

Enterprise Java Beans

Les EJB pour Enterprise Java Beans sont un standard de la plateforme JakartaEE pour créer des composants serveur. Cette architecture propose un conteneur pour créer des composants Java distribués hébergés sur un serveur d'application.

Il existe plusieurs types d'EJB.

Entity

Les EJB Entity permettent de représenter les données manipulées par l'application. On en parlera plus en détail dans le chapitre suivant sur JPA.

Message

Les EJB Message Driven servent à accomplir des tâches de manière asynchrone en utilisant des mécanismes de file de message.

Session

Les EJB session permettent de proposer des services avec ou sans état. Ils peuvent être injectés par le conteneur EJB à un service local (comme une servlet par exemple), mais ils peuvent aussi être appelés par un client distant, via le protocole RMI (Remote Method Invocation).

EJB Session

Interface

Pour définir un EJB session, il faut définir une interface qui définit les méthodes accessibles par le code client. Il faut y ajouter une interface pour signaler au conteneur EJB qu'il s'agit d'une interface d'EJB ; `@Remote` pour un accès par un client RMI distant, et `@Local` pour un accès par un composant local déployé sur le serveur également.

```
@Remote
public interface HelloWorldRemote {
    String helloWorld(String name);
}
```

```
@Local
public interface HelloWorldLocal {
    String helloWorld(String name);
}
```

Implémentation

On peut ensuite créer une classe EJB qui implémente cette interface. On doit utiliser une annotation pour signaler au conteneur qu'il s'agit d'une classe d'implémentation EJB. On utilise `@Stateless` pour un composant qui sera instancié à chaque demande. Le paramètre d'annotation `mappedName` permet de préciser un nom dans l'annuaire JNDI qui va référencer tous les EJB sur serveur.

```
@Stateless(mappedName="HelloWorld")
public class HelloWorld implements HelloWorldRemote {
    public String helloWorld(String name) {
        return "Hello, " + name;
    }
}
```

Pour avoir toujours la même instance du composant fournie par le conteneur pour une session de client donnée, on peut utiliser à la place `@Stateful`.

```
@Stateful(mappedName="HelloWorld")
public class HelloWorld implements HelloWorldRemote {
    public String helloWorld(String name) {
        return "Hello, " + name;
    }
}
```

Utilisation

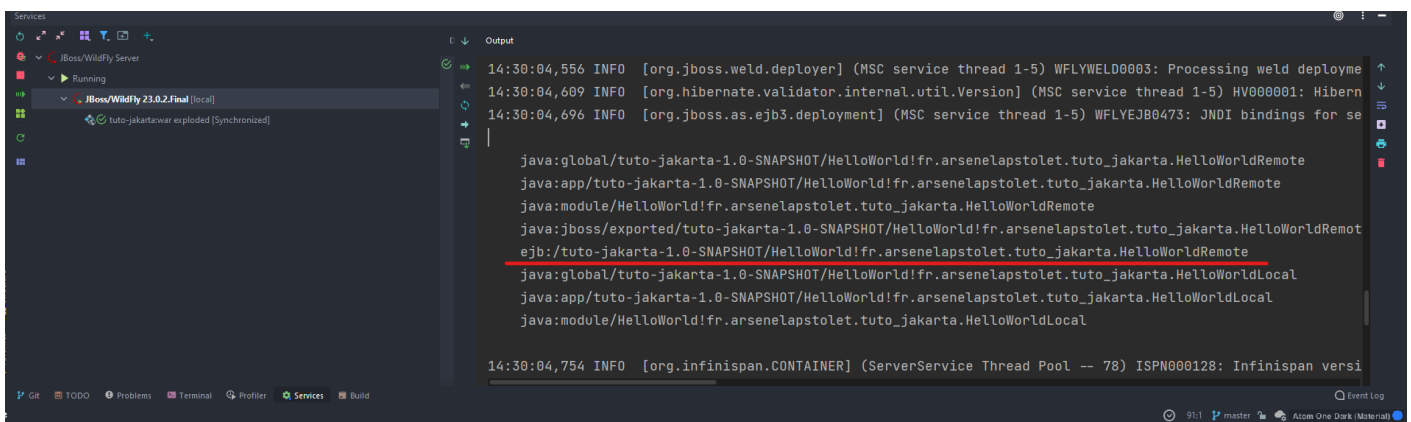
Pour demander au conteneur d'EJB d'injecter un EJB dans un autre composant, comme un autre EJB, ou une Servlet par exemple, utilisez l'annotation `@EJB` sur un attribut du type d'une interface

EJB locale :

```
@EJB
private HelloWorldLocal helloWorld;
```

Client RMI

On peut accéder aux EJB remote via la protocole RMI depuis un client distant Java. Après avoir lancé le serveur Wildfly qui possède des EJB Remote, on peut regarder ensuite les logs du serveur Wildfly dans l'interface d'IntelliJ IDEA, et y voir une ligne affirmant que l'EJB a bien été détecté et ajouté à l'annuaire JNDI. Copiez l'URL qui commence par EJB et gardez là dans un bloc note, elle sera utile plus tard.



Configuration du projet Client RMI

Pour le client EJB, créez un nouveau projet IntelliJ, cette fois en utilisant le template **Maven** et en utilisant l'archétype **Quickstart**. Une fois votre projet généré, ajoutez la dépendance suivante dans votre fichier `pom.xml` :

```
<dependency>
  <groupId>org.wildfly</groupId>
  <artifactId>wildfly-ejb-client-bom</artifactId>
  <version>21.0.0.Final</version>
  <type>pom</type>
</dependency>
```

Mettez à jours vos dépendances Maven, puis ajoutez une Run Config d'application Java normale.

