.16. Modifier le fichier `persistence.xml` pour utiliser Infinispan

1. **Configurer Infinispan pour une application JPA dans JBoss EAP 6**
   Voici comment vous devez spécifier les propriétés pour obtenir la même configuration pour une application JPA qui utilise Infinispan dans JBoss EAP 6 :

   ```
   <property name="hibernate.cache.use_second_level_cache"
   value="true"/>
   ```

   De plus, vous devrez spécifier un **`shared-cache-mode`** ayant pour valeur **`ENABLE_SELECTIVE`** or **`ALL`** comme suit :

   - **`ENABLE_SELECTIVE`** est la valeur par défaut recommandée. Cela signifie que les entités ne sont pas mises en cache tant que vous ne les avez pas marquées « à mettre en cache » (cachable).

     ```
     <shared-cache-mode>ENABLE_SELECTIVE</shared-cache-mode>
     ```

   - **`ALL`** signifie que les entités sont toujours mises en cache, même si vous aviez indiqué qu'elles n'étaient par à mettre en cache.

     ```
     <shared-cache-mode>ALL</shared-cache-mode>
     ```

2. **Configurer Infinispan pour une application native Hibernate dans JBoss EAP 6**

Voici comment vous devez spécifier la même configuration pour une application Hibernate native qui utilise Infinispan dans JBoss EAP 6 :

```
<property name="hibernate.cache.region.factory_class"

value="org.jboss.as.jpa.hibernate4.infinispan.InfinispanRegionFactor
y"/>
<property name="hibernate.cache.infinispan.cachemanager"
     value="java:jboss/infinispan/container/hibernate"/>
<property name="hibernate.transaction.manager_lookup_class"

value="org.hibernate.transaction.JBossTransactionManagerLookup"/>
<property name="hibernate.cache.use_second_level_cache"
value="true"/>
```

Vous devez également ajouter les dépendances suivantes au fichier **MANIFEST.MF** :

```
Manifest-Version: 1.0
Dependencies: org.infinispan, org.hibernate
```

Pour plus d'informations sur les propriétés Hibernate cache, voir Section 3.2.2.11, « Propriétés d'Hibernate Cache ».

Rapporter un bogue

3.2.2.11. Propriétés d'Hibernate Cache

Tableau 3.6. Propriétés

Nom de propriété	Description
hibernate.cache.region.factory_class	Le nom de classe d'un **CacheProvider** personnalisé.
hibernate.cache.use_minimal_puts	Booléen. Optimise l'opération cache de second niveau pour minimiser les écritures, au détriment de lectures plus fréquentes. Cette configuration est surtout utile pour les caches clusterisés et, dans Hibernate 3, est activée par défaut pour les implémentations cache clusterisées.
hibernate.cache.use_query_cache	Booléen. Active le cache de recherche. Les recherches individuelles doivent toujours être définies comme cachables.
hibernate.cache.use_second_level_cache	Booléen. Utilisé pour désactiver totalement le cache de second niveau, qui est activé par défaut pour les classes qui spécifient un mappage **<cache>**.
hibernate.cache.query_cache_factory	Le nom de classe d'une interface **QueryCache** personnalisée. La valeur par défaut est le **StandardQueryCache** intégré.

Nom de propriété	Description
`hibernate.cache.region_prefix`	Un préfixe à utiliser pour les noms régionaux de cache de second niveau.
`hibernate.cache.use_structured_entries`	Booléen. Force Hibernate à stocker des données dans un cache de second niveau dans un format plus amical pour l'utilisateur.
`hibernate.cache.default_cache_concurrency_strategy`	Configuration utilisée pour donner le nom de la statégie qu'il faut `org.hibernate.annotations.CacheConcurrencyStrategy` quand `@Cacheable` ou `@Cache` sont utilisés. `@Cache(strategy="..")` est utilisé pour remplacer cette valeur par défaut.

Rapporter un bogue

3.2.2.12. Migrer dans Hibernate Validator 4

Résumé

Hibernate Validator 4.x est une nouvelle base de code qui implémente JSR 303 - Bean Validation. Le processus de migration de Validator 3.x à 4.x est assez simple, mais vous devrez procéder à quelques changements si vous souhaitez faire migrer votre application.

Procédure 3.17. Vous devrez sans doute procéder à au moins une des tâches suivantes

1. **Accéder au ValidatorFactory par défaut**
   JBoss EAP 6 relie la ValidatorFactory par défaut au contexte JNDI sous le nom `java:comp/ValidatorFactory`.

2. **La validation déclenchée par le cycle de vie**
   Avec Hibernate Core 4, la validation basée cycle de vie est automatiquement activée par Hibernate Core.

   a. La validation a lieu sur les opérations **`INSERT`**, **`UPDATE`**, et **`DELETE`**.

   b. Vous pouvez configurer les groupes à valider par type d'événement par les propriétés suivantes :

      - `javax.persistence.validation.group.pre-persist`,
      - `javax.persistence.validation.group.pre-update`, et
      - `javax.persistence.validation.group.pre-remove`.

   Les valeurs de ces propriétés sont les noms de classe complets, séparés par des virgules des groupes à valider.

   Les groupes de validation représentent une nouvelle fonctionnalité de Bean Validation. Si vous ne souhaitez pas en profiter, aucun changement n'est requis quand vous migrez vers Hibernate Validator 4.

c. Vous pouvez désactiver la validation basée cycle de vie en définissant la propriété **`javax.persistence.validation.mode`** à **none**. Les autres valeurs possibles pour cette propriété sont **`auto`** (par défaut), **`callback`** et **`ddl`**.

3. **Configurer votre application pour la validation manuelle**

   a. Si vous souhaitez contrôler la validation manuellement, vous pourrez créer un Validateur d'une des manières suivantes :

      - Créer une instance de **`Validator`** à partir de **`ValidatorFactory`** par la méthode **`getValidator()`**.
      - Injecter des instances de Validateur dans vos EJB,CDI bean ou autre ressource Java EE injectable.

   b. Vous pouvez utiliser le **`ValidatorContext`** renvoyé par le **`ValidatorFactory.usingContext()`** pour personnaliser votre instance de Validator. Avec cet API, vous pourrez configurer des **`MessageInterpolator`**, **`TraverableResolver`** et **`ConstraintValidatorFactory`** personnalisés. Ces interfaces sont spécifiées dans Bean Validation et sont nouvelles dans Hibernate Validator 4.

4. **Modifier le code pour utiliser les nouvelles contraintes de Bean Validation**
   Les nouvelles contraintes de validation niveau Bean nécessitent des changements niveau code quand vous migrez vers Hibernate Validator 4.

   a. Pour mettre à niveau vers Hibernate Validator 4, vous devez utiliser les contraintes pour les packages suivants :

      - **`javax.validation.constraints`**
      - **`org.hibernate.validator.constraints`**

   b. Toutes les contraintes qui existent dans Hibernate Validator 3 sont toujours disponibles dans Hibernate Validator 4. Pour les utiliser, vous aurez besoin d'importer la classe indiquée, et dans certains cas, il vous faudra changer le nom ou le type du paramètre de contrainte.

5. **Usage des contraintes personnalisées**
   Dans Hibernate Validator 3, une contrainte personnalisée devait implémenter l'interface **`org.hibernate.validator.Validator`**. Dans Hibernate Validator 4, vous devez implémenter l'interface **`javax.validation.ConstraintValidator`**. Cette interface contient les mêmes méthodes **`initialize()`** et **`isValid()`** que l'interface précédente, mais la signature de la méthode a changé. De plus, l'altération **DDL** n'est plus prise en charge dans Hibernate Validator 4.

Rapporter un bogue

3.2.3. Changements JSF

3.2.3.1. Activer les Application pour qu'elles utilisent d'anciennes Versions JSF

Résumé

Si votre application utilise une ancienne version JSF, vous n'avez pas besoin de mise à niveau vers JSF 2.0. Au lieu de cela, vous pouvez créer un fichier **`jboss-déploiement-structure.xml`** pour

demander que JBoss EAP 6 utilise JSF 1.2 plutôt que JSF 2.0 avec votre déploiement d'application. Ce descripteur de déploiement spécifique de JBoss est utilisé pour contrôler le chargement de classe et est placé dans le répertoire **`META-INF /`** ou **`WEB-INF /`** de votre WAR, ou encore dans le répertoire **`META-INF /`** de votre EAR.

Ce qui suit est un exemple de fichier **`jboss-deployment-structure.xml`** qui ajoute une dépendance au module JSF 1.2 et qui exclut ou empêche le chargement automatique du module JSF 2.0.

```
<jboss-deployment-structure xmlns="urn:jboss:deployment-structure:1.0">
    <deployment>
        <dependencies>
            <module name="javax.faces.api" slot="1.2" export="true"/>
            <module name="com.sun.jsf-impl" slot="1.2" export="true"/>
        </dependencies>
    </deployment>
    <sub-deployment name="jboss-seam-booking.war">
        <exclusions>
            <module name="javax.faces.api" slot="main"/>
            <module name="com.sun.jsf-impl" slot="main"/>
        </exclusions>
        <dependencies>
            <module name="javax.faces.api" slot="1.2"/>
            <module name="com.sun.jsf-impl" slot="1.2"/>
        </dependencies>
    </sub-deployment>
</jboss-deployment-structure>
```

Rapporter un bogue

3.2.4. Modifications aux services Web

3.2.4.1. Modifications aux Services Web

JBoss EAP 6 inclut un support pour le déploiement des points de terminaison du service JAX-WS Web Service. Ce support est donné par JBossWS. Pour plus d'informations sur les Web Services, voir le chapitre qui s'intitule *JAX-WS Web Services* dans le *Development Guide* pour JBoss EAP 6 https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/.

JBossWS 4 inclut les changements suivants qui pourraient avoir un impact sur votre migration

Changements Projet API JBossWS

Les composants communs et SPI ont été refactorisés dans JBossWS 4. Le tableau suivant énumère les changements suivants qui pourraient avoir un impact sur votre migration d'application.

Tableau 3.7. Propriétés de Log Handlers de Taille

Old JAR	Old Package	New JAR	New Package
JBossWS SPI	org.jboss.wsf.spi.annotation.*	JBossWS API	org.jboss.ws.api.annotation.*
JBossWS SPI	org.jboss.wsf.spi.binding.*	JBossWS API	org.jboss.ws.api.binding.*

Old JAR	Old Package	New JAR	New Package
JBossWS SPI	org.jboss.wsf.spi.management.recording.*	JBossWS API	org.jboss.ws.api.monitoring.*
JBossWS SPI	org.jboss.wsf.spi.tools.*	JBossWS API	org.jboss.ws.api.tools.*
JBossWS SPI	org.jboss.wsf.spi.tools.ant.*	JBossWS API	org.jboss.ws.tools.ant.*
JBossWS SPI	org.jboss.wsf.spi.tools.cmd.*	JBossWS API	org.jboss.ws.tools.cmd.*
JBossWS SPI	org.jboss.wsf.spi.util.ServiceLoader	JBossWS API	org.jboss.ws.api.util.ServiceLoader
JBossWS Common	org.jboss.wsf.common.*	JBossWS API	org.jboss.ws.common.*
JBossWS Common	org.jboss.wsf.common.handler.*	JBossWS API	org.jboss.ws.api.handler.*
JBossWS Common	org.jboss.wsf.common.addressing.*	JBossWS API	org.jboss.ws.api.addressing.*
JBossWS Common	org.jboss.wsf.common.DOMUtils	JBossWS API	org.jboss.ws.api.util.DOMUtils
JBossWS Native	org.jboss.ws.annotation.EndpointConfig	JBossWS API	org.jboss.ws.api.annotation.EndpointConfig

@WebContext Annotation

Dans JBossWS 3.4.x, cette annotation a été empaquetée sous la forme **`org.jboss.wsf.spi.annotation.WebContext`** dans le projet JBossWS SPI. Dans JBossWS 4.0, cette annotation a été déplacée dans **`org.jboss.ws.api.annotation.WebContext`** dans le projet de l'API JBossWS. Si votre application inclut la dépendance obsolète, vous devrez remplacer les importations et les dépendances dans le code source de votre application et le compiler avec le nouveau JAR API JBossWS.

Il y a également un changement à un attribut non rétrocompatible. L'attribut **`String [] virtualHosts`** a été changé en **`String virtualHost`**. Dans JBoss EAP 6, vous ne pouvez spécifier qu'un seul hôte virtuel par déploiement. Si plusieurs services Web utilisent l'annotation **`@WebContext`**, la valeur de virtualHost doit être identique pour tous les points de terminaison définis dans l'archive de déploiement.

Configuration des points de terminaison

JBossWS 4.0 assure l'intégration de la pile de Services Web JBoss dans la plupart des modules du projet Apache CXF. La couche d'intégration permet l'utilisation d'API de services Web standards, y compris JAX-WS. Il permet également l'utilisation des fonctionnalités avancées d'Apache CFX sur le conteneur de JBoss EAP 6, sans nécessiter d'installation ou de configuration complexe.

Le sous-système de **webservice** dans la configuration du domaine de JBoss EAP 6 inclut les configurations de points de terminaison prédéfinies. Vous pouvez également définir vos propres configurations de points de terminaison supplémentaires. L'annotation **@org.jboss.ws.api.annotation.EndpointConfig** est utilisée pour faire référence à une configuration de point de terminaison donnée.

Pour plus d'informations sur la façon de configurer les points de terminaison du serveur JBoss, voir le chapitre intitulé *JAX-WS Web Services* dans le Guide développement de JBoss EAP 6 https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/.

jboss-webservices.xml Deployment Descriptor

JBossWS 4.0 introduit un nouveau descripteur de déploiement pour configurer des services web. Le fichier **jboss-webservices.xml** fournit des informations supplémentaires pour le déploiement donné et remplace partiellement le fichier obsolète **jboss.xml**.

Pour les déploiements du webservice EJB, l'emplacement du fichier descripteur **jboss-webservices.xml** attendu est dans le répertoire **META-INF /**. Pour les points de terminaison de service Web POJO et EJB regroupés dans le fichier WAR, l'emplacement prévu du fichier **jboss-webservices.xml** est dans le répertoire **WEB-INF /**.

Voici un exemple de fichier descripteur **jboss-webservices.xml** et un tableau décrivant les éléments.

```
<webservices>
    <context-root>foo<context-root>
    <config-name>Standard WSSecurity Endpoint</config-name>
    <config-file>META-INF/custom.xml</config-file>
    <property>
        <name>prop.name</name>
        <value>prop.value</value>
    </property>
    <port-component>
        <ejb-name>TestService</ejb-name>
        <port-component-name>TestServicePort</port-component-name>
        <port-component-uri>/*</port-component-uri>
        <auth-method>BASIC</auth-method>
        <transport-guarantee>NONE</transport-guarantee>
        <secure-wsdl-access>true</secure-wsdl-access>
    </port-component>
    <webservice-description>
        <webservice-description-name>TestService</webservice-
description-name>
        <wsdl-publish-location>file:///bar/foo.wsdl</wsdl-publish-
location>
    </webservice-description>
</webservices>
```

Tableau 3.8. jboss-webservice.xml File Element Description

Nom de l'élément	Description
context-root	Utilisé pour personnaliser le root contextuel du déploiement des services web.

Nom de l'élément	Description
config-name config-file	Utilisé pour associer un déploiement de point de terminaison à une configuration de point de terminaison donnée. Les configurations de points de terminaison sont spécifiées dans le fichier de configuration référencé ou dans le sous-système **`webservices`** de la configuration de domaine.
propriété	Utilisé pour définir une paire de valeurs de noms de propriétés simples en vue de configurer le comportement de la pile de services web.
port-component	Utilisé pour personnaliser l'URI cible du point de terminaison EJB ou pour configurer les propriétés liées à la sécurité.
webservice-description	Utilisé pour personnaliser ou pour remplacer l'emplacement publié WSDL des Services Web.

Rapporter un bogue

3.2.5. Changements JAX-RS et RESTEasy

3.2.5.1. Configurer les changements de JAX-RS and RESTEasy

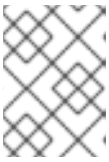
JBoss EAP 6 installe RESTEasy automatiquement, donc vous n'avez pas besoin de le configurer vous-même. Vous devez donc supprimer toute la configuration RESTEasy existante de votre fichier **`web.xml`** et la remplacer par l'une de ces trois options :

1. Sous-classe de **`javax.ws.rs.core.Application`** et utiliser l'annotation **`@ApplicationPath`**.

   C'est l'option la plus simple qui ne nécessite pas de configuration xml. Il suffit de mettre **`javax.ws.rs.core.Application`** en sous-classe dans votre application et de l'annoter par le chemin d'accès où vous souhaitez rendre vos classes de JAX-RS disponibles. Par exemple :

   ```
   @ApplicationPath("/mypath")
   public class MyApplication extends Application {
   }
   ```

   Dans l'exemple suivant, les ressources JAX-RS sont disponibles dans ***`/MY_WEB_APP_CONTEXT`***`/mypath/`.

   NOTE

   Notez que le chemin doit être spécifié **`/mypath`**, et non pas **`/mypath/*`**. Il ne doit pas y avoir de barre oblique, ni d'astérisque.

2. Mettre **`javax.ws.rs.core.Application`** en sous-classe et utiliser le fichier **`web.xml`** pour mettre en place le mappage de JAX-RS.

Si vous ne souhaitez pas utiliser l'annotation **@ApplicationPath**, vous devrez toujours mettre **javax.ws.rs.core.Application** en sous-classe. Puis, mettrez en place le mappage JAX-RS dans le fichier **web.xml**. Ainsi :

```
public class MyApplication extends Application {
}
```

```
<servlet-mapping>
   <servlet-name>com.acme.MyApplication</servlet-name>
   <url-pattern>/hello/*</url-pattern>
</servlet-mapping>
```

Dans l'exemple ci-dessus, vos ressources JAX-RS sont disponibles dans le chemin ***/MY_WEB_APP_CONTEXT*/hello**.

NOTE

Vous pouvez également utiliser cette approche pour remplacer le chemin d'application qui était mis en place pour l'annotation **@ApplicationPath**.

3. Modifier le fichier **web.xml**

Si vous ne souhaitez pas mettre **Application** en sous-classe, vous pourrez mettre en place le mappage JAX-RS dans le fichier **web.xml** comme suit :

```
<servlet-mapping>
   <servlet-name>javax.ws.rs.core.Application</servlet-name>
   <url-pattern>/hello/*</url-pattern>
</servlet-mapping>
```

Dans l'exemple ci-dessus, vos ressources JAX-RS sont disponibles dans le chemin ***/MY_WEB_APP_CONTEXT*/hello**.

NOTE

Quand vous choisissez cette option, vous n'avez qu'à ajouter le mappage. Vous n'avez pas besoin d'ajouter le servlet correspondant. Le serveur est responsable d'ajouter le servet correspondant automatiquement.

Rapporter un bogue

3.2.6. Changements au niveau domaine de sécurité LDAP

3.2.6.1. Configurer les changements de domaine de sécurité LDAP

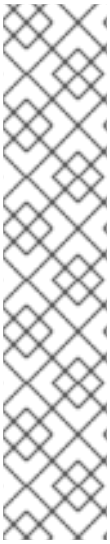
Dans JBoss EAP 5, le domaine de sécurité LDAP était configuré dans un élément **<application-policy>** du fichier **login-config.xml**. Dans JBoss EAP 6, le domaine de sécurité LDAP est configuré dans l'élément **<security-domain>** de fichier de configuration du serveur. Pour le serveur autonome, il s'agit du fichier **standalone/configuration/standalone.xml**. Si vous exécutez le serveur dans un domaine géré, il s'agira du fichier **domain/configuration/domain.xml**.

Ce qui suit est un exemple de configuration de domaine de sécurité LDAP du fichier **login-config.xml** de JBoss EAP 5 :

```
<application-policy name="mcp_ldap_domain">
  <authentication>
    <login-module code="org.jboss.security.auth.spi.LdapExtLoginModule"
flag="required">
      <module-option
name="java.naming.factory.initial">com.sun.jndi.ldap.LdapCtxFactory</modul
e-option>
      <module-option
name="java.naming.security.authentication">simple</module-option>
      ....
    </login-module>
  </authentication>
</application-policy>
```

Ce qui suit est un exemple de configuration du fichier de configuration de serveur de JBoss EAP 6 :

```
<subsystem xmlns="urn:jboss:domain:security:1.2">
  <security-domains>
    <security-domain name="mcp_ldap_domain" cache-type="default">
      <authentication>
        <login-module code="org.jboss.security.auth.spi.LdapLoginModule"
flag="required">
          <module-option name="java.naming.factory.initial"
value="com.sun.jndi.ldap.LdapCtxFactory"/>
          <module-option name="java.naming.security.authentication"
value="simple"/>
          ...
        </login-module>
      </authentication>
    </security-domain>
  </security-domains>
</subsystem>
```



NOTE

L'analyseur XML a changé dans JBoss EAP 6. Dans JBoss EAP 5, on spécifiait les options de module comme contenu d'élément ainsi :

```
<module-option
name="java.naming.factory.initial">com.sun.jndi.ldap.LdapCtxFac
tory</module-option>
```

Maintenant, les options de module doivent être spécifiées comme attributs d'éléments par "value=" comme suit :

```
<module-option name="java.naming.factory.initial"
value="com.sun.jndi.ldap.LdapCtxFactory"/>
```

Rapporter un bogue

3.2.7. Changements HornetQ

3.2.7.1. HornetQ et NFS

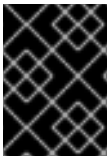
Dans la plupart des cas, NFS n'est pas une méthode appropriée de stocker des données JMS pour utilisation avec HornetQ, si vous utilisez NIO comme type de journal, en raison de la façon dont fonctionne le mécanisme de blocage synchrone. Cependant, NFS peut être utilisé dans certaines circonstances, uniquement sur les serveurs Red Hat Enterprise Linux. C'est en raison de l'implémentation NFS utilisée par Red Hat Enterprise Linux.

L'implémentation NFS de Red Hat Linux utilisée supporte à la fois le direct E/S (ouverture des fichiers avec l'indicateur O_DIRECT défini), et l'E/S asynchrone basé noyau. Avec ces deux fonctionnalités présentes, il est possible d'utiliser NFS comme option de stockage, sous conditions de configuration strictes :

- Le cache client Red Hat Enterprise Linux NFS doit être désactivé.

IMPORTANT

Le journal du serveur doit être vérifié après le démarrage de JBoss EAP 6, pour s'assurer que la bibliothèque native est bien chargée, et que le type de journal ASYNCIO est utilisé. Si la bibliothèque native ne se charge pas, HornetQ échouera dans le journal NIO, et cela va être précisé dans le journal du serveur.



IMPORTANT

La bibliothèque native qui implémente des e/s asynchrones exige que `libaio` soit installée sur le système Red Hat Enterprise Linux sur lequel JBoss EAP 6 exécute.

Rapporter un bogue

3.2.7.2. Configurer un pontage JMS pour migrer les Messages JMS dans JBoss EAP 6

JBoss EAP 6 a remplacé JBoss Messaging par HornetQ comme implémentation JMS par défaut. La manière la plus simple de migrer les Messages JMS d'un environnement à un autre est d'utiliser un pontage JMS. Le rôle d'un pontage JMS est de consommer des messages d'une destination source JMS, et de les envoyer vers une destination cible JMS. Vous pouvez configurer et déployer un pontage JMS dans un serveur JBoss EAP 5.x ou dans JBoss 6.1 ou autre serveur plus récent.

Pour obtenir des informations sur la façon de migrer des messages JMS de JBoss EAP 5.x vers JBoss EAP 6.x, voir : Section 3.2.7.3, « Créer un pontage JMS »

Rapporter un bogue

3.2.7.3. Créer un pontage JMS

Résumé

Un pontage JMS consomme des messages d'une file d'attente ou une topic JMS source et les envoie à une file d'attente JMS de cible ou un sujet, se trouvant en général sur un autre serveur. Il peut être utilisé pour faire un pontage entre les messages entre les serveurs JMS, tant qu'ils sont compatibles avec JMS 1.1. Les ressources JMS de source et de destination sont cherchées à l'aide de JNDI et les classes de client doivent être regroupées dans un module pour la recherche JNDI. Le nom du module est ensuite déclaré dans la configuration de pontage JMS.

Procédure 3.18. Créer un pontage JMS

Cette procédure montre comment configurer un pontage JMS pour faire migrer des messages d'un serveur JBoss EAP 5.x vers un serveur JBoss EAP 6.

