59. Les EJB (Entreprise Java Bean)

Niveau : Supérieur

Les Entreprise Java Bean ou EJB sont des composants serveurs donc non visuels qui respectent les spécifications d'un modèle éditées par Sun. Ces spécifications définissent une architecture, un environnement d'exécution et un ensemble d'API.

Le respect de ces spécifications permet d'utiliser les EJB de façon indépendante du serveur d'applications J2EE dans lequel ils s'exécutent, du moment que le code de mise en oeuvre n'utilise pas d'extensions proposées par un serveur d'applications particulier.

Le but des EJB est de faciliter la création d'applications distribuées pour les entreprises.

Une des principales caractéristiques des EJB est de permettre aux développeurs de se concentrer sur les traitements orientés métiers car les EJB et l'environnement dans lequel ils s'exécutent prennent en charge un certain nombre de traitements tel que la gestion des transactions, la persistance des données, la sécurité, ...

Physiquement, un EJB est un ensemble d'au moins deux interfaces et une classe regroupées dans un module contenant un descripteur de déploiement particulier.

Pour obtenir des informations complémentaires sur les EJB, il est possible de consulter le site : http://www.oracle.com/technetwork/java/index-jsp-140203.html

Il existe plusieurs versions des spécifications des E.J.B. :

- 1.0 :
- 1.1 :
- 2.0 :
- 2.1 :
- 3.0 :

Remarque : dans ce chapitre, le mot bean sera utilisé comme synonyme d'EJB. Ce chapitre couvre essentiellement la version 2.x des EJB.

Ce chapitre contient plusieurs sections :

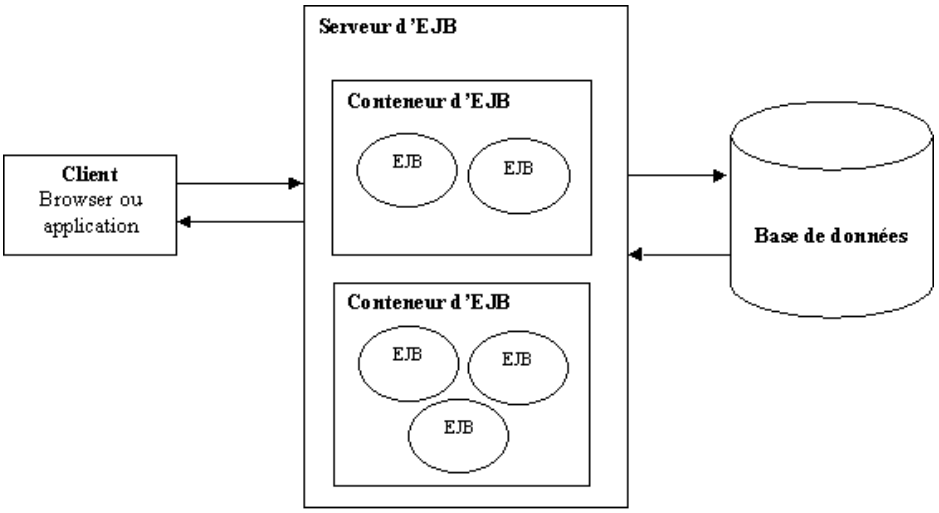
59.1. La présentation des EJB

Les EJB sont des composants et en tant que tels, ils possèdent certaines caractéristiques comme la réutilisabilité, la possibilité de s'assembler pour construire une application, etc ... Les EJB et les beans n'ont en commun que d'être des composants. Les JavaBeans sont des composants qui peuvent être utilisés dans toutes les circonstances. Les EJB doivent obligatoirement s'exécuter dans un environnement serveur dédié.

Les EJB sont parfaitement adaptés pour être intégrés dans une architecture trois tiers ou plus. Dans une telle architecture, chaque tier assure une fonction particulière :

- le client « léger » assure la saisie et l'affichage des données
- sur le serveur, les objets métiers contiennent les traitements. Les EJB sont spécialement conçus pour constituer de telles entités.
- une base de données assure la persistance des informations

Les EJB s'exécutent dans un environnement particulier : le serveur d'EJB. Celui-ci fournit un ensemble de fonctionnalités utilisées par un ou plusieurs conteneurs d'EJB qui constituent le serveur d'EJB. En réalité, c'est dans un conteneur que s'exécute un EJB et il lui est impossible de s'exécuter en dehors.



