eBook Gratuit

APPRENEZ ejb

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

#ejb

Table des matières

À propos

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: ejb

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

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

Chapitre 1: Démarrer avec ejb

Remarques

Cette section fournit une vue d'ensemble de ce qu'est ejb et pourquoi un développeur peut vouloir l'utiliser.

Il devrait également mentionner tous les grands sujets dans ejb, et établir un lien avec les sujets connexes. La documentation de ejb étant nouvelle, vous devrez peut-être créer des versions initiales de ces rubriques connexes.

Examples

Configurer EJB avec JBoss AS 7.1

1. Vue d'ensemble

Dans cet article, nous allons discuter de la manière de démarrer avec Enterprise JavaBeans (EJB). Nous utiliserons JBoss AS 7.1.1.Final, mais vous êtes libre d'utiliser n'importe quel serveur de votre choix.

2. Dépendances Maven pour le haricot

Pour utiliser EJB, assurez-vous d'ajouter la dernière version à la section dépendances de votre fichier pom.xml:

```
<dependency>
    <groupId>org.jboss.spec.javax.ejb</groupId>
    <artifactId>jboss-ejb-api_3.2_spec</artifactId>
    <version>1.0.0.Final</version>
</dependency>
```

Assurez-vous d'ajouter correctement les dépendances JBoss car nous utiliserons JBoss comme serveur d'applications dans ce didacticiel. Dans la dernière partie du didacticiel, nous discuterons en détail de la configuration du build Maven pour le projet.

3. EJB Remote

Créons d'abord l'interface Bean appelée HelloWorldRemote.

```
public interface HelloWorldRemote {
    public String getHelloWorld();
}
```

Nous allons maintenant implémenter l'interface ci-dessus et la nommer `HelloWorldBean` .

```
@Stateless
```

```
public class HelloWorldBean implements HelloWorldRemote {

    public HelloWorldBean() {

    }

    @Override
    public String getHelloWorld(){
        return ("Hello World");
    }
}
```

Notez la notation `@Stateless` en haut de la déclaration de classe. Il désigne un bean session sans état.

4. Maven Setup pour Remote Bean

Dans cette section, nous allons discuter de la configuration de maven pour créer et exécuter l'application sur le serveur.

Regardons les plugins un par un.

4.1. Plugin du compilateur

Le plugin maven-compiler est utilisé pour compiler les sources de notre projet.

Ici, nous avons utilisé la version 2.3.1 du plugin avec le JDK source et cible défini sur 1.7 en configuration.

Nous avons défini ces paramètres comme des propriétés à l'intérieur de l'étiquette et les avons référencées via $ {property}.

```
<version.compiler.plugin>2.3.1</version.compiler.plugin>
<!-- maven-compiler-plugin -->
<maven.compiler.target>1.7</maven.compiler.target>
<maven.compiler.source>1.7</maven.compiler.source>
```

4.2 Le plugin EJB

Ce plugin génère un fichier Bean ainsi que le fichier client associé.

Nous avons spécifié la version ejb en 3.2 et la propriété generateClient est définie sur true, ce qui génère le client.

4.3 Déploiement dans JBoss

Le plugin jboss-as-maven est utilisé pour déployer, redéployer, annuler le déploiement ou exécuter l'application dans JBoss AS 7.

Dans cette configuration, nous spécifions le nom du fichier de construction identique à celui du fichier de construction du projet, qui est par défaut la forme artefactid-version dans notre cas `ejb-remote-1.0-SNAPSHOT` .

4.4 Dépendances Maven requises pour les EJB

jboss-javaee-6.0 définit la version des API Java EE 6 de JBoss à utiliser.

JBoss distribue un ensemble complet d'API Java EE 6, y compris une nomenclature.

Une nomenclature spécifie les versions d'une pile (ou d'une collection) d'artefacts. Nous spécifions cette balise afin que nous obtenions toujours les versions correctes des artefacts. Le type de cette dépendance elle-même est un pom qui contient les dépendances requises.

```
<dependency>
    <groupId>org.jboss.spec</groupId>
    <artifactId>jboss-javaee-6.0</artifactId>
    <version>${version.org.jboss.spec.jboss.javaee.6.0}</version>
    <type>pom</type>
    <scope>import</scope>
</dependency>
```

4.5 Annotations

Les éléments suivants obtiendront la dépendance aux annotations:

```
<dependency>
    <groupId>org.jboss.spec.javax.annotation</groupId>
    <artifactId>jboss-annotations-api_1.1_spec</artifactId>
    <scope>provided</scope>
</dependency>
```

4.6 EJB version 3.2

Dans le morceau de code suivant, nous obtenons la dernière version des spécifications:

```
<dependency>
    <groupId>org.jboss.spec.javax.ejb</groupId>
    <artifactId>jboss-ejb-api_3.2_spec</artifactId>
    <version>1.0.0.Final</version>
</dependency>
```

Pour exécuter le projet ci-dessus sur un serveur JBoss, nous devons d'abord exécuter:

```
mvn clean install
```

Ensuite, nous devons le déployer sur un serveur JBoss en cours d'exécution en exécutant la commande maven suivante:

```
jboss-as:deploy
```

Maintenant, vous devriez voir le fichier jar en cours de déploiement sur le serveur jboss.