1. **Configurer le Pontage sur le serveur JBoss EAP 5.x source**
   Pour éviter les conflits de classes entre les sorties, vous devrez suivre les étapes suivantes pour configurer le pontage JMS dans JBoss EAP 5.x. Les noms du répertoire SAR et du pontage sont arbitraires et peuvent être changés si vous le souhaitez.

   a. Créer un sous-répertoire dans le répertoire de déploiement de JBoss EAP 5 qui puisse contenir le SAR, comme par exemple : ***EAP5_HOME*/server/*PROFILE_NAME*/deploy/myBridge.sar/**.

   b. Créer un sous-répertoire nommé **META-INF** dans ***EAP5_HOME*/server/*PROFILE_NAME*/deploy/myBridge.sar/**.

   c. Créer un fichier **jboss-service.xml** dans le répertoire ***EAP5_HOME*/server/*PROFILE_NAME*/deploy/myBridge.sar/META-INF/**. Ce fichier devra contenir des informations semblables à celles de l'exemple suivant :

```
<server>
   <loader-repository>
      com.example:archive=unique-archive-name
      <loader-repository-
config>java2ParentDelegation=false</loader-repository-config>
   </loader-repository>

   <!-- JBoss EAP 6 JMS Provider -->
   <mbean code="org.jboss.jms.jndi.JMSProviderLoader"
name="jboss.messaging:service=JMSProviderLoader,name=EnterpriseAp
plicationPlatform6JMSProvider">
      <attribute
name="ProviderName">EnterpriseApplicationPlatform6JMSProvider</at
tribute>
      <attribute
name="ProviderAdapterClass">org.jboss.jms.jndi.JNDIProviderAdapte
r</attribute>
      <attribute
name="FactoryRef">jms/RemoteConnectionFactory</attribute>
      <attribute
name="QueueFactoryRef">jms/RemoteConnectionFactory</attribute>
      <attribute
name="TopicFactoryRef">jms/RemoteConnectionFactory</attribute>
      <attribute name="Properties">

java.naming.factory.initial=org.jboss.naming.remote.client.Initia
lContextFactory

java.naming.provider.url=remote://EnterpriseApplicationPlatform6h
ost:4447
         java.naming.security.principal=jbossuser
         java.naming.security.credentials=jbosspass
      </attribute>
   </mbean>
```

```
   <mbean code="org.jboss.jms.server.bridge.BridgeService"
name="jboss.jms:service=Bridge,name=MyBridgeName" xmbean-
dd="xmdesc/Bridge-xmbean.xml">
      <depends optional-attribute-
name="SourceProviderLoader">jboss.messaging:service=JMSProviderLo
ader,name=JMSProvider</depends>
      <depends optional-attribute-
name="TargetProviderLoader">jboss.messaging:service=JMSProviderLo
ader,name=EnterpriseApplicationPlatform6JMSProvider</depends>
      <attribute
name="SourceDestinationLookup">/queue/A</attribute>
      <attribute
name="TargetDestinationLookup">jms/queue/test</attribute>
      <attribute name="QualityOfServiceMode">1</attribute>
      <attribute name="MaxBatchSize">1</attribute>
      <attribute name="MaxBatchTime">-1</attribute>
      <attribute name="FailureRetryInterval">60000</attribute>
      <attribute name="MaxRetries">-1</attribute>
      <attribute name="AddMessageIDInHeader">false</attribute>
      <attribute name="TargetUsername">jbossuser</attribute>
      <attribute name="TargetPassword">jbosspass</attribute>
   </mbean>
</server>
```

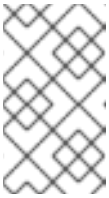
NOTE

Le **`load-repository`** est là pour veiller à ce que le SAR ait un chargeur de classes isolé. Notez également que le JNDI Look-up et que le Pont «cible» incluent les informations d'identification de sécurité pour l'utilisateur "jbossuser" ayant pour mot de passe "jbosspass". C'est parce que JBoss EAP 6 est sécurisé par défaut. L'utilisateur qui ne nomme "jbossuser" et ayant comme mot de passe "jbosspass" a été créé dans **`Domaine d'application`** avec le rôle **`invité`** utilisant le script ***`EAP_HOME`*`/bin/add_user.sh`**.

d. Copier les JAR suivants à partir du répertoire ***`EAP_HOME`*`/modules/system/layers/base/`** dans le répertoire ***`EAP5_HOME`*`/server/`*`PROFILE_NAME`*`/deploy/myBridge.sar/`**. Remplacer chaque *VERSION_NUMBER* par le numéro de version qui se trouve dans la distro JBoss EAP 6.

- **`org/hornetq/main/hornetq-core-`*`VERSION_NUMBER`*`.jar`**
- **`org/hornetq/main/hornetq-jms-`*`VERSION_NUMBER`*`.jar`**
- **`org/jboss/ejb-client/main/jboss-ejb-client-`*`VERSION_NUMBER`*`.jar`**
- **`org/jboss/logging/main/jboss-logging-`*`VERSION_NUMBER`*`.jar`**
- **`org/jboss/logmanager/main/jboss-logmanager-`*`VERSION_NUMBER`*`.jar`**
- **`org/jboss/marshalling/main/jboss-marshalling-`*`VERSION_NUMBER`*`.jar`**
- **`org/jboss/marshalling/river/main/jboss-marshalling-river-`*`VERSION_NUMBER`*`.jar`**

- **org/jboss/remote-naming/main/jboss-remote-naming-*VERSION_NUMBER*.jar**
- **org/jboss/remoting3/main/jboss-remoting-*VERSION_NUMBER*.jar**
- **org/jboss/sasl/main/jboss-sasl-*VERSION_NUMBER*.jar**
- **org/jboss/netty/main/netty-*VERSION_NUMBER*.jar**
- **org/jboss/remoting3/remote-jmx/main/remoting-jmx-*VERSION_NUMBER*.jar**
- **org/jboss/xnio/main/xnio-api-*VERSION_NUMBER*.jar**
- **org/jboss/xnio/nio/main.xnio-nio-*VERSION_NUMBER*.jar**



NOTE

Ne vous contentez pas de copier ***EAP_HOME*/bin/client/jboss-client.jar** parce que les classes API javax vont entrer en conflit avec celles de JBoss EAP 5.x.

2. **Configurer un pontage déployé dans de serveur JBoss EAP 6.x de destination**
Dans JBoss EAP 6.1 et dans les versions supérieures, le pontage JMS peut servir à combler des messages depuis n'importe quel serveur compatible JMS 1.1. Comme les ressources JMS source et cible sont recherchées à l'aide de JNDI, les classes de recherche JNDI du fournisseur de messagerie source, ou fournisseur de messages, doivent être regroupées dans un Module de JBoss. Les étapes suivantes utilisent le fournisseur de messages fictive « MyCustomMQ » a titre d'exemple.

   a. Créer le module JBoss pour le fournisseur de messagerie.

      i. Créer une structure de répertoire sous ***EAP_HOME*/modules/system/layers/base/** pour le nouveau module. Le sous-répertoire **main/** contiendra les JAR du client et le fichier **module.xml**. L'exemple suivant est un exemple de structure de répertoires créé pour le fournisseur de messageries MyCustomMQ : ***EAP_HOME*/modules/system/layers/base/org/mycustommq/main/**

      ii. Dans le sous-répertoire **main/**, créer un fichier **module.xml** qui contienne la définition de module suivante pour le fournisseur de messagerie. Ce qui suit est un exemple de **module.xml** créé pour le fournisseur de messagerie MyCustomMQ.

```
<?xml version="1.0" encoding="UTF-8"?>
<module xmlns="urn:jboss:module:1.1" name="org.mycustommq">
    <properties>
        <property name="jboss.api" value="private"/>
    </properties>

    <resources>
        <!-- Insert resources required to connect to the
source or target   -->
        <resource-root path="mycustommq-1.2.3.jar" />
        <resource-root path="mylogapi-0.0.1.jar" />
    </resources>
```

```
    <dependencies>
       <!-- Add the dependencies required by JMS Bridge code
-->
       <module name="javax.api" />
       <module name="javax.jms.api" />
       <module name="javax.transaction.api"/>
       <!-- Add a dependency on the org.hornetq module since
we send         -->
       <!-- messages tothe HornetQ server embedded in the
local EAP instance -->
       <module name="org.hornetq" />
    </dependencies>
</module>
```

iii. Copier les JAR de fournisseur de messagerie requises pour la recherche JNDI des ressources source vers le sous-répertoire **`main/`** du module. La structure du répertoire du module MyCustomMQ ne devra pas ressembler à ce qui suit.

```
modules/
`-- system
    `-- layers
        `-- base
            `-- org
                `-- mycustommq
                    `-- main
                        |-- mycustommq-1.2.3.jar
                        |-- mylogapi-0.0.1.jar
                        |-- module.xml
```

b. Configurer le pontage JMS dans le sous-système de **`messaging`** du serveur JBoss EAP.

i. Avant de commencer, arrêtez le serveur et sauvegardez les fichiers de configuration du serveur actuel. Si vous exécutez un serveur autonome, il s'agira du fichier ***`EAP_HOME`*** **`/standalone/configuration/standalone-full-ha.xml`**. Si vous exécutez un domaine géré, sauvegardez les fichiers ***`EAP_HOME`*** **`/domain/configuration/host.xml`** et ***`EAP_HOME`*** **`/domain/configuration/domain.xml`**.

ii. Ajouter l'élément **`jms-bridge`** au sous-système **`messaging`** dans le fichier de configuration du serveur. Les éléments **`source`** et **`target`** procurent les noms des ressources JMS utlisées pour les recherches JNDI. Si les informations d'authentification **`user`** et **`password`** sont spécifiées, elles seront passées comme arguments quand une connexion JMS est créée.

Ce qui suit est un exemple d'élément **`jms-bridge`** configuré pour le fournisseur de messagerie MyCustomMQ :

```
<subsystem xmlns="urn:jboss:domain:messaging:1.3">
   ...
   <jms-bridge name="myBridge" module="org.mycustommq">
      <source>
         <connection-factory name="ConnectionFactory"/>
         <destination name="sourceQ"/>
         <user>user1</user>
```

```
         <password>pwd1</password>
         <context>
            <property key="java.naming.factory.initial"
value="org.mycustommq.jndi.MyCustomMQInitialContextFactory"/>
            <property key="java.naming.provider.url"
value="tcp://127.0.0.1:9292"/>
         </context>
      </source>
      <target>
         <connection-factory name="java:/ConnectionFactory"/>
         <destination name="/jms/targetQ"/>
      </target>
      <quality-of-service>DUPLICATES_OK</quality-of-service>
      <failure-retry-interval>500</failure-retry-interval>
      <max-retries>1</max-retries>
      <max-batch-size>500</max-batch-size>
      <max-batch-time>500</max-batch-time>
      <add-messageID-in-header>true</add-messageID-in-header>
   </jms-bridge>
</subsystem>
```

Dans l'exemple suivant, les propriétés JNDI sont définies dans l'élément **context** pour la **source**. Si l'élément **context** est omis, comme dans l'exemple **target** ci-dessus, les ressources JMS seront recherchées dans l'instance locale.

Rapporter un bogue

3.2.7.4. Migrer votre Application pour qu'elle utilise HornetQ comme JMS Provider

JBoss Messaging n'est plus inclus dans JBoss EAP 6. Si votre application utilise JBoss Messaging comme fournisseur de messagerie, vous devrez remplacer le code JBoss Messaging par HornetQ.

Procédure 3.19. Avant de commencer

1. Fermer le client et le serveur.
2. Faire une copie de sauvegarde de données JBoss Messaging. Les données du message sont stockées dans la base de données dans des tables ayant pour préfixe **JBM_**.

Procédure 3.20. Changer le fournisseur en HornetQ

1. **Transférer les configurations**
   Transférer les configurations JBoss Messaging existantes dans la configuration JBoss EAP 6. Les configurations suivantes se trouvent dans les descripteurs de déploiement qui se situent dans le serveur JBoss Messaging :
   - Configuration du Service des fabriques de connexions

     Cette configuration décrit les fabriques de connexions JMS déployées dans le serveur JBoss Messaging. JBoss Messaging configure les fabriques de connexions dans un fichier nommé **connection-factories-service.xml** qui se trouve dans le répertoire de déploiement du serveur d'application.
   - Configuration de la destination

     Cette configuration décrit les files d'attente JMS et thèmes déployés avec le serveur

JBoss Messaging. Par défaut, JBoss Messaging configure des destinations dans un fichier nommé **`destination-service.xml`** qui se trouve dans le répertoire de déploiement du serveur d'application.

- Configuration du Service de pontage des messages

  Cette configuration comprend les services de pontage déployés dans le serveur JBoss Messaging. Aucun pont n'est déployé par défaut, donc le nom du fichier de déploiement dépend de votre installation JBoss Messaging.

2. **Modifier votre code d'application**
   Si le code d'application utilise JMS standard, aucune modification de code n'est nécessaire. Cependant, si l'application doit se connecter à un cluster, vous devez soigneusement examiner la documentation HornetQ sur la sémantique de clustering. Clustering déborde le cadre de la spécification JMS. HornetQ et JBoss Messaging ont adopté des approches très différentes dans leurs implémentations respectives de la fonctionnalité de clustering.

   Si l'application utilise les fonctionnalités spécifiques à JBoss Messaging, vous devez modifier le code pour utiliser les fonctionnalités équivalentes disponibles dans HornetQ.

   Pour plus d'informations sur la façon de configurer la messagerie dans HornetQ, voir Section 3.2.7.5, « Configurer la Messagerie dans HornetQ »

3. **Migration des messages existants**
   Déplacez tous les messages dans la base de données JBoss Messaging du journal de HornetQ à l'aide d'un pont JMS. Les instructions pour configurer le pont JMS se trouvent ici : Section 3.2.7.2, « Configurer un pontage JMS pour migrer les Messages JMS dans JBoss EAP 6 ».

Rapporter un bogue

3.2.7.5. Configurer la Messagerie dans HornetQ

La méthode recommandée pour configurer la messagerie dans JBoss EAP 6 est soit la Console de gestion, soit Management CLI. Vous pouvez effectuer des modifications persistantes avec l'un ou l'autre de ces outils de gestion sans avoir besoin de modifier manuellement les fichiers de configuration **`standalone.xml`** ou **`domain.xml`**. Cependant, il est utile de se familiariser avec les composants de messagerie des fichiers de configuration par défaut, où les exemples de documentation utilisant des outils de gestion donnent des extraits de fichiers de configuration comme référence.

Rapporter un bogue

3.2.8. Changements au clustering

3.2.8.1. Changements à votre application pour le clustering

1. **Démarrez JBoss EAP 6 avec le clustering activé**
   Pour activer le clustering dans JBoss EAP 5.x, vous devrez démarrer vos instances de serveur avec le profil **`all`** ou undérivé, comme :

   ```
   $ EAP5_HOME/bin/run.sh -c all
   ```

   Dans JBoss EAP 6, la méthode d'activation du clustering dépend si les serveurs sont autonomes ou s'ils exécutent dans un domaine géré.

a. **Activer le clustering pour les serveurs qui exécutent dans un domaine géré**
Pour activer le clustering pour les serveurs déjà démarrés, qui utilisent le contrôleur de domaines, mettez à jour votre **domain.xml** et désignez un groupe de serveurs qui utilise le profil **ha** et un groupe de liaisons de sockets **ha-sockets**. Par exemple :

```
<server-groups>
  <server-group name="main-server-group" profile="ha">
    <jvm name="default">
      <heap size="64m" max-size="512m"/>
    </jvm>
    <socket-binding-group ref="ha-sockets"/>
  </server-group>
</server-group>
```

b. **Activer le clustering pour les serveurs autonomes**
Afin d'activer le clustering dans les serveurs autonomes, démarrez le serveur par le fichier de configuration qui convient comme suit :

```
$ EAP_HOME/bin/standalone.sh --server-config=standalone-ha.xml -
Djboss.node.name=UNIQUE_NODE_NAME
```

2. **Indiquer l'adresse de liaison**
Dans JBoss EAP 5.x, vous devez normalement indiquer l'adresse de liaison utilisée pour le clustering avec l'argument de ligne de commande **-b** comme suit :

```
$ EAP5_HOME/bin/run.sh -c all -b 192.168.0.2
```

JBoss EAP 6 relie les sockets aux adresses IP et aux interfaces contenues dans les éléments **<interfaces>** des fichiers **standalone.xml**, **domain.xml** et **host.xml**. Les configurations standards fournies dans JBoss EAP incluent deux configurations d'interface :

```
<interfaces>
    <interface name="management">
        <inet-address
value="${jboss.bind.address.management:127.0.0.1}"/>
    </interface>
    <interface name="public">
       <inet-address value="${jboss.bind.address:127.0.0.1}"/>
    </interface>
</interfaces>
```

Ces configurations d'interface utilisent les valeurs des propriétés système **jboss.bind.address.management** et **jboss.bind.address**. Si ces propriétés système ne sont pas définies, la valeur par défaut **127.0.0.1** sera utilisée pour chaque valeur.

Vous pouvez également spécifier l'adresse de liaison en argument de ligne de commande lorsque vous démarrez le serveur ou vous pouvez la définir explicitement dans le fichier de configuration du serveur JBoss EAP 6.

- Spécifier l'argument de liaison en ligne de commande quand vous démarrez le serveur JBoss EAP autonome.

  L'exemple suivant explique comment indiquer l'adresse de liaison en ligne de commande pour un serveur autonome :

```
EAP_HOME/bin/standalone.sh -Djboss.bind.address=127.0.0.1
```

NOTE

Vous pouvez également faire usage de l'argument **-b**, un raccourci de **-Djboss.bind.address=127.0.0.1**:

```
EAP_HOME/bin/standalone.sh -b=127.0.0.1
```

Le format de syntaxe JBoss EAP 5 est également pris en charge :

```
EAP_HOME/bin/standalone.sh -b 127.0.0.1
```

Notez que l'argument **-b** ne fait que changer l'interface **publique**. Il n'affecte pas l'interface de **management**.

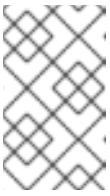
- Indiquer l'adresse de liaison dans le fichier de configuration du serveur.

  Pour les serveurs qui exécutent en domaines gérés, indiquer les adresses de liaison dans le fichier **domain/configuration/host.xml**. Pour les serveurs autonomes, indiquer les adresses de liaison dans le fichier **standalone-ha.xml**.

  Dans l'exemple suivant, l'interface **public** est spécifiée comme interface par défaut pour tous les sockets au sein du groupe de liaison de socket **ha-sockets**.

```
<interfaces>
  <interface name="management">
    <inet-address value="192.168.0.2"/>
  </interface>
  <interface name="public">
    <inet-address value="192.168.0.2"/>
  </interface>
</interfaces>
```

```
<socket-binding-groups>
  <socket-binding-group name="ha-sockets" default-
interface="public">
    <!-- ... -->
  </socket-binding-group>
</socket-binding-groups>
```



NOTE

Si vous spécifiez l'adresse de liaison sous forme de valeur codée en dur et non pas sous forme de propriété système de configuration, vous ne pourrez pas remplacer cette valeur codée par un argument en ligne de commande.

3. **Configurer `jvmRoute` pour supporter mod_jk et mod_proxy**
   Dans JBoss EAP 5, le serveur web **jvmRoute** a été configuré à l'aide d'une propriété dans le fichier **server.xml**. Dans JBoss EAP 6, l'attribut **jvmRoute** est configuré dans le sous-système web du fichier de configuration de serveur à l'aide de l'attribut **instance-id** comme

suit :

```
<subsystem xmlns="urn:jboss:domain:web:1.1" default-virtual-
server="default-host" native="false" instance-id="
{JVM_ROUTE_SERVER}">
```

{JVM_ROUTE_SERVER} ci-dessus doit être remplacé par l'ID du serveur jvmRoute.

instance-id peut également être défini par le biais de la console de gestion.

4. **Indiquer l'adresse multidiffusion et le port**
Dans JBoss EAP 5.x, vous pouvez spécifier l'adresse multidiffusion et le port utilisés pour les communications intra-cluster par les arguments de ligne de commande **-u** et **-m**, respectivement, comme ceci :

```
$ EAP5_HOME/bin/run.sh -c all -u 228.11.11.11 -m 45688
```

Dans JBoss EAP 6, vous pouvez spécifier l'adresse multidiffusion et le port utilisés pour la communication entre les cluster par la liaison de socket référencée par la pile de protocoles JGroup qui convient.

```
<subsystem xmlns="urn:jboss:domain:jgroups:1.0" default-stack="udp">
    <stack name="udp">
        <transport type="UDP" socket-binding="jgroups-udp"/>
        <!-- ... -->
    </stack>
</subsystem>
```

```
<socket-binding-groups>
    <socket-binding-group name="ha-sockets" default-
interface="public">
        <!-- ... -->
        <socket-binding name="jgroups-udp" port="55200" multicast-
address="228.11.11.11" multicast-port="45688"/>
        <!-- ... -->
    </socket-binding-group>
</socket-binding-groups>
```

Si vous voulez spécifier l'adresse de multidiffusion et le port dans la ligne de commande, vous pouvez définir l'adresse de multidiffusion et les ports comme des propriétés système et ensuite utiliser ces propriétés sur la ligne de commande lorsque vous démarrez le serveur. Dans l'exemple suivant, **jboss.mcast.addr** est le nom de la variable pour l'adresse de multidiffusion et **jboss.mcast.port** est le nom de la variable pour le port.

```
<socket-binding name="jgroups-udp" port="55200"
 multicast-address="${jboss.mcast.addr:230.0.0.4}" multicast-
port="${jboss.mcast.port:45688}"/>
```

Vous pouvez alors démarrer votre serveur en utilisant les arguments de ligne de commande suivants :

```
$ EAP_HOME/bin/domain.sh -Djboss.mcast.addr=228.11.11.11 -
Djboss.mcast.port=45688
```

5. **Utiliser une pile de protocoles différente**
   Dans JBoss EAP 5.x, vous pouviez manipuler la pile de protocoles par défaut utilisée pour tous les services de clustering qui utilisaient la propriété système **`jboss.default.jgroups.stack`**.

   ```
   $ EAP5_HOME/bin/run.sh -c all -Djboss.default.jgroups.stack=tcp
   ```

   Dans JBoss EAP 6, la pile de protocoles par défaut est définie par le sous-système JGroups dans **`domain.xml`** ou **`standalone-ha.xml`**:

   ```
   <subsystem xmlns="urn:jboss:domain:jgroups:1.0" default-stack="udp">
       <stack name="udp">
           <!-- ... -->
       </stack>
   </subsystem>
   ```

6. **Remplacer la réplication de Buddy**
   JBoss EAP 5.x utilise JBoss Cache Buddy Replication pour supprimer la réplication des données dans toutes les instances d'un cluster.

   Dans JBoss EAP 6, la réplication de Buddy a été remplacée par le cache distribué d'Infinispan, connu également sous le nom mode **`DIST`**. La distribution est un mode de gestion de clusters puissant qui permet à Infinispan de mettre à échelle en linéaire lorsque un certain nombre de serveurs sont ajoutés au cluster. Voici un exemple sur la façon de configurer le serveur pour utiliser le mode de mise en cache de DIST.

   a. Ouvrir une ligne de commandes et démarrer le serveur avec un Profil HA ou un FUll Profile, comme par exemple :

      ```
      EAP_HOME/bin/standalone.sh -c standalone-ha.xml
      ```

   b. Ouvrir une nouvelle ligne de commandes et connectez-vous au Management CLI.

      - Dans Linux, saisir ce qui suit au niveau de la ligne de commande :

        ```
        $ EAP_HOME/bin/jboss-cli.sh --connect
        ```

      - Dans Windows, saisir ce qui suit au niveau de la ligne de commande :

        ```
        C:\>EAP_HOME\bin\jboss-cli.bat --connect
        ```

      Vous devriez voir apparaître le résultat suivant :

      ```
      Connected to standalone controller at localhost:9999
      ```

   c. Lancer les commandes suivantes :

      ```
      /subsystem=infinispan/cache-container=web/:write-
      attribute(name=default-cache,value=dist)
      /subsystem=infinispan/cache-container=web/distributed-
      cache=dist/:write-attribute(name=owners,value=3)
      :reload
      ```

Vous devriez voir la réponse suivante après chaque commande :

```
"outcome" => "success"
```

Ces commandes modifient l'élément **dist <distributed-cache>** dans la configuration **web <cache-container>** du sous-système **infinispan** du fichier **standalone-ha.xml** comme suit :

```
<cache-container name="web" aliases="standard-session-cache"
default-cache="dist"
module="org.jboss.as.clustering.web.infinispan">
    <transport lock-timeout="60000"/>
    <replicated-cache name="repl" mode="ASYNC" batching="true">
        <file-store/>
    </replicated-cache>
    <replicated-cache name="sso" mode="SYNC" batching="true"/>
    <distributed-cache name="dist" owners="3" l1-lifespan="0"
mode="ASYNC" batching="true">
        <file-store/>
    </distributed-cache>
</cache-container>
```

Pour plus d'informations, voir le chapitre intitulé *Clustering in Web Applications* du *Development Guide* de JBoss EAP 6 qui se situe dans le Portail clients https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/.

Rapporter un bogue

3.2.8.2. Implémenter un HA Singleton

Résumé

La procédure suivante illustre comment déployer un Service qui se trouve dans un decorator de SingletonService et qui est utilisé comme service singleton dans l'ensemble du cluster. Le service active une minuterie programmée, qui n'est lancée qu'à une seule reprise dans le cluster.