Le conteneur d'EJB propose un certain nombre de services qui assurent la gestion :

- du cycle de vie du bean
- de l'accès au bean
- de la sécurité d'accès
- des accès concurrents
- des transactions

Les entités externes au serveur qui appellent un EJB ne communiquent pas directement avec celui-ci. Les accès aux EJB par un client se font obligatoirement par le conteneur. Un objet héritant de la classe EJBObject assure le dialogue entre ces entités et les EJB en passant par le conteneur. L'avantage de passer par le conteneur est que celui-ci peut utiliser les services qu'il propose et libérer ainsi le développeur de cette charge de travail. Ceci permet au développeur de se concentrer sur les traitements métiers proposés par le bean.

Il existe de nombreux serveurs d'EJB commerciaux : BEA Weblogic, IBM Webpshere, Sun IPlanet, Macromedia JRun, Borland AppServer, etc ... Il existe aussi des serveurs d'EJB open source dont les plus avancés sont JBoss et Jonas.

59.1.1. Les différents types d'EJB

Il existe deux types d'EJB : les beans de session (session beans) et les beans entité (les entity beans). Depuis la version 2.0 des EJB, il existe un troisième type de bean : les beans orientés message (message driven beans). Ces trois types de bean possèdent des points communs notamment celui de devoir être déployés dans un conteneur d'EJB.

Les session beans peuvent être de deux types : sans état (stateless) ou avec état (stateful).

Les beans de session sans état peuvent être utilisés pour traiter les requêtes de plusieurs clients. Les beans de session avec état ne sont accessibles que lors d'un ou plusieurs échanges avec le même client. Ce type de bean peut conserver des données entre les échanges avec le client.

Les beans entité assurent la persistance des données. Il existe deux types d'entity bean :

- persistance gérée par le conteneur (CMP : Container Managed Persistence)
- persistance gérée par le bean (BMP : Bean Managed Persistence).

Avec un bean entité CMP (container-managed persistence), c'est le conteneur d'EJB qui assure la persistance des données. Un bean entité BMP (bean-managed persistence), assure lui-même la persistance des données grâce à du code inclus dans le bean.

La spécification 2.0 des EJB définit un troisième type d'EJB : les beans orientés message (message-driven beans).

59.1.2. Le développement d'un EJB

Le cycle de développement d'un EJB comprend :

- la création des interfaces et des classes du bean
- le packaging du bean sous forme de fichier archive jar

- le déploiement du bean dans un serveur d'EJB
- le test du bean

La création d'un bean nécessite la création d'au minimum deux interfaces et une classe pour respecter les spécifications de Sun : la classe du bean, l'interface remote et l'interface home.

L'interface remote permet de définir l'ensemble des services fournis par le bean. Cette interface étend l'interface EJBObject. Dans la version 2.0 des EJB, l'API propose une interface supplémentaire, EJBLocalObject, pour définir les services fournis par le bean qui peuvent être appelés en local par d'autres beans. Ceci permet d'éviter de mettre en oeuvre toute une mécanique longue et coûteuse en ressources pour appeler des beans s'exécutant dans le même conteneur.

L'interface home permet de définir l'ensemble des services qui vont permettre la gestion du cycle de vie du bean. Cette interface étend l'interface EJBHome.

La classe du bean contient l'implémentation des traitements du bean. Cette classe implémente les méthodes déclarées dans les interfaces home et remote. Les méthodes définissant celles de l'interface home sont obligatoirement préfixées par "ejb".

L'accès aux fonctionnalités du bean se fait obligatoirement par les méthodes définies dans les interfaces home et remote.

Il existe un certain nombre d'API qu'il n'est pas possible d'utiliser dans un EJB :

- les threads
- flux pour des entrées/sorties
- du code natif
- AWT et Swing

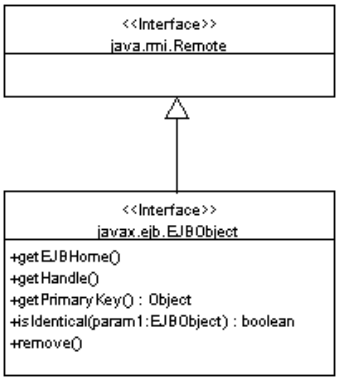
59.1.3. L'interface remote

L'interface remote permet de définir les méthodes qui contiendront les traitements proposés par le bean. Cette interface doit étendre l'interface javax.ejb.EJBObject.