Vous pouvez également copier le fichier jar disponible à partir du dossier cible du projet et le coller dans le dossier Webapp du serveur.

5. Configuration du projet client

Après avoir créé le bean distant, nous devons tester le bean déployé en créant un client.

Commençons par discuter de la configuration de Maven pour le projet.

5.1 Plugins Maven utilisés

Le plugin maven-compiler est utilisé pour compiler les sources de votre projet.

Nous avons spécifié la version de jdk 1.7 pour les classes source et cible.

Notre client est un programme Java. Pour l'exécuter, nous utilisons le `exec-maven-plugin` qui permet d'exécuter des programmes système et Java. Nous devons spécifier l'exécutable (c.-à-d. Java), classpath et la classe java (com.baeldung.ejb.client.Client).

Le classpath est laissé vide car le plugin inclut les arguments de classpath nécessaires en fonction des dépendances fournies.

5.2 Dépendances Maven pour le client EJB3

Pour exécuter le client EJB3, nous devons inclure les dépendances suivantes.

Nous dépendons des interfaces professionnelles distantes EJB de cette application pour exécuter le client. Nous devons donc spécifier la dépendance jj du client ejb. La balise avec la valeur «ejb-client» est utilisée pour spécifier la dépendance de ce projet sur le jar client EJB.

```
<dependency>
    <groupId>com.theopentutorials.ejb3</groupId>
    <artifactId>ejbmavendemo</artifactId>
    <type>ejb-client</type>
    <version>${project.version}</version>
</dependency>
```

Les dépendances `jboss-transaction-api_1.1_spec` , `jboss-ejb-api_3.1_spec` , `jboss-ejb-client` , `xnio-api` , `xnio-nio` , `jboss-remoting` , `jboss-sasl` , `jboss-marshalling-river` ont une portée d'exécution car ceux-ci sont requis temps d'exécution et pas pendant la compilation.

Les dépendances **jboss-javaee-6.0** et **jboss-as-ejb-client-bom** sous dependencyManagement ont une portée en tant qu'import. Ceci est utilisé pour inclure les informations de gestion des dépendances d'un POM distant dans le projet en cours. Ces POM distants sont fournis par JBoss, qui contient les dépendances nécessaires à l’exécution du client.

5.3 Propriétés du client JBoss EJB

Créez un fichier sous «src / main / resources» et nommez-le jboss-ejb-client.properties.

```
remote.connectionprovider.create.options.org.xnio.Options.SSL_ENABLED=false
remote.connections=default
remote.connection.default.host=localhost
remote.connection.default.port = 4447
remote.connection.default.connect.options.org.xnio.Options.SASL_POLICY_NOANONYMOUS=false
```

6. Création de la classe client

Nous créons d'abord une classe ClientUtility:

```
public class ClientUtility {
    private static Context initialContext;
    private static final String PKG_INTERFACES = "org.jboss.ejb.client.naming";

    public static Context getInitialContext() throws NamingException {
        if (initialContext == null) {
            Properties properties = new Properties();
            properties.put(Context.URL_PKG_PREFIXES, PKG_INTERFACES);
            initialContext = new InitialContext(properties);
         }
        return initialContext;
    }
}
```

Créons maintenant la classe Client réelle qui consommera le bean que nous avons déployé sur le serveur:

```
public class Client {

    //The lookup method to get the EJB name
    private static HelloWorldRemote doLookup() {
        Context context = null;
        HelloWorldRemote bean = null;
        try {
            // 1. Obtaining Context
            context = ClientUtility.getInitialContext();
            // 2. Generate JNDI Lookup name
            String lookupName = getLookupName();
            // 3. Lookup and cast
            bean = (HelloWorldRemote) context.lookup(lookupName);

        } catch (NamingException e) {
            e.printStackTrace();
        }
        return bean;
    }

    private static String getLookupName() {

         // The app name is the EAR name of the deployed EJB without .ear suffix.
         // Since we haven't deployed the application as a .ear, the app name for
         // us will be an empty string

        String appName = "";


         // The module name is the JAR name of the deployed EJB without the .jar
         // suffix.
        String moduleName = "ejb-remote-0.0.1-SNAPSHOT";


        // AS7 allows each deployment to have an (optional) distinct name. This
        // can be an empty string if distinct name is not specified.
        String distinctName = "";
```

```
        // The EJB bean implementation class name
        String beanName = "HelloWorldBean";

        // Fully qualified remote interface name
        final String interfaceName = "com.baeldung.ejb.tutorial.HelloWorldRemote";

        // Create a look up string name
        String name = "ejb:" + appName + "/" + moduleName + "/" + distinctName
                + "/" + beanName + "!" + interfaceName;

        return name;
    }
}
```

La classe Client consomme le bean et affiche le résultat.

7. Conclusion

Nous avons donc créé un serveur EJB et un client qui consomme le service. Le projet peut être exécuté sur n'importe quel serveur d'applications.

Lire Démarrer avec ejb en ligne: https://riptutorial.com/fr/ejb/topic/5704/demarrer-avec-ejb

Crédits

S. No	Chapitres	Contributeurs
1	Démarrer avec ejb	Community, Pritam Banerjee, RamenChef