Procédure 3.21. Implémenter un Service HA Singleton

1. **Rédiger une application de Service HA Singleton**
   Voici un exemple simple de **Service** se trouvant dans le decorator **SingletonService** devant être déployé comme service singleton. On trouvera un exemple complet dans le quickstart de **cluster-ha-singleton** qui est livré avec Red Hat JBoss Enterprise Application Platform 6. Ce quickstart contient toutes les instructions pour générer et déployer l'application.

   a. **Créer un service.**
   Voici un exemple de service :

```
package org.jboss.as.quickstarts.cluster.hasingleton.service.ejb;

import java.util.Date;
import java.util.concurrent.atomic.AtomicBoolean;

import javax.naming.InitialContext;
import javax.naming.NamingException;
```

```
import org.jboss.logging.Logger;
import org.jboss.msc.service.Service;
import org.jboss.msc.service.ServiceName;
import org.jboss.msc.service.StartContext;
import org.jboss.msc.service.StartException;
import org.jboss.msc.service.StopContext;


/**
 * @author <a href="mailto:wfink@redhat.com">Wolf-Dieter Fink</a>
 */
public class HATimerService implements Service<String> {
    private static final Logger LOGGER =
Logger.getLogger(HATimerService.class);
    public static final ServiceName SINGLETON_SERVICE_NAME =
ServiceName.JBOSS.append("quickstart", "ha", "singleton",
"timer");

    /**
     * A flag whether the service is started.
     */
    private final AtomicBoolean started = new
AtomicBoolean(false);

    /**
     * @return the name of the server node
     */
    public String getValue() throws IllegalStateException,
IllegalArgumentException {
        LOGGER.infof("%s is %s at %s",
HATimerService.class.getSimpleName(), (started.get() ? "started"
: "not started"), System.getProperty("jboss.node.name"));
        return "";
    }

    public void start(StartContext arg0) throws StartException {
        if (!started.compareAndSet(false, true)) {
            throw new StartException("The service is still
started!");
        }
        LOGGER.info("Start HASingleton timer service '" +
this.getClass().getName() + "'");

        final String node =
System.getProperty("jboss.node.name");
        try {
            InitialContext ic = new InitialContext();
            ((Scheduler) ic.lookup("global/jboss-cluster-ha-
singleton-
service/SchedulerBean!org.jboss.as.quickstarts.cluster.hasingleto
n.service.ejb.Scheduler")).initialize("HASingleton timer @" +
node + " " + new Date());
        } catch (NamingException e) {
            throw new StartException("Could not initialize
timer", e);
```

```
        }
    }

    public void stop(StopContext arg0) {
        if (!started.compareAndSet(true, false)) {
            LOGGER.warn("The service '" +
this.getClass().getName() + "' is not active!");
        } else {
            LOGGER.info("Stop HASingleton timer service '" +
this.getClass().getName() + "'");
            try {
                InitialContext ic = new InitialContext();
                ((Scheduler) ic.lookup("global/jboss-cluster-ha-
singleton-
service/SchedulerBean!org.jboss.as.quickstarts.cluster.hasingleto
n.service.ejb.Scheduler")).stop();
            } catch (NamingException e) {
                LOGGER.error("Could not stop timer", e);
            }
        }
    }
}
```

b. **Créer un activator qui installe le `Service` en tant que singleton clusterisé.**
La liste suivante est un exemple d'activtor de service qui installe le **`HATimerService`** comme service singleton clusterisé :

```
package org.jboss.as.quickstarts.cluster.hasingleton.service.ejb;

import org.jboss.as.clustering.singleton.SingletonService;
import org.jboss.logging.Logger;
import org.jboss.msc.service.DelegatingServiceContainer;
import org.jboss.msc.service.ServiceActivator;
import org.jboss.msc.service.ServiceActivatorContext;
import org.jboss.msc.service.ServiceController;


/**
 * Service activator that installs the HATimerService as a
clustered singleton service
 * during deployment.
 *
 * @author Paul Ferraro
 */
public class HATimerServiceActivator implements ServiceActivator
{
    private final Logger log = Logger.getLogger(this.getClass());

    @Override
    public void activate(ServiceActivatorContext context) {
        log.info("HATimerService will be installed!");

        HATimerService service = new HATimerService();
        SingletonService<String> singleton = new
SingletonService<String>(service,
HATimerService.SINGLETON_SERVICE_NAME);
```

```
        /*
         * To pass a chain of election policies to the singleton,
for example,
         * to tell JGroups to prefer running the singleton on a
node with a
         * particular name, uncomment the following line:
         */
        // singleton.setElectionPolicy(new
PreferredSingletonElectionPolicy(new
SimpleSingletonElectionPolicy(), new
NamePreference("node2/cluster")));

        singleton.build(new
DelegatingServiceContainer(context.getServiceTarget(),
context.getServiceRegistry()))
                .setInitialMode(ServiceController.Mode.ACTIVE)
                .install()
        ;
    }
}
```

NOTE

L'exemple de code ci-dessus utilise une classe, **`org.jboss.as.clustering.singleton.SingletonService`**, qui fait partie de l'API privée de JBoss EAP. Une API publique deviendra disponible dans la version EAP 7 et les classes privées seront obsolètes, mais ces classes seront entretenues et rendues disponible pendant toute la durée du cycle de version EAP 6.x.

c. **Créer un fichier ServiceActivator**
Créer un fichier nommé **`org.jboss.msc.service.ServiceActivator`** dans le répertoire **`resources/META-INF/services/`** de l'application. Ajouter une ligne contenant le nom complet de la classe de Service Activator créée dans l'étape précédente.

```
org.jboss.as.quickstarts.cluster.hasingleton.service.ejb.HATimerS
erviceActivator
```

d. **Créer un bean singleton qui implémente une minuterie de signleton à utiliser dans tout le cluster.**
Ce bean Singleton ne doit pas avoir d'interface distante et ne doit pas référencer son interface locale à partir d'un autre EJB d'application. Cela évite une recherche client ou de la part d'un autre composant et assure un parfait contrôle du Singleton par le SingletonService.

i. **Créer une interface de planification.**

```
package
org.jboss.as.quickstarts.cluster.hasingleton.service.ejb;

/**
 * @author <a href="mailto:wfink@redhat.com">Wolf-Dieter
Fink</a>
 */
```

```
public interface Scheduler {

    void initialize(String info);

    void stop();

}
```

ii. **Créer le bean Singleton qui implémente la minuterie de signleton à utiliser dans tout le cluster.**

```
package
org.jboss.as.quickstarts.cluster.hasingleton.service.ejb;

import javax.annotation.Resource;
import javax.ejb.ScheduleExpression;
import javax.ejb.Singleton;
import javax.ejb.Timeout;
import javax.ejb.Timer;
import javax.ejb.TimerConfig;
import javax.ejb.TimerService;

import org.jboss.logging.Logger;


/**
 * A simple example to demonstrate a implementation of a
cluster-wide singleton timer.
 *
 * @author <a href="mailto:wfink@redhat.com">Wolf-Dieter
Fink</a>
 */
@Singleton
public class SchedulerBean implements Scheduler {
    private static Logger LOGGER =
Logger.getLogger(SchedulerBean.class);
    @Resource
    private TimerService timerService;

    @Timeout
    public void scheduler(Timer timer) {
        LOGGER.info("HASingletonTimer: Info=" +
timer.getInfo());
    }

    @Override
    public void initialize(String info) {
        ScheduleExpression sexpr = new ScheduleExpression();
        // set schedule to every 10 seconds for demonstration
        sexpr.hour("*").minute("*").second("0/10");
        // persistent must be false because the timer is
started by the HASingleton service
        timerService.createCalendarTimer(sexpr, new
TimerConfig(info, false));
    }
```

```
    @Override
    public void stop() {
        LOGGER.info("Stop all existing HASingleton timers");
        for (Timer timer : timerService.getTimers()) {
            LOGGER.trace("Stop HASingleton timer: " +
timer.getInfo());
            timer.cancel();
        }
    }
}
```

2. **Démarrez chaque instance de JBoss EAP 6 avec le clustering activé.**

   Pour activer le clustering dans les serveurs autonomes, démarrer chaque serveur par le profil **HA**, en utilisant un nom de code unique et un offset (décallage) de port pour chaque instance comme suit :

   - Dans Linux, on utilise la commande suivante pour démarrer les serveurs :

     ```
     EAP_HOME/bin/standalone.sh --server-config=standalone-ha.xml -
     Djboss.node.name=UNIQUE_NODE_NAME -Djboss.socket.binding.port-
     offset=PORT_OFFSET
     ```

     Exemple 3.1. Démarrage de multiples serveurs autonomes sur Linux

     ```
     $ EAP_HOME/bin/standalone.sh --server-config=standalone-ha.xml
     -Djboss.node.name=node1
     $ EAP_HOME/bin/standalone.sh --server-config=standalone-ha.xml
     -Djboss.node.name=node2 -Djboss.socket.binding.port-offset=100
     ```

   - Dans Microsoft Windows, on utilise la commande suivante pour démarrer les serveurs :

     ```
     EAP_HOME\bin\standalone.bat --server-config=standalone-ha.xml -
     Djboss.node.name=UNIQUE_NODE_NAME -Djboss.socket.binding.port-
     offset=PORT_OFFSET
     ```

     Exemple 3.2. Démarrage de multiples serveurs autonomes dans Microsoft Windows

     ```
     C:> EAP_HOME\bin\standalone.bat --server-config=standalone-
     ha.xml -Djboss.node.name=node1
     C:> EAP_HOME\bin\standalone.bat --server-config=standalone-
     ha.xml -Djboss.node.name=node2 -Djboss.socket.binding.port-
     offset=100
     ```

NOTE

Si vous ne souhaitez pas utiliser des arguments en ligne de commande, vous pouvez configurer le fichier `standalone-ha.xml` pour que chaque instance de serveur puisse se lier à une interface séparée.

3. **Déployer l'application dans les serveurs**
   Si vous utilisez la commande Maven suivante pour déployer votre application dans un serveur autonome qui exécute sur les ports par défaut.

   ```
   mvn clean install jboss-as:deploy
   ```

   To deploy to additional servers, pass the server name. if it is on a different host, pass the host name and port number on the command line:

   ```
   mvn clean package jboss-as:deploy -Djboss-as.hostname=localhost -
   Djboss-as.port=10099
   ```

   Voir le quickstart **`cluster-ha-singleton`** fourni dans JBoss EAP 6 pour la configuration et les détails de déploiement de Maven.

Rapporter un bogue

3.2.9. Changements dans les déploiement style-service

3.2.9.1. Mise à jour des Applications qui utilisent les Déploiements Style-Service

Résumé

Malgré que JBoss EAP 6 n'utilise plus de descripteurs de style Service, le conteneur prend en charge ces déploiements de style Service sans changement dans la mesure du possible. Cela signifie que si vous avez utilisé des descripteurs de déploiement **`jboss-service.xml`** ou **`jboss-beans.xml`** dans votre application JBoss EAP 5.x, ils devraient fonctionner avec peu ou nulle modification dans JBoss EAP 6. Vous pouvez continuer d'empaqueter les fichiers dans les EAR ou SAR, ou vous pouvez placer les fichiers directement dans le répertoire de déploiement. Si vous exécutez un serveur autonome, le répertoire de déploiement se trouvera ici : ***`EAP_HOME`*`/standalone/deployments/`**. Si vous utilisez un domaine géré, vous devrez utiliser la console ou le CLI pour déployer l'application.

Rapporter un bogue

3.2.10. Changements dans les invocations à distance

3.2.10.1. Migrer des Applications dépoyées dans JBoss EAP 5 qui font des invocations dans JBoss EAP 6

Résumé

Dans JBoss EAP 5, l'interface distante EJB était liée dans JNDI, par défaut, sous le nom "ejbName/local" pour les interfaces locales, et "ejbName/remote" pour les interfaces éloignées. L'application client consulte ensuite le bean qui utilise "ejbName/remote".

Dans JBoss EAP 6, il y a un nouvel API client EJB pour faire l'invocation. Cependant, si vous ne souhaitez pas écrire votre code à nouveau pour pouvoir utiliser le nouvel API, vous pourrez modifier le code existant pour utiliser l'**`ejb:`***BEAN_REFERENCE* d'accès distant aux EJB avec la syntaxe suivante :

Pour les beans stateless, la syntaxe **`ejb:`***BEAN_REFERENCE* est la suivante :

```
ejb:<app-name>/<module-name>/<distinct-name>/<bean-name>!<fully-qualified-
classname-of-the-remote-interface>
```

Pour les beans stateful, la syntaxe **ejb:***BEAN_REFERENCE* est la suivante :

```
ejb:<app-name>/<module-name>/<distinct-name>/<bean-name>!<fully-qualified-classname-of-the-remote-interface>?stateful
```

Les valeurs à substituer dans la syntaxe ci-dessus sont :

- **<app-name>** - le nom de l'application des EJB déployés. C'est généralement le nom de l'ear sans le suffixe .ear. Cependant, le nom peut être substitué dans le fichier application.xml. Si l'application n'est pas déployée comme un .ear, cette valeur correspondra à une chaîne vide. Assumons que cet exemple n'était pas déployé en tant qu'EAR.
- **<module-name>** - le nom du module des EJB déployés sur le serveur. Il s'agit normalement du nom du jar du déploiement EJB, sans le suffixe .jar, mais il peut être remplacé par ejb-jar.xml. Dans cet exemple, on assume que les EJB sont déployés dans jboss-ejb-remote-app.jar, donc le nom du module est jboss-ejb-remote-app.
- **<distinct-name>** - un nom d'EJB distinct, en option. Cet exemple n'utilise pas un nom distinct, mais un string vide.
- **<bean-name>** - correspond par défaut au nom de simple classe d'implémentation du bean.
- **<fully-qualified-classname-of-the-remote-interface>** - le nom de classe complet de la vue distante.

Mise à jour du code client

Assumons que vous avez déployé l'EJB stateless suivant dans un serveur JBoss EAP 6. Notez qu'il expose une vue distante du bean :

```
@Stateless
@Remote(RemoteCalculator.class)
public class CalculatorBean implements RemoteCalculator {

    @Override
    public int add(int a, int b) {
        return a + b;
    }

    @Override
    public int subtract(int a, int b) {
        return a - b;
    }
}
```

Dans JBoss EAP 5, l'invocation et la recherche EJB sont codées de la façon suivante :

```
InitialContext ctx = new InitialContext();
RemoteCalculator calculator = (RemoteCalculator)
ctx.lookup("CalculatorBean/remote");
int a = 204;
int b = 340;
int sum = calculator.add(a, b);
```

Dans JBoss EAP 6, à partir des informations ci-dessus, l'invocation et la recherche sont codées ainsi :

```
final Hashtable jndiProperties = new Hashtable();
jndiProperties.put(Context.URL_PKG_PREFIXES,
"org.jboss.ejb.client.naming");
final Context context = new InitialContext(jndiProperties);
final String appName = "";
final String moduleName = "jboss-ejb-remote-app";
final String distinctName = "";
final String beanName = CalculatorBean.class.getSimpleName();
final String viewClassName = RemoteCalculator.class.getName();
final RemoteCalculator statelessRemoteCalculator =  (RemoteCalculator)
context.lookup("ejb:" + appName + "/" + moduleName + "/" + distinctName +
"/" + beanName + "!" + viewClassName);

int a = 204;
int b = 340;
int sum = statelessRemoteCalculator.add(a, b);
```

Si votre client accède à un EJB stateful, vous devrez ajouter «?stateful» à la fin de la recherche contexte, comme suit :

```
final RemoteCalculator statefulRemoteCalculator =  (RemoteCalculator)
context.lookup("ejb:" + appName + "/" + moduleName + "/" + distinctName +
"/" + beanName + "!" + viewClassName + "?stateful")
```

Pour trouver un exemple, qui comprend à la fois le code client et le code serveur, regarder dans Quickstarts, à *Modules* dans le chapitre Guide de démarrage de Quickstart (*Get Started Developing Applications*) dans le Guide de développement de JBoss EAP 6 (*Development Guide*) à https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/ pour la manière de procéder.

Pour obtenir plus d'informations sur les invocations à distance par JNDI, consulter Section 3.2.10.2, « Invoquer un session bean à distance avec JNDI » .

Rapporter un bogue

3.2.10.2. Invoquer un session bean à distance avec JNDI

Cette tâche décrit comment ajouter un support à un client distant pour l'invocation de session beans par JNDI. La tâche assume que le projet est construit avec Maven.

Le Quickstart **`ejb-remote`** contient des projets Maven qui démontrent cette fonctionnalité. Le Quickstart contient des projets à la fois pour le déploiement des session beans et pour le client distant. Les exemples de code ci-dessous sont extraits du projet du client distant.

Cette tâche assume que les session beans n'ont pas besoin d'authentification.

Pré-requis

Les prérequis suivants doivent être respectés avant de commencer :

- Vous devez déjà avoir un projet Maven créé, prêt à l'utilisation.
- La configuration du référentiel JBoss EAP 6 Maven a déjà été ajoutée.
- Les session beans que vous souhaitez invoquer sont déjà déployés.

- Les session beans déployés implémentent les interfaces commerciales éloignées.
- Les interfaces commerciales éloignées des beans de session sont disponibles sous forme de dépendance de Maven. Si les interfaces commerciales éloignées ne sont disponibles que sous forme de fichier JAR, alors il est recommandé d'ajouter le JAR à votre référentiel Maven comme un artefact. Reportez-vous à la documentation de Maven pour obtenir des directives **`install:install-file`**, http://maven.apache.org/plugins/maven-install-plugin/usage.html
- Vous aurez besoin de connaître le nom d'hôte et le port JNDI du serveur qui hébergent les session beans.

Pour invoquer un session bean d'un client distant, vous devez tout d'abord configurer le projet correctement.

Procédure 3.22. Ajouter une configuration de projet Maven pour l'invocation à distance des session beans

1. **Ajouter les dépendances de projet utiles**
   Le **`pom.xml`** du projet doit être mis à jour pour pouvoir inclure les dépendances nécessaires.

2. **Ajouter le fichier `jboss-ejb-client.properties`**
   L'API du client JBoss EJB s'attend à trouver un fichier dans le root du projet nommé **`jboss-ejb-client.properties`** qui contient les informations de connexion aux services JNDI. Ajouter ce fichier au répertoire **`src/main/resources/`** de votre projet avec le contenu suivant.

   ```
   # In the following line, set SSL_ENABLED to true for SSL
   remote.connectionprovider.create.options.org.xnio.Options.SSL_ENABLE
   D=false
   remote.connections=default
   # Uncomment the following line to set SSL_STARTTLS to true for SSL
   #
   remote.connection.default.connect.options.org.xnio.Options.SSL_START
   TLS=true
   remote.connection.default.host=localhost
   remote.connection.default.port = 4447
   remote.connection.default.connect.options.org.xnio.Options.SASL_POLI
   CY_NOANONYMOUS=false
   # Add any of the following SASL options if required
   #
   remote.connection.default.connect.options.org.xnio.Options.SASL_POLI
   CY_NOANONYMOUS=false
   #
   remote.connection.default.connect.options.org.xnio.Options.SASL_POLI
   CY_NOPLAINTEXT=false
   #
   remote.connection.default.connect.options.org.xnio.Options.SASL_DISA
   LLOWED_MECHANISMS=JBOSS-LOCAL-USER
   ```

   Changer le nom d'hôte et le port pour qu'ils correspondent au serveur. **4447** est le numéro de port par défaut. Pour une connexion sécurisée, définir la ligne **SSL_ENABLED** à **`true`** et décommenter la ligne **SSL_STARTTLS**. L'interface éloignée du conteneur supporte les connexions sécurisées et non sécurisées en utilisant le même port.

3. **Ajouter des dépendances aux interfaces commerciales à distance.**
   Ajouter les dépendances Maven au **`pom.xml`** aux interfaces commerciales éloignées des session beans.

```
<dependency>
   <groupId>org.jboss.as.quickstarts</groupId>
   <artifactId>jboss-ejb-remote-server-side</artifactId>
   <type>ejb-client</type>
   <version>${project.version}</version>
</dependency>
```

Maintenant que le projet a été configuré correctement, vous pouvez ajouter le code pour accéder et invoquer les session beans.

Procédure 3.23. Obtenez un Bean Proxy par JNDI et invoquez les méthodes du Bean

1. **Exceptions vérifiées par Handle**
   Deux des méthodes utilisées dans le code suivant (**`InitialContext()`** et **`lookup()`**) ont une exception vérifiée du type **`javax.naming.NamingException`**. Ces appels de méthode doivent soit être contenus dans un bloc try/catch qui intercepte **`NamingException`** ou dans une méthode déclarée pour lancer **`NamingException`**. Le Quickstart **`ejb-remote`** utilise la seconde technique.

2. **Créer un contexte JNDI**
   Un objet de contexte JNDI fournit le mécanisme pour demander les ressources dans le serveur. Créer un contexte JNDI avec le code suivant :

```
final Hashtable jndiProperties = new Hashtable();
jndiProperties.put(Context.URL_PKG_PREFIXES,
"org.jboss.ejb.client.naming");
final Context context = new InitialContext(jndiProperties);
```

   Les propriétés de connexion du service JNDI sont lues à partir du fichier **`jboss-ejb-client.properties`**.

3. **Utiliser la méthode JNDI Context's lookup() pour obtenir un Bean Proxy**
   Invoquer le méthode **`lookup()`** du bean proxy et lui faire passer le nom JNDI du session bean dont vous avez besoin. Un objet sera renvoyé et il devra correspondre au type de méthode d'interface commerciale qui contient les méthodes que vous souhaitez invoquer.

```
final RemoteCalculator statelessRemoteCalculator =
(RemoteCalculator) context.lookup(
    "ejb:/jboss-ejb-remote-server-side//CalculatorBean!" +
    RemoteCalculator.class.getName());
```

   Les noms de session bean JNDI sont définis par une syntaxe particulière. Pour plus d'informations, consulter Section 3.2.10.3, « Référence de nommage EJB JNDI » .

4. **Méthodes d'invocation**
   Maintenant que vous avez un objet bean proxy, vous pouvez invoquer n'importe quelle méthode qui contient l'interface commerciale distante.

```
int a = 204;
```

```
int b = 340;
System.out.println("Adding " + a + " and " + b + " via the remote
stateless calculator deployed on the server");
int sum = statelessRemoteCalculator.add(a, b);
System.out.println("Remote calculator returned sum = " + sum);
```

Le proxy bean transmet la demande d'invocation de méthode au bean de session sur le serveur, où elle est exécutée. Le résultat est retourné au proxy bean qui, ensuite, le retourne à l'appelant. La communication entre le proxy bean et le bean de session distant est transparente à l'appelant.

Vous devriez maintenant être en mesure de configurer un projet Maven pour soutenir les sessions beans invoquant de session sur un serveur distant et écrire le code d'invocation des méthodes de sessions beans à l'aide d'un proxy bean récupéré du serveur par JNDI.

Rapporter un bogue

3.2.10.3. Référence de nommage EJB JNDI

Le nom de recherche JNDI d'une session bean a la syntaxe suivante :

```
 ejb:<appName>/<moduleName>/<distinctName>/<beanName>!<viewClassName>?
stateful
```

<appName>

Si le fichier JAR du session bean a été déployé dans un EAR, alors ce se sera le nom de l'EAR. Par défaut, le nom d'un EAR correspond à son nom de fichier sans le suffixe **.ear**. Le nom de l'application peut également être remplacé dans son fichier **application.xml**. Si le bean de session n'est pas déployé dans un EAR, laissez le vide.

<moduleName>

Le nom du module correspond au nom du fichier dans lequel le session bean est déployé. Par défaut, le nom du fichier JAR correspond à son nom de fichier sans le suffixe **.jar**. Le nom du module peut également être remplacé par le **ejb-jar.xml** du JAR.

<distinctName>

JBoss EAP 6 permet à chaque déploiement de spécifier un nom distinct en option. Si le déploiement n'a pas de nom distinct, laisser le vide.

<beanName>

Le nom du bean est le nom de classe de la session bean à invoquer.

<viewClassName>

Le nom de classe de vue est le nom de classe complet de l'interface distante de l'interface distante. Inclut le nom du package de l'interface.

?stateful

Le suffixe **?stateful** est requis quand le nom JNDI se réfère à un bean de session stateful. Non inclus dans d'autres types de bean.

Rapporter un bogue

3.2.11. Changements EJB 2.x

3.2.11.1. Mise à jour de l'application qui utilise EJB 2.x

JBoss EAP 6 a été construit sur des standards ouverts et est conforme à la spécification Java Enterprise Edition 6. Malgré que le serveur d'applications fournit un support pour EJB 2.x, il ne prend plus en charge les fonctionnalités qui vont au-delà de la spécification. Gardez à l'esprit que la spécification Java EE 7 a note que EJB 2.x est facultatif, donc il est fortement recommandé que vous réécriviez le code de votre application aux spécifications EJB 3.x.

Si vous souhaitez migrer votre code EJB 2.x, vous devrez, dans la plupart des cas, effectuer des modifications pour exécuter dans JBoss EAP 6. Cette section décrit certains des changements que vous aurez sans doute besoin d'exécuter dans JBoss EAP 6.

Changements de configuration requis pour exécuter EJB 2.x dans JBoss EAP6

Démarrer le serveur avec tous les profils complets

Les beans CMP EJB 2.x (Container Managed Persistence) nécessitent Java Enterprise Edition 6 Full Profile. Ce profil contient des éléments de configuration nécessaires à l'exécution de CMP EJB.

Cette configuration contient le module d'extension de **`org.jboss.as.cmp`** :

```
<extensions>
    ...
    <extension module="org.jboss.as.cmp"/>
    ...
</extensions>
```

Il contient également le sous-système **`cmp`** :

```
<profiles>
    ...
    <subsystem xmlns="urn:jboss:domain:cmp:1.1"/>
    ...
</profiles>
```

.

Pour démarrer un serveur autonome JBoss EAP 6 avec un profil complet, passer l'argument ***`-c standalone-full.xml`*** ou ***`-c standalone-full-ha.xml`*** sur la ligne de commande quand vous démarrez le serveur.

La configuration du conteneur n'est plus prise en charge

Dans les versions précédentes de JBoss EAP, il était possible de configurer différents conteneurs pour l'entité CMP ou les autres beans et de les utiliser en définissant des références dans le fichier de descripteur de déploiement d'application **`jboss.xml`**. Ainsi, il y avait différentes configurations pour SLSB en session beans en général.

In JBoss EAP 6.x, it is possible to use EJB 2 Entity beans with a standard container. However, the different container configurations are no longer supported. The recommended approach is to migrate the EJB2 Stateful Session Beans (SFSB), Stateless Session Beans (SLSB), Message Driven

Beans (MDB) to EJB 3, and for the Container-Managed Persistence (CMP) and Bean-Managed Persistence (BMP) Entity Beans to use the Java Persistence API (JPA) according to the EJB 3 specification.

La configuration de conteneur par défaut de JBoss EAP contient un certain nombre de changements pour les beans EJB 2 CMP :

- Par défaut, le verrouillage pessimistique est actif. Cela peut résulter en blocages.
- Le code de détection de blocages qui se trouvait au niveau CMP dans JBoss EAP 5.x ne se trouve plus dans JBoss EAP 6.

Dans JBoss EAP 5.x, il était également possible de personnaliser la mise en cache, le pooling, les **`commit-options`** et la pile d'intercepteur. Dans JBoss EAP 6, ce n'est plus possible. Il y a qu'une seule implémentation, qui est semblable à la politique d' **`Instance par transaction`** avec **`commit-option C`**. Si vous migrez une application qui utilise la configuration de conteneur de l'entité bean **`cmp2.x jdbc2 pm`**, qui utilise le gestionnaire de perssistance basé JDBC compatible avec CMP2.x, il y aura un impact sur les performances. Ce conteneur a été optimisé pour les performances. Il est recommandé de migrer ces entités dans EJB 3 avant la migration de l'application.