Récupérer les EJB

Dans un premier temps, il faut copier le package qui contient les interfaces dans le projet client (les classes doivent avoir le même nom et le même package dans le projet client). Retirez dans le

projet client les annotations `@Remote` des interfaces.

Dans la méthode `main` ajoutez d'abord le code suivant, afin de configurer la connexion EJB :

```
final Hashtable<String, String> jndiProperties = new Hashtable<>();
jndiProperties.put(Context.INITIAL_CONTEXT_FACTORY, "org.wildfly.naming.client.WildFlyInitialContextFactory");
if(Boolean.getBoolean("http")) {
    jndiProperties.put(Context.PROVIDER_URL, "http://localhost:8080/wildfly-services");
} else {
    jndiProperties.put(Context.PROVIDER_URL, "remote+http://localhost:8080");
}
final Context context = new InitialContext(jndiProperties);
```

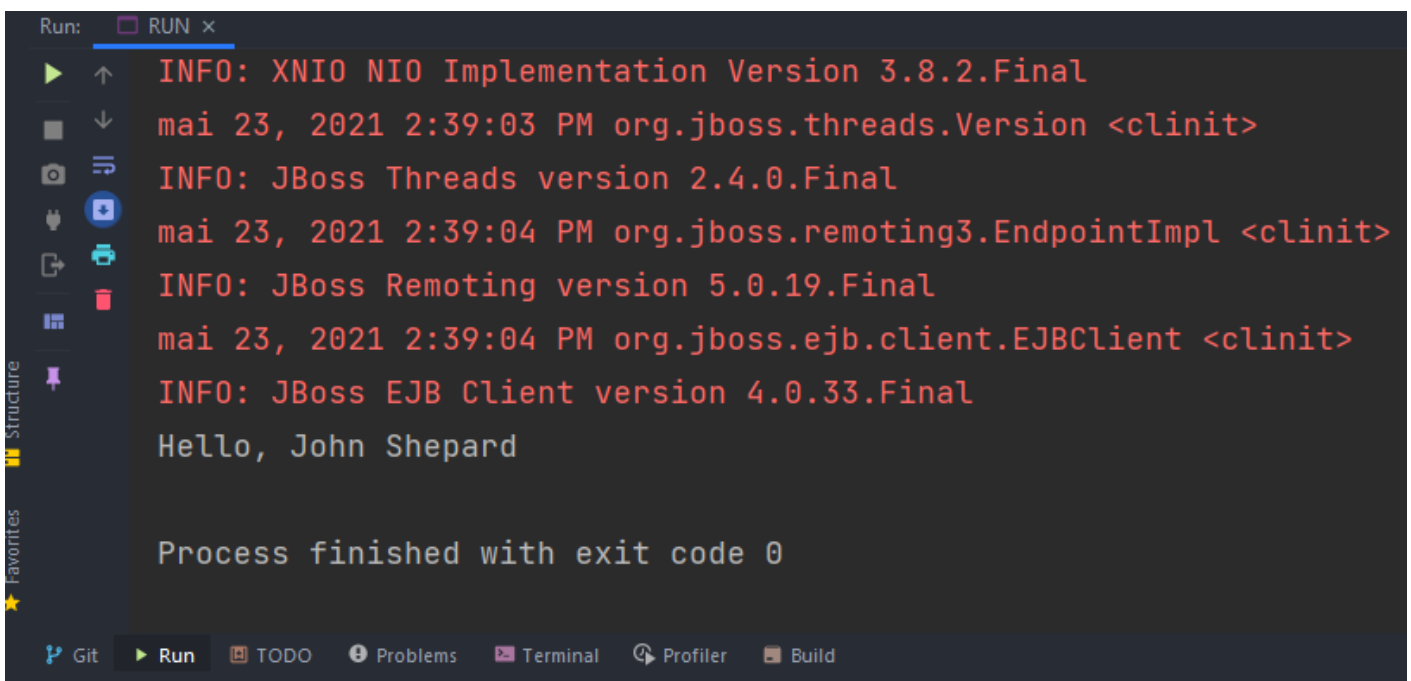
Ensuite pour récupérer votre EJB, ajouter ce code, en modifiant la string passée à la méthode `lookup()` par l'URL ejb que nous avons noté tout à l'heure :

```
final GestionContactRemote ejb = (GestionContactRemote) context.lookup("ejb:/tuto-jakarta-1.0-SNAPSHOT/HelloWorld!fr.arsenelapostolet.tuto_jakarta.HelloWorldRemote");
```

Vous pouvez enfin tester votre EJB :

```
System.out.println(ejb.helloWorld("John Shepard"));
```

Si tout s'est bien passé, vous devriez observer les logs du client EJB dans votre console ainsi que votre affichage :



```
Run: RUN x
INFO: XNIO NIO Implementation Version 3.8.2.Final
mai 23, 2021 2:39:03 PM org.jboss.threads.Version <clinit>
INFO: JBoss Threads version 2.4.0.Final
mai 23, 2021 2:39:04 PM org.jboss.remoting3.EndpointImpl <clinit>
INFO: JBoss Remoting version 5.0.19.Final
mai 23, 2021 2:39:04 PM org.jboss.ejb.client.EJBClient <clinit>
INFO: JBoss EJB Client version 4.0.33.Final
Hello, John Shepard

Process finished with exit code 0
```