Configuration d'intercepteur côté serveur

JBoss EAP 6 prend en charge l' **`Interceptor`** Java EE standard qui utilise des annotations **`@Interceptors`** et **`@AroundInvoke`**. Cependant, cela ne permet pas les manupulations en dehors de Sécurity ou Transaction.

Dans les versions précédentes de JBoss EAP, il était possible de modifier la pile de l'intercepteur pour avoir des intercepteurs personnalisés pour chaque invocation d'EJB. Cela a été souvent utilisé pour implémenter la sécurité personnalisée ou essayer à nouveau des mécanismes avant les contrôles de sécurité, de contrôles de transaction ou de création. JBoss EAP 6.1 introduit des intercepteurs de conteneur pour fournir des fonctionnalités similaires. Pour plus d'informations sur les intercepteurs de conteneur, voir le chapitre du Guide de développement intitulé *Container Interceptors* dans le *Development Guide* pour JBoss EAP.

Une autre approche procure un contrôle plus soutenu, pendant, ou après la phase de validation d'une transaction tout en respectant la spécification Java EE. Il s'agit de la méthode qui utilise le Registre de synchronisation de transactions pour ajouter un listener.

La ressource peut être extraite par l'une des méthodes suivantes :

- Utiliser **`InitialContext`**

```
TransactionSynchronizationRegistry tsr =
(TransactionSynchronizationRegistry)
  new
InitialContext().lookup("java:jboss/TransactionSynchronizationRegi
stry");
tsr.registerInterposedSynchronization(new MyTxCallback());
```

- Utiliser l'injection

```
@Resource(mappedName =
"java:comp/TransactionSynchronizationRegistry")
TransactionSynchronizationRegistry tsr;
```

```
...
tsr.registerInterposedSynchronization(new MyTxCallback());
```

La routine de callback doit implémenter l'interface **`javax.transaction.Synchronization`**. Utiliser la méthode **`beforeCompletion{}`** pour effectuer une vérification avant que la transaction soit validée ou ou restaurée. Si une exception **`RuntimeException`** est levée par cette méthode, la transaction sera annulée et le client sera informé par une exception **`EJBTransactionRolledbackException`**. Dans le cas d'une Transaction XA, toutes les ressources seront annulées suivant les termes du contrat XA. Il est également possible d'activer la logique métier pour qu'elle dépende de l'état de la transaction, à l'aide de la méthode **`afterCompletion(int txStatus)`**. Si une exception **`RuntimeException`** est levée par cette méthode, la transactions demeurera dans son ancien état, soit validée, soit restaurée, et le client n'en sera pas informé. Seul le gestionnaire de transactions affichera un avertissement dans les fichiers journaux du serveur.

Configuration côté client pour les intercepteurs côté client

Dans les versions précédentes de JBoss EAP 6, il était possible de configurer les intercepteurs de client dans la configuration serveur et de ne fournir que les classes de l'API client.

Dans JBoss EAP 6, ce n'est plus possible parce que le client Proxy n'est plus créé côté serveur et est transmis au client après la recherche. Maintenant, le proxy est généré côté client. Cette optimisation permet d'éviter un appel de serveur pour les téléchargements de recherche et de classe,

Configuration de Bean Pool

La configuration de Bean Pool n'est pas recommandée dans JBoss EAP 6. Comme elle est limitée à la configuration de l'élément **`< stricte-max-pool >`**, des blocages ou autres problèmes peuvent se produire si le pool est trop petit pour charger toutes les entités du résultats. Les beans n'ont pas de grandes méthodes de cycle de vie pendant l'initialisation, donc créer l'instance et le conteneur environnant n'est pas plus lent que d'utiliser une instance d'entité bean regroupée.

Remplacer le fichier de descripteur de déploiement jboss.xml

Le descripteur de déploiement **`jboss-ejb3.xml`** remplace le fichier de descripteur de déploiement **`jboss.xml`**. Ce fichier de remplacement est là pour améliorer les fonctionnalités fournies par le descripteur de déploiement **`ejb-jar.xml`** de Java Enterprise Edition (EE) . Le nouveau fichier est incompatible avec **`jboss.xml`**, et **`jboss.xml`** est maintenant ignoré dans les déploiements.

Ainsi, dans les anciennes versions deJBoss EAP, si vous définissiez un **`<resource-ref>`** dans le fichier **`ejb-jar.xml`**, file, vous aviez besoin d'une définition de ressource correspondante de nom JNDI dans le fichier **`jboss.xml`**. XDoclet génère ces fichiers de descripteur de déploiement automatiquement. Dans JBoss EAP 6, l'information de mappage de JNDI est maintenant définie dans le fichier **`jboss-ejb3.xml`**. Assumez que la source de données soit définie dans le code source Java comme suit :

```
DataSource ds1 = (DataSource) new
InitialContext().lookup("java:comp/env/jdbc/Resource1");
DataSource ds2 = (DataSource) new
InitialContext().lookup("java:comp/env/jdbc/Resource2");
```

Le **`ejb-jar.xml`** définit les références de ressource suivantes :

```
<resource-ref >
```

```
    <res-ref-name>jdbc/Resource1</res-ref-name>
    <res-type>javax.sql.DataSource</res-type>
    <res-auth>Container</res-auth>
</resource-ref>
<resource-ref >
    <res-ref-name>java:comp/env/jdbc/Resource2</res-ref-name>
    <res-type>javax.sql.DataSource</res-type>
    <res-auth>Container</res-auth>
</resource-ref>
```

Le fichier **jboss-ejb3.jxml** mappe les noms JNDI avec les références en utilisant la syntaxe XML suivante.

```
<resource-ref>
    <res-ref-name>jdbc/Resource1</res-ref-name>
    <jndi-name>java:jboss/datasources/ExampleDS</jndi-name>
</resource-ref>
<resource-ref>
    <res-ref-name>java:comp/env/jdbc/Resource2</res-ref-name>
    <jndi-name>java:jboss/datasources/ExampleDS</jndi-name>
</resource-ref>
```

Some of the configuration options that were available in the JBoss EAP 5.x **jboss.xml** file were not implemented in JBoss EAP 6. The following list describes some of the commonly used attributes in the **jboss.xml** file and whether there is an alternate way to achieve them in JBoss EAP 6.

- L'élément **method-attribute** était utilisé pour configurer les méthodes de beans de sessions et des entités individuelles.
  - Les options de configuration **read-only** et **idempotent** n'ont pas été transférées dans JBoss EAP 6.
  - The **transaction-timeout** option is now configured in the **jboss-ejb3.xml** file.
- L'attribut **missing-method-permission-exclude-mode** modifie le comportement des méthodes sans implémenter de métadonnées explicites sur un bean sécurisé. Dans JBoss EAP6, l'abscence d'une annotation **@RolesAllowed** est actuellement traitée de la même façon que **@PermitAll**

Configuration de mappage de type de source de données

Dans les versions précédentes de JBoss EAP 6, il était possible de configurer les mappages de types de sources de données dans le fichier de configuration de déploiement de source de données ***-ds.xml**.

Dans JBoss EAP 6, cela doit se faire dans le fichier de descripteur de déploiement **jbosscmp-jdbc.xml**.

```
<defaults>
    <datasource-mapping>mySQL</datasource-mapping>
    <create-table>true</create-table>
    ....
</defaults>
```

Dans les versions précédentes de JBoss EAP, on pouvait personnaliser le mappage dans le fichier **standardjbosscmp-jdbc.xml**. Ce fichier n'est plus disponible et le mappage est maintenant effectué dans le fichier de descripteur de déploiement **jbosscmp-jdbc.xml**.

Additional Container-Managed Persistence (CMP) and Container-Managed Relationship (CMR) Changes

Changements de Collection et d'Itérateur de CMR (Container Managed Relationship)

Dans les versions précédentes de JBoss EAP, il était possible pour certains conteneurs, comme **cmp2.x jdbc2 pm** d'itérer des collections CMR, de supprimer ou d'ajouter des relations. Parce que la configuration du conteneur n'est pas pris en charge, ce n'est plus possible dans JBoss EAP 6. Pour plus d'informations sur la façon d'obtenir cette même fonctionnalité dans le code d'application, voir EJB2.1 Finder for CMP entities with relations (CMR) returns duplicates in EAP6 dans la section Solutions de base de connaissances de support du Portail clients.

Entrées doubles de CMR (Container Managed Relationship) dans Finders

In previous versions of JBoss EAP, it was possible to select different CMP containers which used different persistence strategies. The **cmp2.x jdbc2 pm** container in JBoss EAP 5.x used optimized **SQL-92** to generate optimized LEFT OUTER JOIN syntax for finders. Because JBoss EAP 6.x only supports the standard container for CMP and CMR, the implementation does not contain these optimizations. The finder should include the keyword **DISTINCT** in the **SELECT** statement to avoid cartesian product in the result set. For more information , see EJB2.1 Finder for CMP entities with relations (CMR) returns duplicates in EAP6 in the Support Knowledgebase Solutions section of the Customer Portal.

Changement de valeur de suppression cascade pour les entity beans CMP

La valeur par défaut de suppression de cascade est passée à **false**. Cela peut entraîner à des erreurs de suppression dans JBoss EAP 6. Si les relations de l'entité sont marquées **cascade-delete**, vous devez définir explicitement le **batch-cascade-delete** à **true** dans le fichier **jbosscmp-jdbc.xml**. Pour plus d'informations, consultez cascade delete fail for EJB2 CMP Entities after migration to EAP6 dans la section Solutions de base de connaissances de Support du Portail clients.

Mappeurs CMP personnalisés pour champs personnalisés

Si vous utilisez des classes de mappage de clients comme **JDBCParameterSetter**, **JDBCResultSetReader** et **Mapper** dans votre application JBoss EAP 5.x, vous verrez sans doute **java.lang.ClassNotFoundException** quand vous déployez votre application dans JBoss EAP 6. C'est parce que les noms de packages des interfaces sont passés de **org.jboss.ejb.plugins.cmp.jdbc.Mapper** à **org.jboss.as.cmp.jdbc.Mapper**. Pour plus d'informations , voir How to use Field mapping for custom classes in an EJB2 CMP application in EAP6 dans la section Solutions de base de connaissance de Support du Portail clients.

Génération de clés primaires par les entity-commands

Si votre application JBoss EAP 5 utilise des **entity-commands** pour générer des clés primaires, comme **Sequence** ou **Auto-increment**, vous apercevrez sans doute une exception **ClassNotFoundException** pour la classe **JDBCOracleSequenceCreateCommand** quand vous migrez votre application dans JBoss EAP 6. C'est parce que le package de classes a été changé de **org.jboss.ejb.plugins.cmp.jdbc** à **org.jboss.as.cmp.jdbc.keygen**. Si vous utilisez cette classe dans votre application JBoss EAP 6, vous devrez aussi ajouter une dépendance au module **EAP_HOME/modules/system/layers/base/org/jboss/as/cmp**.

Changements dans les applications

Modifier le code pour utiliser les nouvelles règles d'espace-noms JNDI.

Comme avec EJB 3.0, vous devrez utiliser le préfixe JNDI complet avec EJB 2.x. Pour davantage d'informations sur les nouvelles règles d'espace-nom JNDI et pour obtenir des exemples de code, voir Section 3.1.8.1, « Mise à jour des noms d'espace-noms JNDI d'application » .

Exemples qui montrent comment mettre à jour les espace-noms JNDI d'anciennes versions : Section 3.1.8.5, « Exemples d'espace noms JNDI de versions antérieures et la façon dont ils sont spécifiés dans JBoss EAP 6 ».

Modifier le descripteur de fichier `jboss-web.xml`

Modifier le **`<jndi-name>`** pour chaque **`<ejb-ref>`** afin d'utiliser le nouveau format de recherche complet JNDI.

Utiliser XDoclet pour mapper un nom JNDI d'interfaces locales internes

Dans EJB 2, il était commun d'utiliser le modèle de **`Locator`** pour chercher les Beans. Si vous utilisiez ce modèle dans votre application au lieu de modifier le code d'application, vous pourriez utiliser XDoclet pour créer une mappe des nouveaux noms JNDI.

Une annotation XDoclet typique ressemble à ceci :

```
@ejb.bean name="UserAttribute" display-name="UserAttribute" local-jndi-
name="ejb21/UserAttributeEntity" view-type="local" type="CMP" cmp-
version="2.x" primkey-field="id"
```

Le nom JNDI **`ejb21/UserAttributeEntity`** de l'exemple ci-dessus n'est plus valide dans JBoss EAP 6. Vous pouvez mapper ce nom à un nom JNDI valide par le sous-système **`naming`** dans la configuration de serveur et grâce à un correctif XDoclet.

Vous pouvez créer des mappeurs personnalisés, comme indiqué dans le paragraphe ci-dessus intitulé *CMP Customized Mappers for Custom Fields*ou bien, vous pouvez modifier le code comme indiqué dans la procédure ci-dessous.

Procédure 3.24. Modifier le code XDoclet généré et Utiliser le sous-système de nommage

1. Extraire le modèle XDoclet **`lookup.xdt`** qui se trouve dans **`ejb-module.jar`** et modifier le **`lookup()`** dans **`lookupHome`** comme suit :

```
private static Object lookupHome(java.util.Hashtable environment,
String jndiName, Class narrowTo) throws
javax.naming.NamingException {
    // Obtain initial context
    javax.naming.InitialContext initialContext = new
javax.naming.InitialContext(environment);
    try {
        // Replace the existing lookup
        // Object objRef = initialContext.lookup(jndiName);
        // This is the new mapped lookup
        Object objRef;
        try {
            // try JBoss EAP mapping
            objRef = initialContext.lookup("global/"+jndiName);
        } catch(java.lang.Exception e) {
```

```
            objRef = initialContext.lookup(jndiName);
        }
        // only narrow if necessary
        if (java.rmi.Remote.class.isAssignableFrom(narrowTo))
            return javax.rmi.PortableRemoteObject.narrow(objRef,
narrowTo);
        else
            return objRef;
    } finally {
        initialContext.close();
    }
}
```

2. Exécuter Ant, en définissant l'attribut de modèle pour qu'il puisse utiliser le **`lookup.xdt`** modifié pour la tâche **`ejbdoclet`**.

3. Modifier le sous-système **`naming`** dans le fichier de configuration du serveur pour mapper l'ancien nom JNDI en nouveau nom JNDI valide.

```
<subsystem xmlns="urn:jboss:domain:naming:1.2">
    <bindings>
        <lookup name="java:global/ejb21/UserAttributeEntity"
lookup="java:global/ejb2CMP/ejb/UserAttribute!de.wfink.ejb21.cmp.c
mr.UserAttributeLocalHome"/>
    </bindings>
    <remote-naming/>
</subsystem>
```

Liste des fichiers obsolètes

Les fichiers suivants ne sont plus pris en charge dans JBoss EAP 6.

jboss.xml

Le fichier de descripteur de déploiement **`jboss.xml`** n'est plus pris en charge et sera ignoré s'il est inclus dans l'archive déployée.

standardjbosscmp-jdbc.xml

Le fichier de configuration **`standardjbosscmp-jdbc.xml`** n'est plus pris en charge. Cette information de configuration est maintenant incluse dans le module **`org.jboss.as.cmp`** et n'est plus personnalisable.

standardjboss.xml

Le fichier de configuration **`standardjboss.xml`** n'est plus pris en charge. Cette information de configuration est maintenant incluse dans le ficher **`standalone.xml`** quand on exécute un serveur autonome ou le fichier **`domain.xml`** dans un domaine géré.

Rapporter un bogue

3.2.12. Changements dans JBoss AOP

3.2.12.1. Mise à jour des applications qui utilisent JBoss AOP

JBoss AOP (Aspect Oriented Programming) n'est plus inclus dans JBoss EAP 6. Dans les versions précédentes, JBoss AOP a été utilisé par le conteneur EJB. Cependant, dans JBoss EAP 6, le conteneur EJB utilise un nouveau mécanisme. Si votre application utilise JBoss AOP, vous devez modifier votre code d'application comme suit.

Refactoriser l'application

- Les configurations standard EJB3 qui étaient auparavant dans le fichier **`ejb3-interceptors-aop.xml`** sont maintenant dans le fichier de configuration de serveur. Pour un serveur autonome, c'est le fichier **`standalone/configuration/standalone-full.xml`**. Si vous exécutez votre serveur dans un domaine géré, il s'agit du fichier **`domain/configuration/domain.xml`**.

- Les intercepteurs AOP côté serveur doivent être modifiés pour pouvoir utiliser l'**`Interceptor`** standard Java EE. Pour plus d'informations sur les intercepteurs de conteneurs et sur la façon d'utiliser un intercepteur côté client dans une application, voir le chapitre intitulé *Container Interceptors* qui se trouve dans le *Development Guide* de JBoss EAP 6 situé dans le Portail clients https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/.

Utiliser les bibliothèques JBoss AOP

- Si vous n'êtes pas en mesure de refactoriser le code, vous pourrez obtenir une copie des bibliothèques AOP JBoss et les regrouper dans l'application. Les bibliothèques AOP peuvent fonctionner dans JBoss EAP 6, mais ne sont pas déployées. Vous pourrez les déployer manuellement à l'aide de l'argument de ligne de commande suivant au moment du démarrage de votre serveur : **`-Djboss.aop.path=`** ***`PATH_TO_AOP_CONFIG`***

NOTE

Malgré le fait que les bibliothèques AOP JBoss puissent fonctionner dans JBoss EAP 6, ce n'est pas une configuration qui est prise en charge.

Rapporter un bogue

3.2.13. Migrer les applications Seam 2.2

3.2.13.1. Migrer les Archives Seam 2.2 dans JBoss EAP 6

Aperçu

Lorsque vous migrez une application Seam 2.2, vous devez configurer la source de données et spécifier toutes les dépendances de module. Vous devez également déterminer si l'application a des dépendances sur les archives qui ne sont pas fournies avec JBoss EAP 6 et copier tout JAR dépendant dans le répertoire de l'application **`lib/`**.

IMPORTANT

Les applications qui utilisent Hibernate directement avec Seam 2.2 utilisent sans doute une version d'Hibernate 3 empaquetée dans l'application. Hibernate 4, fourni par le module org.hibernate de JBoss EAP 6, n'est pas pris en charge par Seam 2.2. Cet exemple a pour but de vous aider à exécuter votre application dans JBoss EAP 6 comme première étape. Notez qu'empaqueter Hibernate 3 avec une application Seam 2.2 n'est pas une configuration prise en charge.

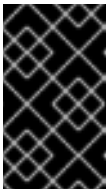
Procédure 3.25. Migrer les Archives Seam 2.2

1. **Mettre à jour la configuration de la source de données**
   Certains exemples Seam 2.2 utilisent la source de données JDBC par défaut nommée **`java:/ExampleDS`**. Cette source de données par défaut a changé dans JBoss EAP 6 en **`java:jboss/datasources/ExampleDS`**. Si votre application utilise la base de donnée de l'exemple, vous pourrez faire ce qui suit :

   - Si vous voulez utiliser la base de données de l'exemple fourni dans JBoss EAP 6, modifier le fichier **`META-INF/persistence.xml`** pour remplacer l'élément **`jta-data-source`** existant par le nom JNDI de la source de données :

     ```
     <!-- <jta-data-source>java:/ExampleDS</jta-data-source> -->
     <jta-data-source>java:jboss/datasources/ExampleDS</jta-data-
     source>
     ```

   - Si vous préférez conserver votre base de données existante, vous pouvez ajouter la définition de votre base de données dans le fichier ***`EAP_HOME`*`/standalone/configuration/standalone.xml`**.

     IMPORTANT

     Vous devez interrompre le serveur avant de modifier le fichier de configuration du serveur pour que votre changement puisse être persisté au redémarrage du serveur.

     Voici une définition qui est une copie de la source de données HSQL par défaut définie dans JBoss EAP 6.

     ```
     <datasource name="ExampleDS" jndi-name="java:/ExampleDS"
     enabled="true" jta="true" use-java-context="true" use-ccm="true">
        <connection-url>jdbc:h2:mem:test;DB_CLOSE_DELAY=-
     1</connection-url>
        <driver>h2</driver>
        <security>
           <user-name>sa</user-name>
           <password>sa</password>
        </security>
     </datasource>
     ```

   - Vous pouvez également ajouter la définition de source de données à l'aide de l'interface de ligne de commande. Voici un exemple de la syntaxe, que vous devez utiliser pour ajouter une source de données. Le « \ » à la fin de la ligne indique la continuation de la commande sur la ligne suivante.

     Exemple 3.3. Exemple de syntaxe à ajouter à la définition de la source de données

```
$ EAP_HOME/bin/jboss-cli --connect
[standalone@localhost:9999 /] data-source add --name=ExampleDS
--jndi-name=java:/ExampleDS \
      --connection-url=jdbc:h2:mem:test;DB_CLOSE_DELAY=-1 --
driver-name=h2 \
      --user-name=sa --password=sa
```

Pour obtenir plus d'informations sur la façon de configurer une source de données, voir Section 3.1.6.2, « Mise à jour de la Configuration de la Source de données » .

2. **Ajouter une dépendance requise**
   Comme les applications Seam 2.2 utilisent JSF 1.2, vous devez ajouter des dépendances aux modules de JSF 1.2 et exclure les modules de JSF 2.0. Pour ce faire, vous devez créer un fichier **`jboss-deployment-structure.xml`** dans le répertoire EAR **`META-INF/`** qui contient les données suivantes :

```
<jboss-deployment-structure xmlns="urn:jboss:deployment-
structure:1.0">
  <deployment>
      <dependencies>
        <module name="javax.faces.api" slot="1.2" export="true"/>
              <module name="com.sun.jsf-impl" slot="1.2"
export="true"/>
      </dependencies>
  </deployment>
  <sub-deployment name="jboss-seam-booking.war">
    <exclusions>
        <module name="javax.faces.api" slot="main"/>
              <module name="com.sun.jsf-impl" slot="main"/>
      </exclusions>
      <dependencies>
        <module name="javax.faces.api" slot="1.2"/>
              <module name="com.sun.jsf-impl" slot="1.2"/>
      </dependencies>
  </sub-deployment>
</jboss-deployment-structure>
```

   Si votre application utilise n'importe quel framework de logging de tiece partie, vous devez ajouter ces dépendances comme décrit ici : Section 3.1.4.1, « Modifier les Dépendances de Logging ».

3. **Si votre application utilise Hibernate 3.x, essayez tout d'abord d'exécuter l'application en utilisant les bibliothèques Hibernate 4**
   Si votre application n'utilise pas le contexte de persistance gérée par Seam, la recherche Hibernate, la validation ou autres éléments qui ont été modifiés dans Hibernate 4, vous pourrez exécuter avec les bibliothèques d'Hibernate 4. Toutefois, si vous voyez **`ClassNotFoundExceptions`** ou **`ClassCastExceptions`** pointant vers des classes Hibernate, ou voir des erreurs similaires à ce qui suit, vous devrez suivre les instructions à l'étape suivante et modifier l'application pour utiliser les bibliothèques d'Hibernate 3.3.

```
        Caused by: java.lang.LinkageError: loader constraint
violation in interface itable initialization: when resolving method
"org.jboss.seam.persistence.HibernateSessionProxy.getSession(Lorg/hi
bernate/EntityMode;)Lorg/hibernate/Session;" the class loader
```

```
(instance of org/jboss/modules/ModuleClassLoader) of the current
class, org/jboss/seam/persistence/HibernateSessionProxy, and the
class loader (instance of org/jboss/modules/ModuleClassLoader) for
interface org/hibernate/Session have different Class objects for the
type org/hibernate/Session used in the signature
```

4. **Copier les archives dépendantes en provenance de frameworks extérieurs ou d'autres locations.**
   Si votre application utilise Hibernate 3.x et que vous n'êtes pas en mesure d'utiliser Hibernate 4 avec succès avec votre application, vous devrez copier les JARs Hibernate 3.x dans le répertoire **`/ lib`** et exclure le module Hibernate de la section de déploiements de **`META-INF/jboss-déploiement-structure.xml`** comme suit :

```
<jboss-deployment-structure xmlns="urn:jboss:deployment-
structure:1.0">
 <deployment>
   <exclusions>
     <module name="org.hibernate"/>
   </exclusions>
 <deployment>
</jboss-deployment-structure>
```

   Il y a des étapes supplémentaires que vous devez prendre pour grouper Hibernate 3.x avec votre application. Pour plus d'informations, consultez Section 3.2.2.2, « Configuration des changements des applications qui utilisent Hibernate et JPA ».

5. **Débogger et résoudre les erreurs de Seam 2.2 JNDI**
   Quand vous migrez une application Seam 2.2, vous apercevez sans doute les erreurs **`javax.naming.NameNotFoundException`** dans le log, sous la forme :

```
javax.naming.NameNotFoundException: Name 'jboss-seam-booking' not
found in context ''
```

   Si vous ne voulez pas modifier les recherches JNDI dans tout le code, vous pouvez modifier les fichiers **`components.xml`** de l'application comme suit :

   a. **Remplacer l'élément core-init existant**
      Tout d'abord, vous devrez remplacer l'élément core-init existant comme suit :

```
<!-- <core:init jndi-pattern="jboss-seam-booking/#
{ejbName}/local" debug="true" distributable="false"/>   -->
<core:init debug="true" distributable="false"/>
```

   b. **Cherchez les messages JNDI binding INFO dans la log du serveur**
      Puis, vous devrez chercher les messages JNDI binding INFO dans la log du serveur quand l'application est déployée. Les messages de liaison JNDI ressemblent à ce qui suit :

```
INFO
org.jboss.as.ejb3.deployment.processors.EjbJndiBindingsDeployment
UnitProcessor (MSC service thread 1-1) JNDI bindings for session
bean
named AuthenticatorAction in deployment unit subdeployment
"jboss-seam-booking.jar" of deployment "jboss-seam-booking.ear"
are as follows:
```

```
    java:global/jboss-seam-booking/jboss-seam-
booking.jar/AuthenticatorAction!org.jboss.seam.example.booking.Au
thenticator
    java:app/jboss-seam-
booking.jar/AuthenticatorAction!org.jboss.seam.example.booking.Au
thenticator

java:module/AuthenticatorAction!org.jboss.seam.example.booking.Au
thenticator
    java:global/jboss-seam-booking/jboss-seam-
booking.jar/AuthenticatorAction
    java:app/jboss-seam-booking.jar/AuthenticatorAction
    java:module/AuthenticatorAction
```

c. **Ajouter des éléments de composants**
Pour chaque message JNDI binding INFO du log, ajouter un élément **component** correspondant au fichier **components.xml** :

```
<component
class="org.jboss.seam.example.booking.AuthenticatorAction" jndi-
name="java:app/jboss-seam-booking.jar/AuthenticatorAction" />
```

Pour en savoir davantage sur la façon de déboguer et résoudre les problèmes de migration, voir Section 4.2.1, « Déboguer et Résoudre les problèmes de migration » .

Pour obtenir une liste des problèmes de migration avec Seam 2, voir Section 3.2.13.2, « Problèmes de Migrations d'archives dans Seam 2.2 ».

Résultat

L'archive Seam 2.2 déploie et exécute successivement sur JBoss EAP 6.

Rapporter un bogue

3.2.13.2. Problèmes de Migrations d'archives dans Seam 2.2

Seam 2.2 et Java 7 ne sont pas compatibles

Seam 2.2 Drools et Java 7 sont incompatibles et l'erreur suivante org.drools.RuntimeDroolsException: value '1.7' n'est pas une erreur valide niveau langage.

Le cglib.jar, signé Seam 2.2.5, empêche l'exemple de Spring de fonctionner

Quand l'exemple de Spring est exécuté avec le **cglib.jar** signé par Seam 2.2.5 de JBoss EAP 5, il échoue pour la cause racine suivante :

```
java.lang.SecurityException: class
"org.jboss.seam.example.spring.UserService$$EnhancerByCGLIB$$7d6c3d12"'s
signer information does not match signer information of other classes in
the same package
```

La solution de contournement pour ce problème est de supprimer la signature du **cglib.jar** comme suit :

```
zip -d $SEAM_DIR/lib/cglib.jar META-INF/JBOSSCOD\*
```

L'exemple Seambay échoue avec `NotLoggedInException`

La raison de ce problème est que l'en-tête du message SOAP est null lors du traitement du message dans SOAPRequestHandler. De ce fait, l'ID de conversation n'est pas défini.

La solution de contournement est de remplacer **`org.jboss.seam.webservice.SOAPRequestHandler.handleOutbound`**, comme décrit dans https://issues.jboss.org/browse/JBPAPP-8376.

L'exemple Seambay échoue dans `UnsupportedOperationException`: no transaction

Ce bogue provient des changements du nom JNDI de la transaction utilisateur dans JBoss EAP 6.

Le solution à ce problème est de remplacer **`org.jboss.seam.transaction.Transaction.getUserTransaction`**, comme décrit dans https://issues.jboss.org/browse/JBPAPP-8322.

L'exemple de tâches lance `org.jboss.resteasy.spi.UnhandledException`: Unable to unmarshall request body

Ce bogue est causé par l'incompatibilité entre le seam-resteasy-2.2.5 inclus dans JBoss EAP 5.1.2) et RESTEasy 2.3.1.GA inclus dans JBoss EAP 6.

La solution de contournement pour ce problème est d'utiliser **`jboss-deployment-structure.xml`** pour exclure les resteasy-jaxrs, resteasy-jettison-provider, et resteasy-jaxb-provider du déploiement principal et les resteasy-jaxrs, resteasy-jettison-provider, resteasy-jaxb-provider, et resteasy-yaml-provider du **`jboss-seam-tasks.war`** comme expliqué dans https://issues.jboss.org/browse/JBPAPP-8315. Il faudra alors inclure les bibliothèques RESTEasy groupées dans Seam 2.2 dans le EAR.

Impasse entre `org.jboss.seam.core.SynchronizationInterceptor` et le verrou EJB de l'instance du composant stateful lors de la requête AJAX.

Erreur de page qui contient le message suivant ou similaire : "Caused by javax.servlet.ServletException with message: "javax.el.ELException: /main.xhtml @36,71 value="#{hotelSearch.pageSize}": org.jboss.seam.core.LockTimeoutException: could not acquire lock on @Synchronized component: hotelSearch".

Le problème est que Seam 2 procède à son propre verrouillage en dehors des SFSB (Stateful Session Bean) et avec une portée différente. Cela signifie que si un thread accède à un EJB deux fois dans la même transaction, après la première invocation, le SFSB sera verrouillé, mais pas le verrou Seam. Un deuxième thread peut alors acquérir le verrou Seam, qui pourra alors atteindre le verrou de l'EJB et attendre. Lorsque le premier thread tente son second appel, il bloque l'intercepteur et l'impasse Seam 2. Dans Java EE 5, les EJB envoient une exception immédiatement en accès simultané. Ce comportement a changé dans Java EE 6.

La solution de comportement est d'ajouter @AccessTimeout(0) à l'EJB. Cela causera l'envoie de l'exception **`ConcurrentAccessException`** immédiatement quand la situation aura lieu.

L'exemple de la commande dvdstore échoue avec l'exception suivante `javax.ejb.EJBTransactionRolledbackException`

L'exemple de la commande dvdstore échoue avec l'exception suivante :

```
JBAS011437: Found extended persistence context in SFSB invocation call
stack but that cannot be used because the transaction already has a
transactional context associated with it. This can be avoided by
```

```
changing application code, either eliminate the extended persistence
context or the transactional context. See JPA spec 2.0 section 7.6.3.1.
```

Le problème est causé par les changements dans la spécification JPA.

La solution à ce problème est de modifier le contexte de persistence à **`transactional`** dans les classes **`CheckoutAction`** et **`ShowOrdersAction`** et d'utiliser l'opération «entity manager merge» des méthodes **`cancelOrder`** et **`detailOrder`**.

Le fournisseur de caches de JBoss Cache ne peut pas être utilisé dans JBoss EAP 6

JBoss Cache n'est pas pris en charge dans EAP 6. Cela entraîne le fournisseur JBoss Cache Seam Cache à échouer avec une application Seam sur le serveur d'applications avec :

```
java.lang.NoClassDefFoundError: org/jboss/util/xml/JBossEntityResolver
```

.

Hibernate 3.3.x Auto scanne les problèmes d'entités JPA dans JBoss EAP 6

La solution à ce problème est de lister toutes les entités du fichier persistence.xml manuellement. Exemple :

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
version="1.0">
    <persistence-unit name="example_pu">
        <description>Hibernate 3 Persistence Unit.</description>
        <jta-data-source>java:jboss/datasources/ExampleDS</jta-data-
source>
        <properties>
            <property name="jboss.as.jpa.providerModule"
value="hibernate3-bundled" />
        </properties>
        <class>com.acme.Foo</class>
        <class>com.acme.Bar</class>
    </persistence-unit>
</persistence>
```

Appeler des composants EJB Seam à partir de Thread non-EJB déclenche l'exception suivante javax.naming.NameNotFoundException

Cette question est le résultat de changements dans JBoss EAP 6 qui servent à mettre en œuvre le nouveau système de chargement de classes modulaires et qui sert à adopter les nouvelles conventions d'espace-noms JNDI normalisés. L'espace-noms **`java:app`** est pour les noms partagés par tous les composants d'une application unique. Les threads non-EE, comme les threads asynchrones de Quartz, doivent utiliser l'espace-noms **`java:global`** qui est partagé par toutes les applications déployées dans une instance de serveur d'applications.

Si vous recevez une **`javax.naming.NameNotFoundException`** quand vous appelez les composants EJB Seam avec des méthodes Quartz, essayez de modifier le fichier **`components.xml`** pour pouvoir utiliser le nom JNDI global, ainsi :

```
<component class="org.jboss.seam.example.quartz.MyBean" jndi-
name="java:global/seam-quartz/quartz-ejb/myBean"/>
```

Four plus d'informations sur les changements JNDI, consultez la section suivante : Section 3.1.8.1, « Mise à jour des noms d'espace-noms JNDI d'application » . Pour plus d'informations sur ce problème particulier, consultez *BZ#948215 - Seam2.3 javax.naming.NameNotFoundException trying to call EJB Seam components from quartz asynchronous methods* dans les notes de sortie *2.2.0 Release Notes* du *Red Hat JBoss Web Framework Kit* qui se trouve sur le Portail clients de Red Hat.

Rapporter un bogue

3.2.14. Migrer les spplications Spring

3.2.14.1. Migrer les Applications Spring

Les informations sur les migrations d'applications Spring se trouvent dans la documentation *Red Hat JBoss Web Framework Kit* que vous pouvez télécharger du Portail clients dans https://access.redhat.com/site/documentation/. Chercher **Red Hat JBoss Middleware**, puis cliquer sur le lien *Red Hat JBoss Web Framework Kit*. Les guides *Spring Installation Guide* et *Spring Developer Guide* sont disponibles dans plusieurs formats.

Rapporter un bogue

3.2.15. Autres changements qui pourraient avoir un impact sur votre migration

3.2.15.1. Familiarisez-vous avec les autres changements qui pourraient avoir un impact sur votre migration

Voici une liste des différences notables entre JBoss EAP 6, et sa dernière version qui pourraient créer un impact sur votre migration.

- Section 3.2.15.2, « Changer le nom du Plug-in Maven »
- Section 3.2.15.3, « Modifier les Applications Client »

Rapporter un bogue

3.2.15.2. Changer le nom du Plug-in Maven

Le **jboss-maven-plugin** n'a pas été mis à jour et ne fonctionne pas dans JBoss EAP 6. Vous devez maintenant utiliser **org.jboss.as.plugins:jboss-as-maven-plugin** pour déployer le répertoire qui convient.

Rapporter un bogue

3.2.15.3. Modifier les Applications Client

Si vous envisagez de migrer une application client qui se connecte à JBoss EAP 6, vous devez être conscient que le nom et l'emplacement du JAR qui regroupe les Bibliothèques Client ont changé. Ce JAR est maintenant nommé **jboss-client.jar** et se trouve dans le répertoire ***JBOSS_HOME*/bin/client/**. Il remplace ***JBOSS_HOME*/client/jbossall-client.jar** et contient toutes les dépendances requises pour se connecter à JBoss EAP 6 à partir du client distant.

Rapporter un bogue

CHAPITRE 4. OUTILS ET ASTUCES

4.1. RESSOURCES POUR ASSISTER À VOTRE MIGRATION

4.1.1. Ressources pour assister à votre migration

Voici une liste des ressources qui pourraient vous aider quand vous faîtes migrer une application vers JBoss EAP 6.

Outils

Il y a plusieurs outils qui peuvent vous aider à automatiser certains changements de configuration. Voir Section 4.1.2, « Familiarisez-vous avec les outils qui pourraient avoir un impact sur votre migration » pour les détails.

Conseils de débogage

Pour obtenir une liste des causes les plus fréquentes et les résolutions des problèmes et erreurs qui peuvent s'afficher lorsque vous migrez votre application, consulter Section 4.2.1, « Déboguer et Résoudre les problèmes de migration ».

Exemples de migration

Pour obtenir des exemples d'applications migrées dans JBoss EAP 6, voir : Section 4.3.1, « Revue de la migration des exemples d'applications ».

Rapporter un bogue

4.1.2. Familiarisez-vous avec les outils qui pourraient avoir un impact sur votre migration

Résumé

Il existe d'autres outils qui peuvent vous assister dans vos efforts de migration. Voici une liste de ces outils, ainsi qu'une description de ce qu'ils peuvent faire.

Tattletale

Avec le changement au niveau du chargement des classes modulaires, vous devez trouver et corriger les dépendances d'applications. Tattletale peut vous aider à identifier les noms de modules dépendants et à générer la configuration XML de votre application.

Section 4.1.3, « Utiliser Tattletale pour trouver des dépendances d'applications »

Outil de migration IronJacamar

Dans JBoss EAP 6, les adaptateurs de ressources et sources de données ne sont plus configurés dans un fichier séparé. Ils sont maintenant définis dans le fichier de configuration et utilisent de nouveaux schémas. L'outil de migration IronJacamar peut vous aider à convertir l'ancienne configuration dans un format qui convient à JBoss EAP 6.

Section 4.1.6, « Utilisation de l'outil IronJacamar pour migrer les Configurations d'aptateurs de ressources et de sources de données »

Rapporter un bogue

4.1.3. Utiliser Tattletale pour trouver des dépendances d'applications

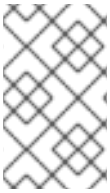
Résumé

Compte tenu des changement au niveau chargement de classes dans JBoss EAP 6, vous verrez sans doute des traces **`ClassNotFoundException`** ou **`ClassCastException`** dans le journal de JBoss quand vous migrez votre application. Pour résoudre ces erreurs, vous aurez besoin de trouver les JAR qui contiennent les classes spécifiées par les exceptions.

Tattletale est un excellent outil de tierce partie qui scanne votre application de manière récursive et qui fournit des rapports détaillés sur son contenu. Tatlletale 1.2.0.Beta2 et versions supérieures contient un support supplémentaire pour le nouveau chargement de classe de JBoss Modules, qui est utilisé dans JBoss EAP 6. Le rapport « JBoss AS 7 » de Tattletale peut être utilisé pour identifier automatiquement et pour générer des noms de modules dépendants à inclure dans votre fichier d'application **`jboss-deployment-structure.xml`**.

Procédure 4.1. Installer et exécuter Tattletale pour trouver des dépendances d'application

1. Section 4.1.4, « Télécharger et installer Tattletale »
2. Section 4.1.5, « Créer et Réviser le Rapport Tattletale »



NOTE

Tattletale est un outil de tierce partie et n'est pas supporté dans JBoss EAP 6. Pour obtenir la documentation la plus récente sur la façon d'installer et d'utiliser Tattletale, consulter le site Tattletale http://www.jboss.org/tattletale.

Rapporter un bogue

4.1.4. Télécharger et installer Tattletale

Procédure 4.2. Télécharger et installer Tattletale

1. Télécharger Tattletale version 1.2.0.Beta2 ou version supérieure à partir de http://sourceforge.net/projects/jboss/files/JBoss%20Tattletale.
2. Décompresser le fichier dans le répertoire de votre choix.
3. Modifier le fichier **`TATTLETALE_HOME/jboss-tattletale.properties`** ainsi :
   a. Ajouter **`ee6`** et **`as7`** à la propriété **`profiles`**.

```
profiles=java5, java6, ee6, as7
```

   b. Dé-commenter les propriétés **`scan`** et **`reports`**.

Rapporter un bogue

4.1.5. Créer et Réviser le Rapport Tattletale

1. Créer le Rapport Tattletale par la commande : **`java -jar`** ***`TATTLETALE_HOME`*****`/tattletale.jar`** ***`APPLICATION_ARCHIVE OUTPUT_DIRECTORY`***

Par exemple : `java -jar tattletale-1.2.0.Beta2/tattletale.jar ~/applications/jboss-seam-booking.ear output-results/`

2. Dans le navigateur, ouvrir le fichier ***`OUTPUT_DIRECTORY`***`/index.html` et cliquer sur la section "JBoss AS 7" sous la section "Reports".

   a. La colonne de gauche liste les archives qui sont utilisées par l'application. Cliquer sur le lien *ARCHIVE_NAME* pour voir des informations sur les archives, comme leurs emplacements, les informations sur le manifeste, et les classes qu'elles contiennent.

   b. Le lien `jboss-deployment-structure.xml` sur la colonne de droite indique comment spécifier la dépendance du module de l'archive nommée sur la colonne de droite. Cliquer dessus pour voir comment définir l'information de module de dépendance de déploiement pour cette archive.

Rapporter un bogue

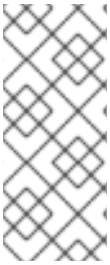
4.1.6. Utilisation de l'outil IronJacamar pour migrer les Configurations d'aptateurs de ressources et de sources de données

Résumé

Dans les versions précédentes du serveur d'applications, les adaptateurs de ressources et de sources de données étaient configurés et déployés à l'aide d'un fichier avec un suffixe de `*-ds.xml`. La distribution IronJacamar 1.1 contient un outil de migration qui peut être utilisé pour convertir ces fichiers de configuration au format attendu par JBoss EAP 6. L'outil analyse le fichier de configuration source de la version précédente, puis crée et écrit la configuration XML dans un fichier de sortie dans le nouveau format. Ce fichier XML peut ensuite être copié et collé sous le sous-système correct dans le fichier de configuration du serveur JBoss EAP 6. Cet outil fait de son mieux pour convertir les anciens attributs et les éléments dans le nouveau format, toutefois, il peut être nécessaire d'apporter des modifications supplémentaires au fichier généré.

Procédure 4.3. Installer et exécuter l'outil de migration IronJacamar

1. Section 4.1.7, « Télécharger et Installer l'outil de migration IronJacamar »
2. Section 4.1.8, « Utiliser l'outil de migration IronJacamar pour convertir un Fichier de configuration de source de données »
3. Section 4.1.9, « Utiliser l'outil de migration IronJacamar pour convertir un Fichier de configuration d'adaptateur de ressources »

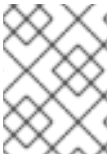


NOTE

L'outil de migration IronJacamar est un outil de tierce partie non pris en charge dans JBoss EAP 6. Pour obtenir plus d'informations sur IronJacamar, visiter http://www.ironjacamar.org/. Pour obtenir la documentation la plus récente sur la façon d'installer et d'utiliser cet outil, consulter http://www.ironjacamar.org/documentation.html.

Rapporter un bogue

4.1.7. Télécharger et Installer l'outil de migration IronJacamar



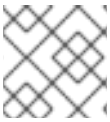
NOTE

L'outil de migration IronJacamar 1.1 n'est disponible que pour les versions 1.1 ou supérieures et requiert Java 7 ou version supérieure.

1. Télécharger la dernière distribution d'IronJacamar à partir du lien suivant : http://www.ironjacamar.org/download.html
2. Décompresser le fichier téléchargé dans un répertoire de votre choix.
3. Trouver le script de conversion dans la distribution IronJacamar
   - Le script Linux se trouve ici : ***IRONJACAMAR_HOME*/doc/as/converter.sh**
   - Le fichier batch Windows se situe à l'adresse suivante : ***IRONJACAMAR_HOME*/doc/as/converter.bat**

Rapporter un bogue

4.1.8. Utiliser l'outil de migration IronJacamar pour convertir un Fichier de configuration de source de données



NOTE

Le script de conversion IronJacamar requiert Java 7 ou version supérieure.

Procédure 4.4. Comment convertir un Fichier de configuration de source de données

1. Ouvrir une ligne de commande et naviguez vers le répertoire ***IRONJACAMAR_HOME*/doc/as/**.
2. Exécuter le script de conversion en saisissant la commande suivante :
   - Dans Linux : **./converter.sh -ds *SOURCE_FILE TARGET_FILE***
   - Dans Microsoft Windows : **./converter.bat -ds *SOURCE_FILE TARGET_FILE***

   Le fichier ***SOURCE_FILE*** est le fichier datasource -ds.xml en provenance de la version précédente. Le fichier ***TARGET_FILE*** contient la nouvelle configuration.

   Ainsi, pour convertir le fichier de configuration d'adaptateur de ressources **jboss-seam-booking-ds.xml** qui se trouve dans le répertoire en cours, vous saisissez :
   - Dans Linux: **./converter.sh -ds jboss-seam-booking-ds.xml new-datasource-config.xml**
   - Dans Microsoft Windows: **./converter.bat -ds jboss-seam-booking-ds.xml new-datasource-config.xml**

   Notez que le paramètre de conversion de source de données est **-ds**.
3. Copier l'élément **<datasource>** à partir du fichier cible et coller le dans le fichier de configuration du serveur sous l'élément **<subsystem xmlns="urn:jboss:domain:datasources:1.1"><datasources>**.

IMPORTANT

Vous devez interrompre le serveur avant de modifier le fichier de configuration du serveur pour que votre changement puisse être persisté au redémarrage du serveur.

- Si vous exécutez dans un domaine géré, copier l'XML dans le fichier ***EAP_HOME*/domain/configuration/domain.xml**.
- Si vous exécutez dans un serveur autonome, copier l'XML dans le fichier ***EAP_HOME*/standalone/configuration/standalone.xml**.

4. Modifier l'XML créé dans le nouveau fichier de configuration.

   Voici un exemple du fichier de configuration de source de données **jboss-seam-booking-ds.xml** pour l'exemple Seam 2.2 Booking fourni dans JBoss EAP 5.x:

```
<?xml version="1.0" encoding="UTF-8"?>
<datasources>
  <local-tx-datasource>
    <jndi-name>bookingDatasource</jndi-name>
    <connection-url>jdbc:hsqldb:.</connection-url>
    <driver-class>org.hsqldb.jdbcDriver</driver-class>
    <user-name>sa</user-name>
    <password></password>
  </local-tx-datasource>
</datasources>
```

   Voici le fichier de configuration qui a été généré en exécutant le script de convertisseur. Le fichier généré contient un élément **<driver-class>**. La meilleure façon de définir la classe de pilote dans JBoss EAP 6 est par un élément **<driver>**. Voici l'XML qui en résulte dans le fichier de configuration JBoss EAP 6 avec des modifications pour décommenter l'élément **<driver-class>** et ajouter l'élément **<driver>** correspondant :

```
<subsystem xmlns="urn:jboss:domain:datasources:1.1">
  <datasources>
    <datasource enabled="true" jndi-
name="java:jboss/datasources/bookingDatasource" jta="true"
            pool-name="bookingDatasource" use-ccm="true" use-java-
context="true">
      <connection-url>jdbc:hsqldb:.</connection-url>
      <!-- Comment out the following driver-class element
           since it is not the preferred way to define this.
           <driver-class>org.hsqldb.jdbcDriver</driver-class>     -
->
      <!-- Specify the driver, which is defined later in the
datasource  -->
      <driver>h2<driver>
      <transaction-isolation>TRANSACTION_NONE</transaction-
isolation>
      <pool>
        <prefill>false</prefill>
        <use-strict-min>false</use-strict-min>
        <flush-strategy>FailingConnectionOnly</flush-strategy>
      </pool>
```

```
        <security>
          <user-name>sa</user-name>
          <password/>
        </security>
        <validation>
          <validate-on-match>false</validate-on-match>
          <background-validation>false</background-validation>
          <use-fast-fail>false</use-fast-fail>
        </validation>
        <timeout/>
        <statement>
          <track-statements>false</track-statements>
        </statement>
      </datasource>
      <drivers>
        <!-- The following driver element was not in the
       XML target file. It was created manually. -->
        <driver name="h2" module="com.h2database.h2">
          <xa-datasource-class>org.h2.jdbcx.JdbcDataSource</xa-
datasource-class>
        </driver>
      </drivers>
    </datasources>
</subsystem>
```

Rapporter un bogue

4.1.9. Utiliser l'outil de migration IronJacamar pour convertir un Fichier de configuration d'adaptateur de ressources

NOTE

Le script de conversion IronJacamar requiert Java 7 ou version supérieure.

1. Ouvrir une ligne de commande et naviguer vers le répertoire ***IRONJACAMAR_HOME*/docs/as/**.
2. Exécuter le script de conversion en saisissant la commande suivante :
   - Dans Linux: **./converter.sh -ra *SOURCE_FILE TARGET_FILE***
   - Dans Microsoft Windows: **./converter.bat -ra *SOURCE_FILE TARGET_FILE***

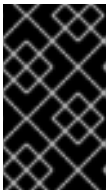
   Le fichier ***SOURCE_FILE*** est le fichier d'adaptateur de ressource -ds.xml en provenance de la version précédente. Le fichier ***TARGET_FILE*** contient la nouvelle configuration.

   Ainsi, pour convertir le fichier de configuration d'adaptateur de ressources **mttestadapter-ds.xml** qui se trouve dans le répertoire en cours, vous saisissez :

   - Dans Linux: **./converter.sh -ra mttestadapter-ds.xml new-adapter-config.xml**
   - Dans Microsoft Windows: **./converter.bat -ra mttestadapter-ds.xml new-adapter-config.xml**

Notez que le paramètre de conversion d'adaptateur de ressources est **`-ra`**.

3. Copier l'élément **`<resource-adapters>`** à partir du fichier cible et coller le dans le fichier de configuration du serveur sous l'élément **`<subsystem xmlns="urn:jboss:domain:resource-adapters:1.1">`**.



IMPORTANT

Vous devez interrompre le serveur avant de modifier le fichier de configuration du serveur pour que votre changement puisse être persisté au redémarrage du serveur.

- Si vous exécutez dans un domaine géré, copier l'XML dans le fichier ***`EAP_HOME`*`/domain/configuration/domain.xml`**.
- Si vous exécutez dans un serveur autonome, copier l'XML dans le fichier ***`EAP_HOME`*`/standalone/configuration/standalone.xml`**.

4. Modifier l'XML créé dans le nouveau fichier de configuration.

Voici un exemple de fichier de configuration d'adaptateur de ressources **`mttestadapter-ds.xml`** de JBoss EAP 5.x TestSuite:

```
<?xml version="1.0" encoding="UTF-8"?>
  <!--
=====================================================================
-->
  <!-- ConnectionManager setup for jboss test adapter
-->
  <!-- Build jmx-api (build/build.sh all) and view for config
documentation -->
  <!--
=====================================================================
-->
<connection-factories>
  <tx-connection-factory>
    <jndi-name>JBossTestCF</jndi-name>
    <xa-transaction/>
    <rar-name>jbosstestadapter.rar</rar-name>
    <connection-
definition>javax.resource.cci.ConnectionFactory</connection-
definition>
    <config-property name="IntegerProperty"
type="java.lang.Integer">2</config-property>
    <config-property name="BooleanProperty"
type="java.lang.Boolean">false</config-property>
    <config-property name="DoubleProperty"
type="java.lang.Double">5.5</config-property>
    <config-property name="UrlProperty"
type="java.net.URL">http://www.jboss.org</config-property>
    <config-property name="sleepInStart" type="long">200</config-
property>
    <config-property name="sleepInStop" type="long">200</config-
property>
  </tx-connection-factory>
```

```
  <tx-connection-factory>
    <jndi-name>JBossTestCF2</jndi-name>
    <xa-transaction/>
    <rar-name>jbosstestadapter.rar</rar-name>
    <connection-
definition>javax.resource.cci.ConnectionFactory</connection-
definition>
    <config-property name="IntegerProperty"
type="java.lang.Integer">2</config-property>
    <config-property name="BooleanProperty"
type="java.lang.Boolean">false</config-property>
    <config-property name="DoubleProperty"
type="java.lang.Double">5.5</config-property>
    <config-property name="UrlProperty"
type="java.net.URL">http://www.jboss.org</config-property>
    <config-property name="sleepInStart" type="long">200</config-
property>
    <config-property name="sleepInStop" type="long">200</config-
property>
  </tx-connection-factory>
  <tx-connection-factory>
    <jndi-name>JBossTestCFByTx</jndi-name>
    <xa-transaction/>
    <track-connection-by-tx>true</track-connection-by-tx>
    <rar-name>jbosstestadapter.rar</rar-name>
    <connection-
definition>javax.resource.cci.ConnectionFactory</connection-
definition>
    <config-property name="IntegerProperty"
type="java.lang.Integer">2</config-property>
    <config-property name="BooleanProperty"
type="java.lang.Boolean">false</config-property>
    <config-property name="DoubleProperty"
type="java.lang.Double">5.5</config-property>
    <config-property name="UrlProperty"
type="java.net.URL">http://www.jboss.org</config-property>
    <config-property name="sleepInStart" type="long">200</config-
property>
    <config-property name="sleepInStop" type="long">200</config-
property>
  </tx-connection-factory>
</connection-factories>
```

Voici le fichier de configuration qui a été généré en exécutant le script de convertisseur. Remplacez la valeur de l'attribut de nom de classe « FIXME_MCF_CLASS_NAME » dans le code XML généré par le nom correct de la fabrique de connexions, dans ce cas, « org.jboss.test.jca.adapter.TestManagedConnectionFactory ». Voici le document XML obtenu dans le fichier de configuration de JBoss EAP 6 sous réserve de modifications de la valeur de l'élément **<class-name>** :

```
<subsystem xmlns="urn:jboss:domain:resource-adapters:1.1">
  <resource-adapters>
    <resource-adapter>
      <archive>jbosstestadapter.rar</archive>
      <transaction-support>XATransaction</transaction-support>
      <connection-definitions>
```

```
  <!-- Replace the "FIXME_MCF_CLASS_NAME" class-name value with the
correct class name
  <connection-definition class-name="FIXME_MCF_CLASS_NAME"
enabled="true"
    jndi-name="java:jboss/JBossTestCF" pool-name="JBossTestCF"
    use-ccm="true" use-java-context="true"> -->
  <connection-definition
    class-
name="org.jboss.test.jca.adapter.TestManagedConnectionFactory"
    enabled="true"
    jndi-name="java:jboss/JBossTestCF" pool-name="JBossTestCF"
    use-ccm="true" use-java-context="true">
    <config-property name="IntegerProperty">2</config-property>
    <config-property name="sleepInStart">200</config-property>
    <config-property name="sleepInStop">200</config-property>
    <config-property name="BooleanProperty">false</config-property>
    <config-property
name="UrlProperty">http://www.jboss.org</config-property>
    <config-property name="DoubleProperty">5.5</config-property>
    <pool>
      <prefill>false</prefill>
      <use-strict-min>false</use-strict-min>
      <flush-strategy>FailingConnectionOnly</flush-strategy>
    </pool>
    <security>
      <application/>
    </security>
    <timeout/>
    <validation>
      <background-validation>false</background-validation>
      <use-fast-fail>false</use-fast-fail>
    </validation>
  </connection-definition>
      </connection-definitions>
    </resource-adapter>
    <resource-adapter>
      <archive>jbosstestadapter.rar</archive>
      <transaction-support>XATransaction</transaction-support>
      <connection-definitions>
  <!-- Replace the "FIXME_MCF_CLASS_NAME" class-name value with the
correct class name
   <connection-definition class-name="FIXME_MCF_CLASS_NAME"
enabled="true"
    jndi-name="java:jboss/JBossTestCF2" pool-name="JBossTestCF2"
    use-ccm="true" use-java-context="true"> -->
  <connection-definition
    class-
name="org.jboss.test.jca.adapter.TestManagedConnectionFactory"
    enabled="true"
    jndi-name="java:jboss/JBossTestCF2" pool-name="JBossTestCF2"
    use-ccm="true" use-java-context="true">
    <config-property name="IntegerProperty">2</config-property>
    <config-property name="sleepInStart">200</config-property>
    <config-property name="sleepInStop">200</config-property>
    <config-property name="BooleanProperty">false</config-property>
    <config-property
```

```
name="UrlProperty">http://www.jboss.org</config-property>
    <config-property name="DoubleProperty">5.5</config-property>
    <pool>
      <prefill>false</prefill>
      <use-strict-min>false</use-strict-min>
      <flush-strategy>FailingConnectionOnly</flush-strategy>
    </pool>
    <security>
      <application/>
    </security>
    <timeout/>
    <validation>
      <background-validation>false</background-validation>
      <use-fast-fail>false</use-fast-fail>
    </validation>
  </connection-definition>
      </connection-definitions>
    </resource-adapter>
    <resource-adapter>
      <archive>jbosstestadapter.rar</archive>
      <transaction-support>XATransaction</transaction-support>
      <connection-definitions>
  <!-- Replace the "FIXME_MCF_CLASS_NAME" class-name value with the
correct class name
  <connection-definition class-name="FIXME_MCF_CLASS_NAME"
enabled="true"
     jndi-name="java:jboss/JBossTestCFByTx" pool-
name="JBossTestCFByTx"
     use-ccm="true" use-java-context="true"> -->
  <connection-definition
    class-
name="org.jboss.test.jca.adapter.TestManagedConnectionFactory"
    enabled="true"
    jndi-name="java:jboss/JBossTestCFByTx" pool-
name="JBossTestCFByTx"
    use-ccm="true" use-java-context="true">
    <config-property name="IntegerProperty">2</config-property>
    <config-property name="sleepInStart">200</config-property>
    <config-property name="sleepInStop">200</config-property>
    <config-property name="BooleanProperty">false</config-property>
    <config-property
name="UrlProperty">http://www.jboss.org</config-property>
    <config-property name="DoubleProperty">5.5</config-property>
    <pool>
      <prefill>false</prefill>
      <use-strict-min>false</use-strict-min>
      <flush-strategy>FailingConnectionOnly</flush-strategy>
    </pool>
    <security>
      <application/>
    </security>
    <timeout/>
    <validation>
      <background-validation>false</background-validation>
      <use-fast-fail>false</use-fast-fail>
    </validation>
```

```
    </connection-definition>
        </connection-definitions>
      </resource-adapter>
    </resource-adapters>
  </subsystem>
```

Rapporter un bogue

4.2. DÉBOGUER LES PROBLÈMES DE MIGRATION

4.2.1. Déboguer et Résoudre les problèmes de migration

À cause du chargement de classes, les règles de nommage de JNDI et d'autres changements dans le serveur d'application, vous pouvez rencontrer des exceptions ou autres erreurs si vous tentez de déployer votre application «telle qu'elle». Ce qui suit décrit comment résoudre certaines exceptions des plus communes ou erreurs que vous pouvez rencontrer.

- Section 4.2.2, « Déboguer et Résoudre ClassNotFoundExceptions et NoClassDefFoundErrors »
- Section 4.2.5, « Déboguer et Résoudre les problèmes de migration »
- Section 4.2.6, « Déboguer et Résoudre les DuplicateServiceExceptions »
- Section 4.2.7, « Déboguer et résoudre les erreurs de débogage de JBoss Seam Debug »

Rapporter un bogue

4.2.2. Déboguer et Résoudre ClassNotFoundExceptions et NoClassDefFoundErrors

Résumé

ClassNotFoundExceptions a lieu habituellement lorsqu'une dépendance n'est pas résolue. Cela signifie que vous devez explicitement

1. Essayer tout d'abord de trouver la dépendance manquante. Cela est décrit en détails dans Section 4.2.3, « Chercher la dépendance de module de JBoss »
2. S'il n'y a pas de module pour la classe manquante, chercher le JAR dans l'installation précédente. Pour plus d'informations, voir Section 4.2.4, « Trouver le JAR dans l'installation précédente »

Rapporter un bogue

4.2.3. Chercher la dépendance de module de JBoss

Pour résoudre la dépendance, essayez tout d'abord de trouver le module qui contient la classe spécifiée par **`ClassNotFoundException`** en cherchant dans le répertoire ***`EAP_HOME`*`/modules/system/layers/base/`**. Si vous trouvez un module pour la classe, vous devrez ajouter une dépendance à l'entrée du manifeste.

Par exemple, si vous apercevez la trace ClassNotFoundException dans le log :

```
Caused by: java.lang.ClassNotFoundException:
org.apache.commons.logging.Log
    from [Module "deployment.TopicIndex.war:main" from Service Module
```

```
Loader]
    at
org.jboss.modules.ModuleClassLoader.findClass(ModuleClassLoader.java:188)
```

Chercher le module JBoss qui contient cette classe ainsi :

Procédure 4.5. Chercher la dépendance

1. Déterminer tout d'abord s'il y a un module évident pour la classe.

   a. Naviguer dans le répertoire ***EAP_HOME*/modules/system/layers/base/** et chercher la classe qui correspond au chemin du module dans **ClassNotFoundException**.

   Vous trouverez le chemin d'accès **org/apache/commons/logging/**.

   b. Ouvrir le fichier ***EAP_HOME*/modules/system/layers/base/org/apache/commons/logging/main/module.xml** et chercher le nom du module. Dans ce cas, ce sera "org.apache.commons.logging".

   c. Ajouter le nom du module aux Dépendances dans le fichier **MANIFEST.MF** :

   ```
   Manifest-Version: 1.0
   Dependencies: org.apache.commons.logging
   ```

2. S'il n'y a pas de chemin de classe évident pour la classe, vous devrez trouver la dépendance dans un autre emplacement.

   a. Trouver la classe nommée par l'exception **ClassNotFoundException** dans le rapport Tattletale.

   b. Trouver le module qui contient le JAR dans le répertoire ***EAP_HOME*/modules** et trouver le nom du module comme dans la première étape.

Rapporter un bogue

4.2.4. Trouver le JAR dans l'installation précédente

Si vous ne trouvez pas la classe dans un JAR empaqueté dans un module défini par le serveur, chercher le JAR dans votre installation *EAP5_HOME* ou dans le répertoire **lib/** de votre serveur précédent.

Par exemple, si vous apercevez la trace **ClassNotFoundException** dans le log :

```
Causé par : java.lang.NoClassDefFoundError:
org/hibernate/validator/ClassValidator à
java.lang.Class.getDeclaredMethods0(Native Method)
```

Chercher le JAR qui contient cette classe ainsi :

1. Ouvrir un terminal et naviguer dans le répertoire ***EAP5_HOME/***.

2. Lancer la commande :

   grep 'org.hibernate.validator.ClassValidator' `find . \-name '*.jar'`

3. Vous risquez d'apercevoir plus d'un résultat. Dans ce cas, nous avons besoin du résultat de JAR suivant :

```
Le fichier binaire ./jboss-eap-5.1/seam/lib/hibernate-validator.jar
correspond
```

4. Copier ce JAR dans le répertoire **`lib/`** de l'application.

   Si vous pensez que vous avez besoin d'un grand nombre de JARS, il est sans doute plus facile de définir un module pour les classes. Voir *Modules* dans le chapitre Guide de démarrage des application de développement (*Get Started Developing Applications*) dans le Guide de développement de JBoss EAP 6 (*Development Guide*) à https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/.

5. Construire et déployer l'application à nouveau.

Rapporter un bogue

4.2.5. Déboguer et Résoudre les problèmes de migration

Les ClassCastExceptions apparaissent souvent quand une classe est chargée par un chargeur de classes qui diffère de la classe qu'il étend. Souvent le résultat d'une même classe qui peut exister dans plusieurs JAR.

1. Chercher dans l'application pour trouver tous les JAR qui contiennent la classe nommée par l'exception **`ClassCastException`**. S'il y a un module défini pour la classe, chercher et supprimer les JAR en double des WAR et EAR de l'application.

2. Chercher le module JBoss qui contient la classe et qui définit explicitement la dépendance dans le fichier **`MANIFEST.MF`** ou dans le fichier **`jboss-deployment-structure.xml`**. Pour plus d'informations sur le chargement de classes, voir Chargement de classes et Sous-déploiements (*Class Loading and Subdeployments*) dans le chapitre *Class Loading and Modules*du *Development Guide* pour JBoss EAP 6 https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/.

3. Si vous ne pouvez le résoudre par les étapes ci-dessus, vous pourrez souvent déterminer la cause du problème en imprimant les informations de chargement de classe dans le log. Par exemple, vous pourrez voir l'exception **`ClassCastException`** dans le log :

```
java.lang.ClassCastException: com.example1.CustomClass1 cannot be
cast to com.example2.CustomClass2
```

   a. Dans votre code, imprimer les informations de chargement de classe des classes nommées par **`ClassCastException`** dans le log, comme suit :

```
logger.info("Class loader for CustomClass1: " +

com.example1.CustomClass1.getClass().getClassLoader().toString())
;
logger.info("Class loader for CustomClass2: " +

com.example2.CustomClass2.getClass().getClassLoader().toString())
;
```

b. L'information qui se trouve dans le log montre les modules qui correspondent à des classes de chargement et, selon votre application, vous aurez besoin de retirer ou de déplacer les JAR qui entrent en conflit.

Rapporter un bogue

4.2.6. Déboguer et Résoudre les DuplicateServiceExceptions

Si vous obtenez un DuplicateServiceException pour le sous-déploiement d'un JAR ou un message que l'application WAR a déjà été installé lorsque vous déployez votre EAR dans JBoss EAP 6, cela peut être dû aux changements dans la façon dont JBossWS gère le déploiement.

La version JBossWS 3.3.0 introduit un nouvel algorithme de mappage de racine contextuel pour les points de terminaison basés servlet en vue de faciliter la compatibilité avec TCK6. Si l'archive EAR d'application contient un WAR et un JAR du même nom, JBossWS pourra créer un contexte WAR et web portant le même nom. Le contexte web entre en conflit avec le contexte WAR et cela entraîne des erreurs de déploiement. Résoudre les problèmes de déploiement d'une des manières suivantes :

- Renommer le fichier JAR qui est différent du WAR, pour que les contextes générés Web et WAR soient uniques.
- Fournir un élément **`<context-root>`** dans le fichier **`jboss-web.xml`**.
- Fournir un élément **`<context-root>`** dans le fichier **`jboss-webservices.xml`**.
- Personnaliser l'élément **`<context-root>`** du WAR dans le fichier **`application.xml`**.

Rapporter un bogue

4.2.7. Déboguer et résoudre les erreurs de débogage de JBoss Seam Debug

Une fois que vous aurez migré et déployé avec succès votre application, vous risquez de rencontrer une erreur de runtime qui vous redirige vers la page de "JBoss Seam Debug". L'url de cette page est "http://localhost:8080/*APPLICATION_CONTEXT*/debug.seam". Cette page vous permettra de voir et d'inspecter les composants Seam dans n'importe quel contexte Seam associé à votre session de login en cours.

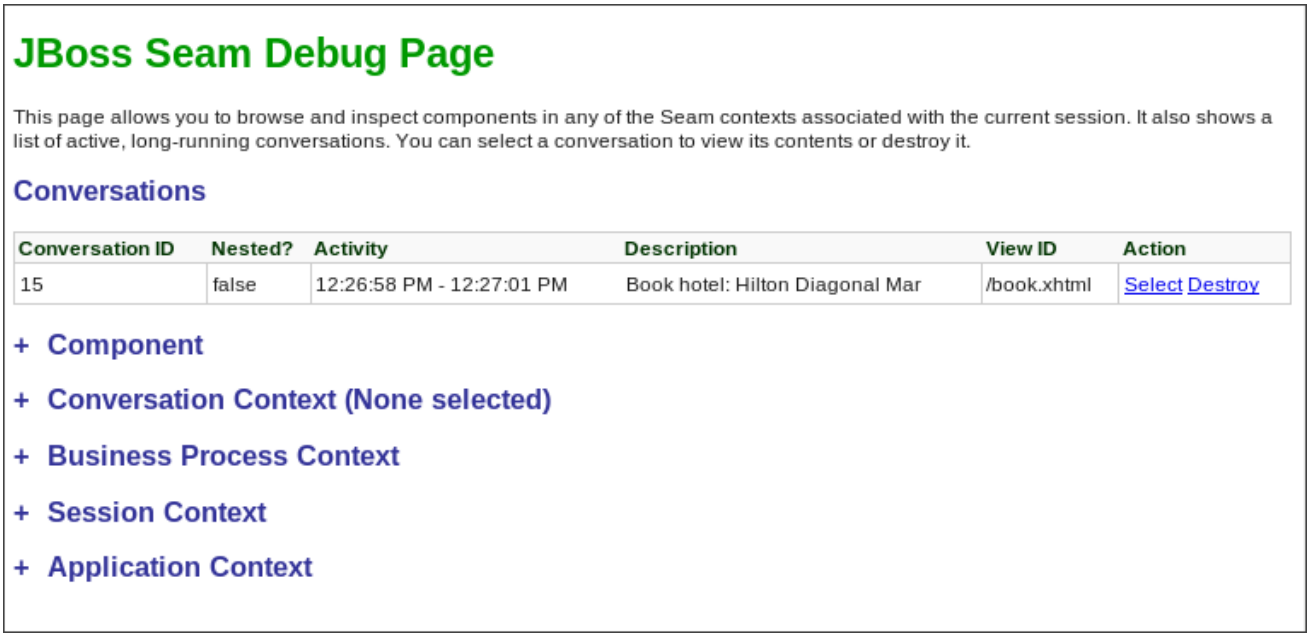


Figure 4.1. JBoss Seam Debug Page

La raison la plus probable pour laquelle vous vous trouvez redirigé sur cette page Seam est parce que Seam a intercepté une exception non prise en considération dans le code d'application. On peut trouver la cause de l'origine de l'exception dans un des liens qui se trouvent sur la page "JBoss Seam Debug Page".

1. Étendre la section **`Component`** sur la page, et chercher le composant **`org.jboss.seam.caughtException`**.

2. La cause et le stacktrace devraient vous pointer vers les dépendances manquantes.

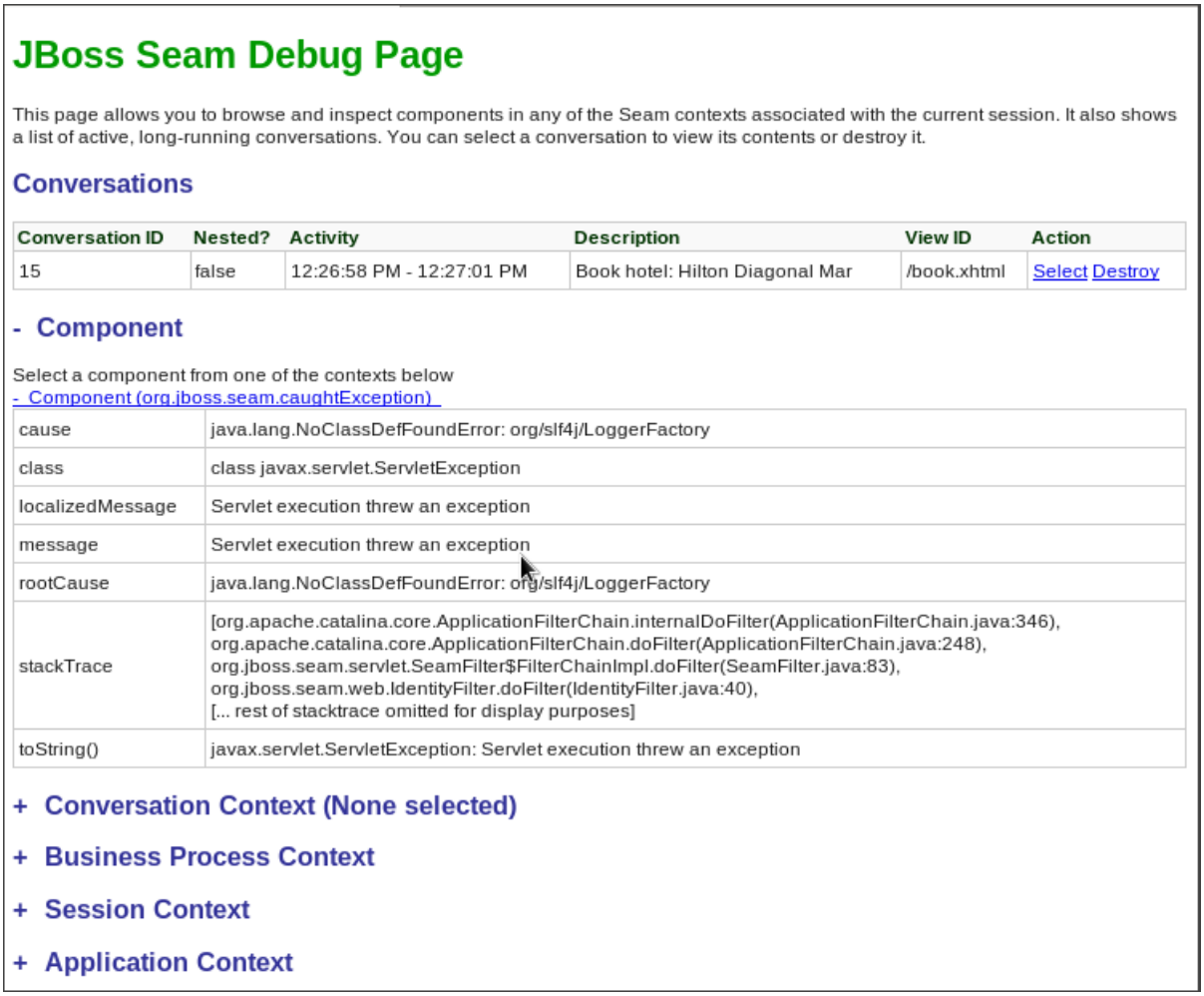


Figure 4.2. Informations sur le composant `org.jboss.seam.caughtException` information

3. Utiliser la technique décrite dans Section 4.2.2, « Déboguer et Résoudre ClassNotFoundExceptions et NoClassDefFoundErrors » pour résoudre les dépendances du module.

   Dans l'exemple ci-dessus, la solution la plus simple est d'ajouter `org.slf4j` à `MANIFEST.MF`

   ```
   Manifest-Version: 1.0
   Dependencies: org.slf4j
   ```

   Une autre option consiste à ajouter une dépendance au module dans le fichier `jboss-deployment-structure.xml`:

   ```
   <jboss-deployment-structure>
      <deployment>
           <dependencies>
             <module name="org.slf4j" />
           </dependencies>
       </deployment>
   </jboss-deployment-structure>
   ```

Rapporter un bogue

4.3. REVUE DE LA MIGRATION DES EXEMPLES D'APPLICATIONS

4.3.1. Revue de la migration des exemples d'applications

Aperçu

Ce qui suit est une liste d'exemples d'applications JBoss EAP 5.x qui ont été migrées dans JBoss EAP 6. Pour voir les détails des changements dans une application en particulier, cliquer sur le lien ci-dessous.

- Section 4.3.2, « Migrer l'exemple Seam 2.2 dans JBoss EAP 6 »
- Section 4.3.3, « Migrer l'exemple de Seam 2.2 Booking dans JBoss EAP 6 »
- Section 4.3.4, « Migrer l'archive du Seam 2.2 Booking dans JBoss EAP 6: Instructions étape par étape »

Rapporter un bogue

4.3.2. Migrer l'exemple Seam 2.2 dans JBoss EAP 6

Résumé

La liste de tâches suivantes récapitule les changements nécessaires pour pouvoir migrer l'exemple d'application Seam 2.2 JPA dans JBoss EAP 6. Cet exemple d'application se trouve dans la dernière distribution JBoss EAP dans ***EAP5.x_HOME*/jboss-eap-5/seam/examples/jpa/**

IMPORTANT

Les applications qui utilisent Hibernate directement avec Seam 2.2 utilisent sans doute une version d'Hibernate 3 empaquetée dans l'application. Hibernate 4, fourni par le module org.hibernate de JBoss EAP 6, n'est pas pris en charge par Seam 2.2. Cet exemple a pour but de vous aider à exécuter votre application dans JBoss EAP 6 comme première étape. Notez qu'empaqueter Hibernate 3 avec une application Seam 2.2 n'est pas une configuration prise en charge.

Procédure 4.6. Migration de l'exemple Seam 2.2 JPA Booking

1. **Retirer le fichier jboss-web.xml**
   Retirer le fichier **jboss-web.xml** du répertoire **jboss-seam-jpa.war/WEB-INF/**. Le chargement de classe défini dans **jboss-web.xml** est maintenant le comportement de chargement de classe par défaut.

2. **Modifier le fichier jboss-seam-jpa.jar/META-INF/persistence.xml comme suit.**

   a. Supprimer et dé-commenter la propriété **hibernate.cache.provider_class** du fichier **jboss-seam-jpa.war/WEB-INF/classes/META-INF/persistence.xml**:

   ```
   <!-- <property name="hibernate.cache.provider_class"
   value="org.hibernate.cache.HashtableCacheProvider"/> -->
   ```

   b. Ajouter la propriété de module du fournisseur dans le fichier **jboss-seam-booking.jar/META-INF/persistence.xml**:

   ```
   <property name="jboss.as.jpa.providerModule" value="hibernate3-
   bundled" />
   ```

c. Modifier la propriété **`jta-data-source`** pour utiliser le nom JNDI de la source de données JDBC par défaut :

```
<jta-data-source>java:jboss/datasources/ExampleDS</jta-data-source>
```

3. **Ajouter les dépendances Seam 2.2**
Copier les JAR suivantes de la bibliothèque de la distro Seam 2.2, **`SEAM_HOME/lib/`**, dans le répertoire **`jboss-seam-jpa.war/WEB-INF/lib/`** :

   - antlr.jar
   - slf4j-api.jar
   - slf4j-log4j12.jar
   - hibernate-entitymanager.jar
   - hibernate-core.jar
   - hibernate-annotations.jar
   - hibernate-commons-annotations.jar
   - hibernate-validator.jar

4. **Créer un fichier jboss-deployment-structure auquel ajouter les dépendances restantes**
Créer le fichier **`jboss-deployment-structure.xml`** dans le dossier **`jboss-seam-jpa.war/WEB-INF/`** contenant les informations suivantes :

```
<jboss-deployment-structure>
   <deployment>
        <exclusions>
          <module name="javax.faces.api" slot="main"/>
          <module name="com.sun.jsf-impl" slot="main"/>
          <module name="org.hibernate" slot="main"/>
        </exclusions>
        <dependencies>
          <module name="org.apache.log4j" />
          <module name="org.dom4j" />
          <module name="org.apache.commons.logging" />
          <module name="org.apache.commons.collections" />
          <module name="javax.faces.api" slot="1.2"/>
          <module name="com.sun.jsf-impl" slot="1.2"/>
        </dependencies>
    </deployment>
</jboss-deployment-structure>
```

Résultat :

L'exemple d'application Seam 2.2 JPA déploie et exécute avec succès sur JBoss EAP 6.

Rapporter un bogue

4.3.3. Migrer l'exemple de Seam 2.2 Booking dans JBoss EAP 6

Résumé

La migration du Seam 2.2 Booking EAR est plus compliquée que celle de l'exemple Seam 2.2 JPA WAR. On peut trouver la documentation pour l'exemple Seam 2.2 JPA WAR à cet endroit : Section 4.3.2, « Migrer l'exemple Seam 2.2 dans JBoss EAP 6 ». Pour migrer l'application, vous devez procéder ainsi :

1. Initialiser JSF 1.2 à la place de JSF 2 (défaut).
2. Regrouper les anciennes versions des JAR Hibernate plutôt que les JARS fournis dans Boss EAP 6.
3. Changer les liaisons JNDI pour qu'elles utilisent la nouvelle syntaxe de Java EE 6 JNDI portable.

Les deux premières étapes ci-dessus avaient lieu dans l'exemple de migration de Seam 2.2 JPA WAR. La troisième étape est nouvelle et elle est nécessaire car l'EAR contient des EJB.

IMPORTANT

Les applications qui utilisent Hibernate directement avec Seam 2.2 utilisent sans doute une version d'Hibernate 3 empaquetée dans l'application. Hibernate 4, fourni par le module org.hibernate de JBoss EAP 6, n'est pas pris en charge par Seam 2.2. Cet exemple a pour but de vous aider à exécuter votre application dans JBoss EAP 6 comme première étape. Notez qu'empaqueter Hibernate 3 avec une application Seam 2.2 n'est pas une configuration prise en charge.

Procédure 4.7. Migration de l'exemple Seam 2.2 Booking

1. **Créer le fichier `jboss-deployment-structure.xml`**
   Créer un nouveau fichier nommé **`jboss-deployment-structure.xml`** dans **`jboss-seam-booking.ear/META-INF/`** et ajouter le contenu suivant :

```
<jboss-deployment-structure xmlns="urn:jboss:deployment-
structure:1.0">
  <deployment>
        <dependencies>
          <module name="javax.faces.api" slot="1.2" export="true"/>
          <module name="com.sun.jsf-impl" slot="1.2"
export="true"/>
          <module name="org.apache.log4j" export="true"/>
          <module name="org.dom4j" export="true"/>
          <module name="org.apache.commons.logging" export="true"/>
          <module name="org.apache.commons.collections"
export="true"/>
        </dependencies>
        <exclusions>
          <module name="org.hibernate" slot="main"/>
       </exclusions>

  </deployment>
  <sub-deployment name="jboss-seam-booking.war">
      <exclusions>
          <module name="javax.faces.api" slot="main"/>
          <module name="com.sun.jsf-impl" slot="main"/>
        </exclusions>
        <dependencies>
```

```
            <module name="javax.faces.api" slot="1.2"/>
            <module name="com.sun.jsf-impl" slot="1.2"/>
        </dependencies>
    </sub-deployment>
</jboss-deployment-structure>
```

2. **Modifier le fichier `jboss-seam-booking.jar/META-INF/persistence.xml` comme suit.**

   a. Supprimer ou dé-commenter la propriété hibernate de la classe de fournisseur de cache :

   ```
   <!-- <property name="hibernate.cache.provider_class"
   value="org.hibernate.cache.HashtableCacheProvider"/> -->
   ```

   b. Ajouter la propriété de module du fournisseur dans le fichier **`jboss-seam-booking.jar/META-INF/persistence.xml`**:

   ```
   <property name="jboss.as.jpa.providerModule" value="hibernate3-
   bundled" />
   ```

   c. Modifier la propriété **`jta-data-source`** pour utiliser le nom JNDI de la source de données JDBC par défaut :

   ```
   <jta-data-source>java:jboss/datasources/ExampleDS</jta-data-
   source>
   ```

3. **Copier les JAR de la distribution Seam 2.2**
   Copier les JAR suivants de la distribution Seam 2.2 ***`EAP5.x_HOME`*`/jboss-eap5.x/seam/lib/`** dans le répertoire **`jboss-seam-booking.ear/lib`**.

   ```
   antlr.jar
   slf4j-api.jar
   slf4j-log4j12.jar
   hibernate-core.jar
   hibernate-entitymanager.jar
   hibernate-validator.jar
   hibernate-annotations.jar
   hibernate-commons-annotations.jar
   ```

4. **Modification des noms de Recherche JNDI**
   Changer les chaînes de recherche JNDI dans le fichier **`jboss-seam-booking.war/WEB-INF/components.xml`**. En raison de nouvelles règles portables JNDI, JBoss EAP 6 se relie maintenant aux EJB à l'aide de règles de syntaxe portables JNDI et vous ne pouvez pas utiliser jndiPattern utilisé dans JBoss EAP 5. Voici les changements requis pour JBoss EAP 6 :

   ```
   java:global/jboss-seam-booking/jboss-seam-
   booking/HotelSearchingAction!org.jboss.seam.example.booking.HotelSea
   rching
   java:app/jboss-seam-
   booking/HotelSearchingAction!org.jboss.seam.example.booking.HotelSea
   rching
   java:module/HotelSearchingAction!org.jboss.seam.example.booking.Hote
   lSearching
   java:global/jboss-seam-booking/jboss-seam-
   ```

```
booking/HotelSearchingAction
java:app/jboss-seam-booking/HotelSearchingAction
java:module/HotelSearchingAction
```

Les chaînes de recherche JNDI des EJB du framework Seam 2.2 doivent être modifiées ainsi :

```
java:global/jboss-seam-booking/jboss-
seam/EjbSynchronizations!org.jboss.seam.transaction.LocalEjbSynchron
izations
java:app/jboss-
seam/EjbSynchronizations!org.jboss.seam.transaction.LocalEjbSynchron
izations
java:module/EjbSynchronizations!org.jboss.seam.transaction.LocalEjbS
ynchronizations
java:global/jboss-seam-booking/jboss-seam/EjbSynchronizations
java:app/jboss-seam/EjbSynchronizations
java:module/EjbSynchronizations
```

Vous pouvez adopter une des approches suivantes :

a. **Ajouter des éléments de composants**
   Vous pouvez ajouter un **`jndi-name`** pour chaque EJB du **`WEB-INF/components.xml`**:

```
    <component
class="org.jboss.seam.transaction.EjbSynchronizations" jndi-
name="java:app/jboss-seam/EjbSynchronizations"/>
    <component
class="org.jboss.seam.async.TimerServiceDispatcher" jndi-
name="java:app/jboss-seam/TimerServiceDispatcher"/>
    <component
class="org.jboss.seam.example.booking.AuthenticatorAction" jndi-
name="java:app/jboss-seam-booking/AuthenticatorAction" />
    <component
class="org.jboss.seam.example.booking.BookingListAction"  jndi-
name="java:app/jboss-seam-booking/BookingListAction" />
    <component
class="org.jboss.seam.example.booking.RegisterAction" jndi-
name="java:app/jboss-seam-booking/RegisterAction" />
    <component
class="org.jboss.seam.example.booking.HotelSearchingAction" jndi-
name="java:app/jboss-seam-booking/HotelSearchingAction" />
    <component
class="org.jboss.seam.example.booking.HotelBookingAction" jndi-
name="java:app/jboss-seam-booking/HotelBookingAction" />
    <component
class="org.jboss.seam.example.booking.ChangePasswordAction" jndi-
name="java:app/jboss-seam-booking/ChangePasswordAction" />
```

b. Vous pouvez modifier le code en ajoutant l'annotation **`@JNDIName(value="")`** qui indique le chemin JNDI. Vous trouverez un exemple de modification de code de bean de session stateless ci-dessous. Vous trouverez également une description détaillée de ce processus dans la documentation de référence Seam 2.2.

```
@Stateless
@Name("authenticator")
```

```
@JndiName(value="java:app/jboss-seam-
booking/AuthenticatorAction")
public class AuthenticatorAction
    implements Authenticator
{
...
}
```

Résultat :

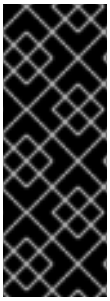
L'exemple Seam 2.2 Booking déploie et exécute avec succès sur JBoss EAP 6.

Rapporter un bogue

4.3.4. Migrer l'archive du Seam 2.2 Booking dans JBoss EAP 6: Instructions étape par étape

Il s'agit d'un guide étape par étape sur la façon de déplacer l'archive de l'application Seam 2.2 Booking de JBoss EAP 5.X vers JBoss EAP 6. Malgré le fait qu'il y ait de meilleures approches pour faire migrer les applications, de nombreux développeurs pourraient être tentés de déployer l'archive de l'application telle-qu'elle dans le serveur de JBoss EAP 6 pour voir ce qui se passe. Le but de ce document est de montrer les types de problèmes que vous pouvez rencontrer quand vous faîtes cela et comment vous pourrez déboguer er résoudre ces problèmes.

Dans cet exemple, l'application EAR est déployée dans le répertoire ***EAP6_HOME*/standalone/deployments** sans aucun autre changement que d'extraire les archives. Cela vous permet de modifier plus facilement les fichiers XML contenus dans les archives au fur et à mesure que vous rencontrez ou résolvez les problèmes.



IMPORTANT

Les applications qui utilisent Hibernate directement avec Seam 2.2 fon sans doute usage d'une version d'Hibernate 3 empaquetée dans l'application. Hibernate 4, fourni par le module org.hibernate de JBoss EAP 6, n'est pas pris en charge par Seam 2.2. Cet exemple a pour but de vous aider à exécuter votre application dans JBoss EAP 6 comme première étape. Notez qu'empaqueter Hibernate 3 avec une application Seam 2.2 n'est pas une configuration prise en charge.

Procédure 4.8. Migrer l'application

1. Section 4.3.5, « Générer et déployer la version JBoss EAP 5.X de l'application Seam 2.2 Booking »
2. Section 4.3.6, « Déboguer et résoudre les Erreurs de déploiement et les Exceptions de Seam 2.2 Booking Archive »
3. Section 4.3.7, « Déboguer et résoudre les Erreurs de runtime et les Exceptions de Seam 2.2 Booking Archive »

À ce point, vous serez en mesure d'accéder facilement à l'application dans un navigateur par l'URL http://localhost:8080/seam-booking/. Connectez-vous par demo/demo et vous apercevrez la page de bienvenue de Booking.

Récapitulatif des changements

Section 4.3.8, « Récapitulatif des changements à effectuer lors d'une Migration de l'application Seam 2.2 Booking »

Rapporter un bogue

4.3.5. Générer et déployer la version JBoss EAP 5.X de l'application Seam 2.2 Booking

Avant de migrer cette application, vous devrez générer l'application Seam 2.2 Booking de JBoss EAP 5.X, extraire l'archive, et la copier dans le dossier de déploiement JBoss EAP 6.

Procédure 4.9. Générer et déployer l'EAR

1. Générer l'EAR :

```
$ cd /EAP5_HOME/jboss-eap5.x/seam/examples/booking
$ ANT_HOME/ant explode
```

   Remplacer *jboss-eap5.x* par la version JBoss EAP à partir de laquelle vous migrez

2. Copier l'EAR dans le répertoire des déploiements *EAP6_HOME*:

```
$ cp -r EAP5_HOME/seam/examples/booking/exploded-archives/jboss-
seam-booking.ear EAP6_HOME/standalone/deployments/
$ cp -r EAP5_HOME/seam/examples/booking/exploded-archives/jboss-
seam-booking.war EAP6_HOME/standalone/deployments/jboss-seam.ear
$ cp -r EAP5_HOME/seam/examples/booking/exploded-archives/jboss-
seam-booking.jar EAP6_HOME/standalone/deployments/jboss-seam.ear
```

3. Démarrer le serveur de JBoss EAP 6 et vérifier le log. Vous apercevrez :

```
INFO [org.jboss.as.deployment] (DeploymentScanner-threads - 1) Found
jboss-seam-booking.ear in deployment directory.
    To trigger deployment create a file called jboss-seam-
booking.ear.dodeploy
```

4. Créer un fichier vide ayant pour nom **`jboss-seam-booking.ear.dodeploy`** et copier le dans le répertoire ***`EAP6_HOME`*`/standalone/deployments`**. Vous devez copier ce fichier dans le répertoire de déploiements plusieurs fois lors de la migration de cette application, donc gardez-le dans un endroit où vous pouvez facilement le trouver. Dans le journal, vous devriez maintenant voir les messages suivants, indiquant que c'est le déploiement :

```
INFO [org.jboss.as.server.deployment] (MSC service thread 1-1)
Starting deployment of "jboss-seam-booking.ear"
INFO [org.jboss.as.server.deployment] (MSC service thread 1-3)
Starting deployment of "jboss-seam-booking.jar"
INFO [org.jboss.as.server.deployment] (MSC service thread 1-6)
Starting deployment of "jboss-seam.jar"
INFO [org.jboss.as.server.deployment] (MSC service thread 1-2)
Starting deployment of "jboss-seam-booking.war"
```

   A ce moment, vous risquerez d'apercevoir votre première erreur de déploiement. Dans la prochaine étape, vous rencontrerez les problèmes un à un et vous apprendrez comment les déboguer et les résoudre.

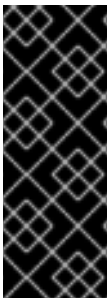
Pour apprendre comment déboguer et comment résoudre les problèmes de déploiement, cliquer ici : Section 4.3.6, « Déboguer et résoudre les Erreurs de déploiement et les Exceptions de Seam 2.2 Booking Archive »

Pour retourner au sujet précédent, cliquer ici : Section 4.3.4, « Migrer l'archive du Seam 2.2 Booking dans JBoss EAP 6: Instructions étape par étape »

Rapporter un bogue

4.3.6. Déboguer et résoudre les Erreurs de déploiement et les Exceptions de Seam 2.2 Booking Archive

Dans l'étape précédente, Section 4.3.5, « Générer et déployer la version JBoss EAP 5.X de l'application Seam 2.2 Booking », vous avez créé l'application JBos EAP 5.X Seam 2.2 Booking et vous l'avez déployée dans le dossier de déploiement JBoss EAP 6. Au cours de cette étape, vous avez résolu et débogué chaque erreur de déploiement que vous avez rencontrée.



IMPORTANT

Les applications qui utilisent Hibernate directement avec Seam 2.2 utilisent sans doute une version d'Hibernate 3 empaquetée dans l'application. Hibernate 4, fourni par le module org.hibernate de JBoss EAP 6, n'est pas pris en charge par Seam 2.2. Cet exemple a pour but de vous aider à exécuter votre application dans JBoss EAP 6 comme première étape. Notez qu'empaqueter Hibernate 3 avec une application Seam 2.2 n'est pas une configuration prise en charge.

Procédure 4.10. Déboguer et résoudre les erreurs et les exceptions de déploiement

1. Problème - java.lang.ClassNotFoundException: javax.faces.FacesException

   Quand vous déployez l'application, le journal contient l'erreur suivante :

   ```
   ERROR \[org.jboss.msc.service.fail\] (MSC service thread 1-1)
   MSC00001: Failed to start service jboss.deployment.subunit."jboss-
   seam-booking.ear"."jboss-seam-booking.war".POST_MODULE:
   org.jboss.msc.service.StartException in service
   jboss.deployment.subunit."jboss-seam-booking.ear"."jboss-seam-
   booking.war".POST_MODULE:
   Failed to process phase POST_MODULE of subdeployment "jboss-seam-
   booking.war" of deployment "jboss-seam-booking.ear"
       (.. additional logs removed ...)
   Caused by: java.lang.ClassNotFoundException:
   javax.faces.FacesException from \[Module "deployment.jboss-seam-
   booking.ear:main" from Service Module Loader\]
       at
   org.jboss.modules.ModuleClassLoader.findClass(ModuleClassLoader.java
   :191)
   ```

   Ceci signifie :

   L'exception ClassNotFoundException indique une dépendance manquante. Dans ce cas, impossible de trouver la classe **`javax.faces.FacesException`** et vous devez donc ajouter explicitement la dépendance.

   Comment résoudre ceci :

Chercher le nom du module pour cette classe dans le répertoire ***EAP6_HOME*/modules/system/layers/base/** à l'aide d'un chemin d'accès qui corresponde à la classe manquante. Dans ce cas, vous trouverez 2 modules qui correspondent :

```
javax/faces/api/main
javax/faces/api/1.2
```

Les deux modules ont le même nom de module : **javax.faces.api** mais un d'entre eux, qui se trouve dans le répertoire principal, est pour JSF 2.0 et l'autre, situé dans le répertoire 1.2, est pour JSF 1.2. S'il n'y avait qu'un seul module disponible, vous pourriez simplement créer un fichier **MANIFEST.MF** et ajouter la dépendance de module. Mais, dans ce cas, vous devez utiliser la version JSF 1.2 et non pas la version 2.0 du répertoire principal, donc vous devez en spécifier une et exclure l'autre. Pour ceci, créer un fichier **jboss-deployment-structure.xml** dans le répertoire **META-INF/** de l'EAR qui contiendra les données suivantes :

```
<jboss-deployment-structure xmlns="urn:jboss:deployment-
structure:1.0">
  <deployment>
      <dependencies>
        <module name="javax.faces.api" slot="1.2" export="true"/>
      </dependencies>
  </deployment>
  <sub-deployment name="jboss-seam-booking.war">
    <exclusions>
        <module name="javax.faces.api" slot="main"/>
      </exclusions>
      <dependencies>
        <module name="javax.faces.api" slot="1.2"/>
      </dependencies>
  </sub-deployment>
</jboss-deployment-structure>
```

Dans la section **deployment**, vous ajoutez la dépendance pour l'API **javax.faces.api** du module JSF 1.2. Vous ajoutez aussi la dépendance du module JSF 1.2 dans la section de sous-déploiement du WAR et vous excluez le module JSF 2.0.

Redéployer l'application en effaçant le fichier ***EAP6_HOME*/standalone/deployments/jboss-seam-booking.ear.failed** et en créant un fichier vide **jboss-seam-booking.ear.dodeploy** dans le même répertoire.

2. Problème - java.lang.ClassNotFoundException: org.apache.commons.logging.Log

   Quand vous déployez l'application, le journal contient l'erreur suivante :

```
ERROR [org.jboss.msc.service.fail] (MSC service thread 1-8)
MSC00001: Failed to start service jboss.deployment.unit."jboss-seam-
booking.ear".INSTALL:
org.jboss.msc.service.StartException in service
jboss.deployment.unit."jboss-seam-booking.ear".INSTALL:
Failed to process phase INSTALL of deployment "jboss-seam-
booking.ear"
    (.. additional logs removed ...)
```

```
Caused by: java.lang.ClassNotFoundException:
org.apache.commons.logging.Log from [Module "deployment.jboss-seam-
booking.ear.jboss-seam-booking.war:main" from Service Module Loader]
```

Ceci signifie :

L'exception **`ClassNotFoundException`** indique une dépendance manquante. Dans ce cas, impossible de trouver la classe **`org.apache.commons.logging.Log`** et vous devez donc ajouter explicitement la dépendance.

Comment résoudre ceci :

Chercher le nom du module pour cette classe dans le répertoire ***`EAP6_HOME`*`/modules/system/layers/base/`** en cherchant un chemin d'accès qui corresponde à la classe manquante. Dans ce cas, vous trouverez un module qui correspond au chemin **`org/apache/commons/logging/`**. Le nom du module est "org.apache.commons.logging".

Modifier le fichier **`jboss-deployment-structure.xml`** pour ajouter la dépendance de module à la section de déploiement du fichier.

```
<module name="org.apache.commons.logging" export="true"/>
```

Le fichier **`jboss-deployment-structure.xml`** devrait maintenant ressembler à ceci :

```
<jboss-deployment-structure xmlns="urn:jboss:deployment-
structure:1.0">
  <deployment>
      <dependencies>
        <module name="javax.faces.api" slot="1.2" export="true"/>
        <module name="org.apache.commons.logging" export="true"/>
      </dependencies>
  </deployment>
  <sub-deployment name="jboss-seam-booking.war">
    <exclusions>
        <module name="javax.faces.api" slot="main"/>
      </exclusions>
      <dependencies>
        <module name="javax.faces.api" slot="1.2"/>
      </dependencies>
  </sub-deployment>
</jboss-deployment-structure>
```

Redéployer l'application en effaçant le fichier ***`EAP6_HOME`*`/standalone/deployments/jboss-seam-booking.ear.failed`** et en créant un fichier vide **`jboss-seam-booking.ear.dodeploy`** dans le même répertoire.

3. Problème - java.lang.ClassNotFoundException: org.dom4j.DocumentException

   Quand vous déployez l'application, le journal contient l'erreur suivante :

```
ERROR [org.apache.catalina.core.ContainerBase.[jboss.web].[default-
host].[/seam-booking]] (MSC service thread 1-3) Exception sending
context initialized event to listener instance of class
org.jboss.seam.servlet.SeamListener: java.lang.NoClassDefFoundError:
org/dom4j/DocumentException
```

```
    (... additional logs removed ...)
Caused by: java.lang.ClassNotFoundException:
org.dom4j.DocumentException from [Module "deployment.jboss-seam-
booking.ear.jboss-seam.jar:main" from Service Module Loader]
```

Ceci signifie :

L'exception **`ClassNotFoundException`** indique une dépendance manquante. Dans ce cas, impossible de trouver la classe **`org.dom4j.DocumentException`**.

Comment résoudre ceci :

Trouver le nom de module dans le répertoire ***`EAP6_HOME`*`/modules/system/layers/base/`** en cherchant **`org/dom4j/DocumentException`**. Le nom du module est “org.dom4j”. Modifier le fichier **`jboss-deployment-structure.xml`** pour ajouter la dépendance de module à la section de déploiement du fichier.

```
<module name="org.dom4j" export="true"/>
```

Le **`jboss-deployment-structure.xml`** devrait maintenant ressembler à ceci :

```
<jboss-deployment-structure xmlns="urn:jboss:deployment-
structure:1.0">
  <deployment>
      <dependencies>
        <module name="javax.faces.api" slot="1.2" export="true"/>
        <module name="org.apache.commons.logging" export="true"/>
            <module name="org.dom4j" export="true"/>
          </dependencies>
  </deployment>
  <sub-deployment name="jboss-seam-booking.war">
    <exclusions>
        <module name="javax.faces.api" slot="main"/>
      </exclusions>
      <dependencies>
        <module name="javax.faces.api" slot="1.2"/>
      </dependencies>
  </sub-deployment>
</jboss-deployment-structure>
```

Redéployer l'application en effaçant le fichier ***`EAP6_HOME`*`/standalone/deployments/jboss-seam-booking.ear.failed`** et en créant un fichier vide **`jboss-seam-booking.ear.dodeploy`** dans le même répertoire.

4. Problème - java.lang.ClassNotFoundException: org.hibernate.validator.InvalidValue

   Quand vous déployez l'application, le journal contient l'erreur suivante :

```
ERROR [org.apache.catalina.core.ContainerBase.[jboss.web].[default-
host].[/seam-booking]] (MSC service thread 1-6) Exception sending
context initialized event to listener instance of class
org.jboss.seam.servlet.SeamListener: java.lang.RuntimeException:
Could not create Component:
org.jboss.seam.international.statusMessages
    (... additional logs removed ...)
```

```
Caused by: java.lang.ClassNotFoundException:
org.hibernate.validator.InvalidValue from [Module "deployment.jboss-
seam-booking.ear.jboss-seam.jar:main" from Service Module Loader]
```

Ceci signifie :

L'exception **`ClassNotFoundException`** indique une dépendance manquante. Dans ce cas, impossible de trouver la classe **`org.hibernate.validator.InvalidValue`**.

Comment résoudre ceci :

Il y a un module "org.hibernate.validator", mais le JAR ne contient pas la classe **`org.hibernate.validator.InvalidValue`**, donc si on ajoute la dépendance de module, on ne résout pas le problème. Dans un tel cas, le JAR qui contient la classe faisait partie du déploiement de JBoss EAP 5.X. Chercher le JAR qui contient la classe manquante dans le répertoire ***`EAP5_HOME/seam/lib/`***. Pour cela, ouvrir la console et tapez ce qui suit :

```
$ cd EAP5_HOME/seam/lib
$ grep 'org.hibernate.validator.InvalidValue' `find . -name '*.jar'`
```

Résultat :

```
$ Binary file ./hibernate-validator.jar matches
$ Binary file ./test/hibernate-all.jar matches
```

Dans ce cas, copier **`hibernate-validator.jar`** dans le répertoire **`jboss-seam-booking.ear/lib/`** :

```
$ cp EAP5_HOME/seam/lib/hibernate-validator.jar jboss-seam-
booking.ear/lib
```

Redéployer l'application en effaçant le fichier ***`EAP6_HOME`*`/standalone/deployments/jboss-seam-booking.ear.failed`** et en créant un fichier vide **`jboss-seam-booking.ear.dodeploy`** dans le même répertoire.

5. Problème - java.lang.InstantiationException: org.jboss.seam.jsf.SeamApplicationFactory

   Quand vous déployez l'application, le journal contient l'erreur suivante :

```
INFO  [javax.enterprise.resource.webcontainer.jsf.config] (MSC
service thread 1-7) Unsanitized stacktrace from failed start...:
com.sun.faces.config.ConfigurationException: Factory
'javax.faces.application.ApplicationFactory' was not configured
properly.
  at
com.sun.faces.config.processor.FactoryConfigProcessor.verifyFactorie
sExist(FactoryConfigProcessor.java:296) [jsf-impl-2.0.4-b09-
jbossorg-4.jar:2.0.4-b09-jbossorg-4]
  (... additional logs removed ...)
Caused by: javax.faces.FacesException:
org.jboss.seam.jsf.SeamApplicationFactory
  at
javax.faces.FactoryFinder.getImplGivenPreviousImpl(FactoryFinder.jav
a:606) [jsf-api-1.2_13.jar:1.2_13-b01-FCS]
  (... additional logs removed ...)
```

```
  at
com.sun.faces.config.processor.FactoryConfigProcessor.verifyFactorie
sExist(FactoryConfigProcessor.java:294) [jsf-impl-2.0.4-b09-
jbossorg-4.jar:2.0.4-b09-jbossorg-4]
  ... 11 more
Caused by: java.lang.InstantiationException:
org.jboss.seam.jsf.SeamApplicationFactory
  at java.lang.Class.newInstance0(Class.java:340) [:1.6.0_25]
  at java.lang.Class.newInstance(Class.java:308) [:1.6.0_25]
  at
javax.faces.FactoryFinder.getImplGivenPreviousImpl(FactoryFinder.jav
a:604) [jsf-api-1.2_13.jar:1.2_13-b01-FCS]
  ... 16 more
```

Ceci signifie :

Les **`com.sun.faces.config.ConfigurationException`** et **`java.lang.InstantiationException`** indiquent un problème de dépendance. Dans un tel cas, la cause n'est pas évidente.

Comment résoudre ceci :

Il vous faut trouver le module qui contient les classes **`com.sun.faces`**. Malgré qu'il n'y ait pas de module **`com.sun.faces`**, il y a deux modules **`com.sun.jsf-impl`**. Une simple vérification du **`jsf-impl-1.2_13.jar`** dans le répertoire 1.2 révèle qu'il contient les classes **`com.sun.faces`**. Comme avec les exceptions **`javax.faces.FacesException`** **`ClassNotFoundException`**, vous devez utiliser la version JSF 1.2 et non pas la version JSF 2.0 dans le principal, donc vous devrez en indiquer une et en exclure l'autre. Vous devrez modifier le fichier **`jboss-deployment-structure.xml`** pour ajouter la dépendance de module à la section de déploiement du fichier. Vous devrez aussi y ajouter le sous-déploiement du WAR et exclure le module JSF 2.0. Le fichier devrait ressembler à ceci :

```
<jboss-deployment-structure xmlns="urn:jboss:deployment-
structure:1.0">
  <deployment>
      <dependencies>
        <module name="javax.faces.api" slot="1.2" export="true"/>
              <module name="com.sun.jsf-impl" slot="1.2"
export="true"/>
        <module name="org.apache.commons.logging" export="true"/>
        <module name="org.dom4j" export="true"/>
      </dependencies>
  </deployment>
  <sub-deployment name="jboss-seam-booking.war">
    <exclusions>
        <module name="javax.faces.api" slot="main"/>
        <module name="com.sun.jsf-impl" slot="main"/>
      </exclusions>
      <dependencies>
        <module name="javax.faces.api" slot="1.2"/>
              <module name="com.sun.jsf-impl" slot="1.2"/>
      </dependencies>
  </sub-deployment>
</jboss-deployment-structure>
```

Redéployer l'application en effaçant le fichier ***EAP6_HOME*/standalone/deployments/jboss-seam-booking.ear.failed** et en créant un fichier vide **jboss-seam-booking.ear.dodeploy** dans le même répertoire.

6. Problème - java.lang.ClassNotFoundException: org.apache.commons.collections.ArrayStack

   Quand vous déployez l'application, le journal contient l'erreur suivante :

```
ERROR [org.apache.catalina.core.ContainerBase.[jboss.web].[default-
host].[/seam-booking]] (MSC service thread 1-1) Exception sending
context initialized event to listener instance of class
com.sun.faces.config.ConfigureListener: java.lang.RuntimeException:
com.sun.faces.config.ConfigurationException: CONFIGURATION FAILED!
org.apache.commons.collections.ArrayStack from [Module
"deployment.jboss-seam-booking.ear:main" from Service Module Loader]
    (... additional logs removed ...)
Caused by: java.lang.ClassNotFoundException:
org.apache.commons.collections.ArrayStack from [Module
"deployment.jboss-seam-booking.ear:main" from Service Module Loader]
```

   Ceci signifie :

   Le **ClassNotFoundException** indique une dépendance manquante. Dans ce cas, il ne peut pas trouver la classe **org.apache.commons.collections.ArrayStack**.

   Comment résoudre ceci :

   Trouver le nom de module dans le répertoire ***EAP6_HOME*/modules/system/layers/base/** en cherchant **org/apache/commons/collections**. Le nom du module est “org.apache.commons.collections”. Modifier le fichier **jboss-deployment-structure.xml** pour ajouter la dépendance de module à la section de déploiement du fichier.

```
<module name="org.apache.commons.collections" export="true"/>
```

   Le **jboss-deployment-structure.xml** devrait maintenant ressembler à ceci :

```
<jboss-deployment-structure xmlns="urn:jboss:deployment-
structure:1.0">
  <deployment>
      <dependencies>
        <module name="javax.faces.api" slot="1.2" export="true"/>
              <module name="com.sun.jsf-impl" slot="1.2"
export="true"/>
        <module name="org.apache.commons.logging" export="true"/>
        <module name="org.dom4j" export="true"/>
        <module name="org.apache.commons.collections"
export="true"/>
    </dependencies>
  </deployment>
  <sub-deployment name="jboss-seam-booking.war">
    <exclusions>
        <module name="javax.faces.api" slot="main"/>
        <module name="com.sun.jsf-impl" slot="main"/>
      </exclusions>
      <dependencies>
        <module name="javax.faces.api" slot="1.2"/>
```

```
                <module name="com.sun.jsf-impl" slot="1.2"/>
        </dependencies>
    </sub-deployment>
</jboss-deployment-structure>
```

Redéployer l'application en effaçant le fichier ***EAP6_HOME*/standalone/deployments/jboss-seam-booking.ear.failed** et en créant un fichier vide **jboss-seam-booking.ear.dodeploy** dans le même répertoire.

7. Problème - Services avec des dépendances manquantes/non disponibles

   Quand vous déployez l'application, le journal contient l'erreur suivante :

```
ERROR [org.jboss.as.deployment] (DeploymentScanner-threads - 2)
{"Composite operation failed and was rolled back. Steps that
failed:" => {"Operation step-2" => {"Services with
missing/unavailable dependencies" =>
["jboss.deployment.subunit.\"jboss-seam-booking.ear\".\"jboss-seam-
booking.jar\".component.AuthenticatorAction.START missing [
jboss.naming.context.java.comp.jboss-seam-booking.\"jboss-seam-
booking.jar\".AuthenticatorAction.\"env/org.jboss.seam.example.booki
ng.AuthenticatorAction/em\" ]","jboss.deployment.subunit.\"jboss-
seam-booking.ear\".\"jboss-seam-
booking.jar\".component.HotelSearchingAction.START missing [
jboss.naming.context.java.comp.jboss-seam-booking.\"jboss-seam-
booking.jar\".HotelSearchingAction.\"env/org.jboss.seam.example.book
ing.HotelSearchingAction/em\" ]","
  (... additional logs removed ...)
"jboss.deployment.subunit.\"jboss-seam-booking.ear\".\"jboss-seam-
booking.jar\".component.BookingListAction.START missing [
jboss.naming.context.java.comp.jboss-seam-booking.\"jboss-seam-
booking.jar\".BookingListAction.\"env/org.jboss.seam.example.booking
.BookingListAction/em\" ]","jboss.persistenceunit.\"jboss-seam-
booking.ear/jboss-seam-booking.jar#bookingDatabase\" missing [
jboss.naming.context.java.bookingDatasource ]"]}}}
```

   Ceci signifie :

   Quand vous avez l'erreur suivante : “Services with missing/unavailable dependencies”, chercher le texte entre les guillemets après “missing”. Dans ce cas, vous verrez :

```
missing [ jboss.naming.context.java.comp.jboss-seam-booking.\"jboss-
seam-
booking.jar\".AuthenticatorAction.\"env/org.jboss.seam.example.booki
ng.AuthenticatorAction/em\" ]
```

   “/em” indique un problème de Gestionnaire d'entité (Entity Manager) et de source de données.

   Comment résoudre ceci :

   Dans JBoss EAP 6, la configuration de la source de données a été modifiée et doit être définie dans le fichier ***EAP6_HOME*/standalone/configuration/standalone.xml**. Comme JBoss EAP 6 est fourni avec une base de données déjà définie dans le fichier **standalone.xml**, modifier le fichier **persistence.xml** pour utiliser cet exemple de base de données dans cette application. Si vous regardez dans le fichier **standalone.xml**, vous verrez que **jndi-name** de l'exemple de source de données est

java:jboss/datasources/ExampleDS. Modifier le fichier **jboss-seam-booking.jar/META-INF/persistence.xml** pour commenter l'élément **jta-data-source** et le remplacer comme suit :

```
<!-- <jta-data-source>java:/bookingDatasource</jta-data-source> -->
<jta-data-source>java:jboss/datasources/ExampleDS</jta-data-source>
```

Redéployer l'application en effaçant le fichier ***EAP6_HOME*/standalone/deployments/jboss-seam-booking.ear.failed** et en créant un fichier vide **jboss-seam-booking.ear.dodeploy** dans le même répertoire.

8. À ce moment là, l'application se déploie sans erreur, mais quand vous accédez à l'URL http://localhost:8080/seam-booking/ par un navigateur et tentez un "Account Login", vous obtenez l'erreur suivante : “The page isn't redirecting properly”. Dans une prochaine étape, vous allez apprendre comment déboguer et résoudre les erreurs de runtime.

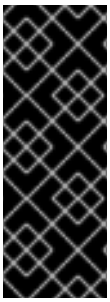
   Pour apprendre comment débogger et comment résoudre les problèmes de runtime, cliquer ici : Section 4.3.7, « Déboguer et résoudre les Erreurs de runtime et les Exceptions de Seam 2.2 Booking Archive »

   Pour retourner au sujet précédent, cliquer ici : Section 4.3.4, « Migrer l'archive du Seam 2.2 Booking dans JBoss EAP 6: Instructions étape par étape »

Rapporter un bogue

4.3.7. Déboguer et résoudre les Erreurs de runtime et les Exceptions de Seam 2.2 Booking Archive

Dans l'étape précédente, Section 4.3.6, « Déboguer et résoudre les Erreurs de déploiement et les Exceptions de Seam 2.2 Booking Archive », vous avez appris comment déboguer des erreurs de déploiement. Au cours de cette étape, vous avez résolu et débogué chaque erreur de déploiement que vous avez rencontrée.



IMPORTANT

Les applications qui utilisent Hibernate directement avec Seam 2.2 utilisent sans doute une version d'Hibernate 3 empaquetée dans l'application. Hibernate 4, fourni par le module org.hibernate de JBoss EAP 6, n'est pas pris en charge par Seam 2.2. Cet exemple a pour but de vous aider à exécuter votre application dans JBoss EAP 6 comme première étape. Notez qu'empaqueter Hibernate 3 avec une application Seam 2.2 n'est pas une configuration prise en charge.

Procédure 4.11. Déboguer et résoudre les erreurs et les exceptions de runtime

À ce moment là, quand vous déployez l'application, vous ne pouvez pas voir d'erreurs dans le log. Cependant, quand vous accédez à l'URL de l'application, des erreurs apparaissent dans le log.

1. Problème - javax.naming.NameNotFoundException: Name 'jboss-seam-booking' not found in context ''

   Quand vous accédez à l'URL http://localhost:8080/seam-booking/ dans un navigateur, vous obtenez "The page isn't redirecting properly" et le journal contient l'erreur suivante :

```
SEVERE [org.jboss.seam.jsf.SeamPhaseListener] (http--127.0.0.1-8080-
```

```
1) swallowing exception: java.lang.IllegalStateException: Could not
start transaction
  at
org.jboss.seam.jsf.SeamPhaseListener.begin(SeamPhaseListener.java:59
8) [jboss-seam.jar:]
  (... log messages removed ...)
Caused by: org.jboss.seam.InstantiationException: Could not
instantiate Seam component:
org.jboss.seam.transaction.synchronizations
  at org.jboss.seam.Component.newInstance(Component.java:2170)
[jboss-seam.jar:]
  (... log messages removed ...)
Caused by: javax.naming.NameNotFoundException: Name 'jboss-seam-
booking' not found in context ''
  at
org.jboss.as.naming.util.NamingUtils.nameNotFoundException(NamingUti
ls.java:109)
  (... log messages removed ...)
```

Ceci signifie :

L'exception **`NameNotFoundException`** indique un problème de nommage JNDI. Les règles de nommage JNDI ont changé dans JBoss EAP 6, donc vous devrez modifier les noms de recherche pour qu'ils se conforment aux nouvelles règles.

Comment résoudre ceci :

Pour déboguer ceci, regardez plus haut dans le suivi du journal serveur quelle liaison JNDI a été utilisée. En regardant le journal de serveur vous verrez ceci :

```
15:01:16,138 INFO
[org.jboss.as.ejb3.deployment.processors.EjbJndiBindingsDeploymentUn
itProcessor] (MSC service thread 1-1) JNDI bindings for session bean
named RegisterAction in deployment unit subdeployment "jboss-seam-
booking.jar" of deployment "jboss-seam-booking.ear" are as follows:
  java:global/jboss-seam-booking/jboss-seam-
booking.jar/RegisterAction!org.jboss.seam.example.booking.Register
  java:app/jboss-seam-
booking.jar/RegisterAction!org.jboss.seam.example.booking.Register
  java:module/RegisterAction!org.jboss.seam.example.booking.Register
  java:global/jboss-seam-booking/jboss-seam-
booking.jar/RegisterAction
  java:app/jboss-seam-booking.jar/RegisterAction
  java:module/RegisterAction
  [JNDI bindings continue ...]
```

Il y a un total de huit liaisons INFO JNDI figurant dans le journal, un pour chaque bean de session : RegisterAction, BookingListAction, HotelBookingAction, AuthenticatorAction, ChangePasswordAction, HotelSearchingAction, EjbSynchronizations et TimerServiceDispatcher. Vous devez modifier le fichier **`lib/components.xml`** du WAR pour utiliser les nouvelles liaisons JNDI. Dans le journal, notez que toutes les liaisons EJB JNDI commencent avec "java: app / jboss-couture-booking.jar". Remplacer l'élément **`core:init`** comme suit :

```
<!--     <core:init jndi-pattern="jboss-seam-booking/#
{ejbName}/local" debug="true" distributable="false"/> -->
```

```
<core:init jndi-pattern="java:app/jboss-seam-booking.jar/#{ejbName}"
debug="true" distributable="false"/>
```

Ensuite, vous devrez ajouter les liaisons JNDI EjbSynchronizations et TimerServiceDispatcher. Ajouter les éléments de composants suivants au fichier :

```
<component class="org.jboss.seam.transaction.EjbSynchronizations"
jndi-name="java:app/jboss-seam/EjbSynchronizations"/>
<component class="org.jboss.seam.async.TimerServiceDispatcher" jndi-
name="java:app/jboss-seam/TimerServiceDispatcher"/>
```

Le fichier components.xml devrait ressembler à ce qui suit :

```
<?xml version="1.0" encoding="UTF-8"?>
<components xmlns="http://jboss.com/products/seam/components"
  xmlns:core="http://jboss.com/products/seam/core"
  xmlns:security="http://jboss.com/products/seam/security"
  xmlns:transaction="http://jboss.com/products/seam/transaction"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation=
    "http://jboss.com/products/seam/core
http://jboss.com/products/seam/core-2.2.xsd
     http://jboss.com/products/seam/transaction
http://jboss.com/products/seam/transaction-2.2.xsd
     http://jboss.com/products/seam/security
http://jboss.com/products/seam/security-2.2.xsd
     http://jboss.com/products/seam/components
http://jboss.com/products/seam/components-2.2.xsd">

    <!-- <core:init jndi-pattern="jboss-seam-booking/#
{ejbName}/local" debug="true" distributable="false"/> -->
    <core:init jndi-pattern="java:app/jboss-seam-booking.jar/#
{ejbName}" debug="true" distributable="false"/>
    <core:manager conversation-timeout="120000"
                  concurrent-request-timeout="500"
                  conversation-id-parameter="cid"/>
    <transaction:ejb-transaction/>
    <security:identity authenticate-method="#
{authenticator.authenticate}"/>
    <component
class="org.jboss.seam.transaction.EjbSynchronizations"
            jndi-name="java:app/jboss-seam/EjbSynchronizations"/>
    <component class="org.jboss.seam.async.TimerServiceDispatcher"
            jndi-name="java:app/jboss-
seam/TimerServiceDispatcher"/>
</components>
```

Redéployer l'application en effaçant le fichier **`standalone/deployments/jboss-seam-booking.ear.failed`** et en créant un fichier vide **`jboss-seam-booking.ear.dodeploy`** dans le même répertoire.

2. Problème - À ce moment là, l'application se déploie et exécute sans erreur. Quand vous accédez à l'URL http://localhost:8080/seam-booking/ dans un navigateur et que vous essayez de vous connecter, vous n'y parviendrez pas, et le message suivant "Login failed. Transaction failed." apparaîtra. Vous devriez en voir une trace dans la journalisation serveur :

```
13:36:04,631 WARN  [org.jboss.modules] (http-/127.0.0.1:8080-1)
Failed to define class
org.jboss.seam.persistence.HibernateSessionProxy in Module
"deployment.jboss-seam-booking.ear.jboss-seam.jar:main" from Service
Module Loader: java.lang.LinkageError: Failed to link
org/jboss/seam/persistence/HibernateSessionProxy (Module
"deployment.jboss-seam-booking.ear.jboss-seam.jar:main" from Service
Module Loader)
....
Caused by: java.lang.LinkageError: Failed to link
org/jboss/seam/persistence/HibernateSessionProxy (Module
"deployment.jboss-seam-booking.ear.jboss-seam.jar:main" from Service
Module Loader)
...
Caused by: java.lang.NoClassDefFoundError:
org/hibernate/engine/SessionImplementor
 at java.lang.ClassLoader.defineClass1(Native Method)
[rt.jar:1.7.0_45]
...
Caused by: java.lang.ClassNotFoundException:
org.hibernate.engine.SessionImplementor from [Module
"deployment.jboss-seam-booking.ear.jboss-seam.jar:main" from Service
Module Loader]
...
```

Ceci signifie :

L'exception ClassNotFoundException indique qu'il y a une bibliothèque manquante. Dans ce cas là, il s'agit de **`hibernate-core.jar`**.

Comment résoudre ceci :

Copier les JAR(s) **`hibernate-core.jar`** et le répertoire **`EAP5_HOME/seam/lib/`** dans le répertoire **`jboss-seam-booking.ear/lib`**.

Redéployer l'application en effaçant le fichier **`standalone/deployments/jboss-seam-booking.ear.failed`** et en créant un fichier vide **`jboss-seam-booking.ear.dodeploy`** dans le même répertoire.

3. Problème - l'application se déploie et s'exécute sans erreur. Lorsque vous accédez à l'URL http://localhost:8080/seam-booking/ dans un navigateur, vous êtes capable de vous connecter. Toutefois, lorsque vous essayez de réserver une chambre d'hôtel, vous pourrez voir une trace de l'exception.

   Pour déboguer ceci, vous devrez supprimer **`jboss-seam-booking.ear/jboss-seam-booking.war/WEB-INF/lib/jboss-seam-debug.jar`** qui masque l'erreur. Puis, vous devriez voir l'erreur suivante :

   ```
   java.lang.NoClassDefFoundError:
   org/hibernate/annotations/common/reflection/ReflectionManager
   ```

   Ceci signifie :

   L'exception NoClassDefFoundError indique qu'il manque une bibliothèque Hibernate.

   Comment résoudre ceci :

Copier les JAR(s) **hibernate-annotations.jar** et **hibernate-commons-annotations.jar** à partir du répertoire **EAP5_HOME/seam/lib/** dans le répertoire **jboss-seam-booking.ear/lib**.

Redéployer l'application en effaçant le fichier **standalone/deployments/jboss-seam-booking.ear.failed** et en créant un fichier vide **jboss-seam-booking.ear.dodeploy** dans le même répertoire.

4. Les erreurs de runtime et d'applications doivent être résolues

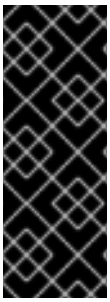
   A ce point dans le temps, l'application se déploie et exécute sans erreur.

   Pour retourner au sujet précédent, cliquer ici : Section 4.3.4, « Migrer l'archive du Seam 2.2 Booking dans JBoss EAP 6: Instructions étape par étape »

Rapporter un bogue

4.3.8. Récapitulatif des changements à effectuer lors d'une Migration de l'application Seam 2.2 Booking

Bien que ce serait beaucoup plus efficace de déterminer à l'avance les dépendances et ajouter les dépendances implicites en une seule étape, cet exercice montre comment les problèmes apparaissent dans le journal et donne des informations sur la façon de les déboguer et de les résoudre. Ce qui suit est un résumé des modifications apportées à la demande lors de sa migration vers JBoss EAP 6.



IMPORTANT

Les applications qui utilisent Hibernate directement avec Seam 2.2 utilisent sans doute une version d'Hibernate 3 empaquetée dans l'application. Hibernate 4, fourni par le module org.hibernate de JBoss EAP 6, n'est pas pris en charge par Seam 2.2. Cet exemple a pour but de vous aider à exécuter votre application dans JBoss EAP 6 comme première étape. Notez qu'empaqueter Hibernate 3 avec une application Seam 2.2 n'est pas une configuration prise en charge.

1. Vous avez créé un fichier **jboss-deployment-structure.xml** dans le répertoire **META-INF/** du EAR. Vous avez ajouté **<dependencies>** et **<exclusions>** pour résoudre **ClassNotFoundExceptions**. Ce fichier contient les données suivantes :

```
<jboss-deployment-structure xmlns="urn:jboss:deployment-
structure:1.0">
  <deployment>
     <dependencies>
       <module name="javax.faces.api" slot="1.2" export="true"/>
       <module name="com.sun.jsf-impl" slot="1.2" export="true"/>
       <module name="org.apache.commons.logging" export="true"/>
           <module name="org.dom4j" export="true"/>
       <module name="org.apache.commons.collections"
export="true"/>
     </dependencies>
  </deployment>
  <sub-deployment name="jboss-seam-booking.war">
   <exclusions>
       <module name="javax.faces.api" slot="main"/>
       <module name="com.sun.jsf-impl" slot="main"/>
```

```
      </exclusions>
      <dependencies>
        <module name="javax.faces.api" slot="1.2"/>
        <module name="com.sun.jsf-impl" slot="1.2"/>
      </dependencies>
    </sub-deployment>
</jboss-deployment-structure>
```

2. Vous avez copié les JAR suivants du répertoire ***EAP5_HOME*/jboss-eap-5.X/seam/lib/** (remplacer 5.X par la version EAP 5 à partir de laquelle vous allez migrer) dans le répertoire **jboss-seam-booking.ear/lib/** afin de résoudre **ClassNotFoundExceptions** :
   - hibernate-core.jar
   - hibernate-validator.jar
3. Vous avez modifié le fichier **jboss-seam-booking.jar/META-INF/persistence.xml** comme suit.
   1. Vous avez changé l'élément **jta-data-source** pour qu'il utilise la base de données Exemple fourni dans JBoss EAP 6 :

```
<!-- <jta-data-source>java:/bookingDatasource</jta-data-source> -
->
<jta-data-source>java:jboss/datasources/ExampleDS</jta-data-
source>
```

   2. Vous avez dé-commenté la propriété hibernate.cache.provider_class :

```
<!-- <property name="hibernate.cache.provider_class"
value="org.hibernate.cache.HashtableCacheProvider"/> -->
```

4. Vous avez modifié les fichiers **lib/components.xml** du WAR pour utiliser les nouvelles liaisons JNDI
   1. Vous avez remplacé l'élément existant **core:init** comme suit :

```
<!-- <core:init jndi-pattern="jboss-seam-booking/#
{ejbName}/local" debug="true" distributable="false"/> -->
<core:init jndi-pattern="java:app/jboss-seam-booking.jar/#
{ejbName}" debug="true" distributable="false"/>
```

   2. Vous avez ajouté des éléments de composants pour les liaisons JNDI "EjbSynchronizations" et les liaisons JNDI "TimerServiceDispatcher".

```
<component class="org.jboss.seam.transaction.EjbSynchronizations"
jndi-name="java:app/jboss-seam/EjbSynchronizations"/>
 <component class="org.jboss.seam.async.TimerServiceDispatcher"
jndi-name="java:app/jboss-seam/TimerServiceDispatcher"/>
```

Rapporter un bogue

ANNEXE A. HISTORIQUE DE RÉVISION

Version 6.3.0-24.1 **Mon Feb 16 2015** **Corina Roe**

Title Revision

Version 6.3.0-24 **Wednesday July 30 2014** **Sande Gilda**

Red Hat JBoss Enterprise Application Platform 6.3.0.GA