Exemple :

```
1. package com.jmdoudoux.ejb;
2.
3. import java.rmi.RemoteException;
4. import javax.ejb.EJBObject;
5.
6. public interface MonPremierEJB extends EJBObject {
7.   public String message() throws RemoteException;
8. }
```

Toutes les méthodes définies dans cette interface doivent obligatoirement respecter les spécifications de RMI et déclarer qu'elles peuvent lever une exception de type RemoteException.



L'interface javax.ejb.EJBObject définit plusieurs méthodes qui seront donc présentes dans tous les EJB :

- EJBHome getEJBHome() throws java.rmi.RemoteException : renvoie une référence sur l'objet Home
- Handle getHandle() throws java.rmi.RemoteException : renvoie un objet permettant de sérialiser le bean
- Object getPrimaryKey() throws java.rmi.RemoteException : renvoie une référence sur l'objet qui encapsule la clé primaire d'un bean entité
- boolean isIdentical(EJBObject) throws java.rmi.RemoteException : renvoie un boolean qui précise si le bean est identique à l'instance du bean fournie en paramètre. Pour un bean session sans état, cette méthode renvoie toujours true. Pour un bean entité, la méthode renvoie true si la clé primaire des deux beans est identique.
- void remove() throws java.rmi.RemoteException, javax.ejb.RemoveException : cette méthode demande la destruction du bean. Pour un bean entité, elle provoque la suppression des données correspondantes dans la base de données.

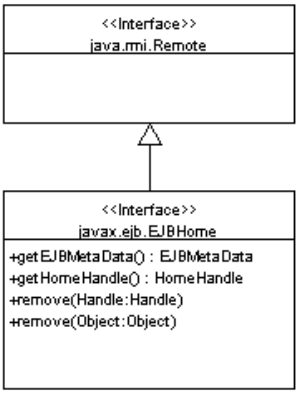
59.1.4. L'interface home

L'interface home permet de définir des méthodes qui vont gérer le cycle de vie du bean. Cette interface doit étendre l'interface EJBHome.

La création d'une instance d'un bean se fait grâce à une ou plusieurs surcharges de la méthode create(). Chacune de ces méthodes renvoie une instance d'un objet du type de l'interface remote.

Exemple :

```
01. package com.jmdoudoux.ejb;
02.
03. import java.rmi.RemoteException;
04. import javax.ejb.CreateException;
05. import javax.ejb.EJBHome;
06.
07. public interface MonPremierEJBHome extends EJBHome {
08.   public MonPremierEJB create() throws CreateException, RemoteException;
09. }
```



L'interface javax.ejb.EJBHome définit plusieurs méthodes :

- EJBMetaData getEJBMetaData() throws java.rmi.RemoteException
- HomeHandle getHomeHandle() throws java.rmi.RemoteException : renvoie un objet qui permet de sérialiser l'objet implémentant l'interface EJBHome
- void remove(Handle) throws java.rmi.RemoteException, javax.ejb.RemoveException : supprime le bean
- void remove(Object) throws java.rmi.RemoteException, javax.ejb.RemoveException : supprime le bean entité dont l'objet encapsulant la clé primaire est fourni en paramètre

La ou les méthodes à définir dans l'interface home dépendent du type d'EJB:

Type de bean	Méthodes à définir
bean session sans état	une seule méthode create() sans paramètre
bean session avec état	une ou plusieurs méthodes create()
bean entité	aucune ou plusieurs méthodes create() et une ou plusieurs méthodes finder()

59.2. Les EJB session

Les EJB session sont des EJB de service dont la durée de vie correspond à un échange avec un client. Ils contiennent les règles métiers de l'application.

Il existe deux types d'EJB session : sans état (stateless) et avec état (stateful).

Les EJB session stateful sont capables de conserver l'état du bean dans des variables d'instance durant toute la conversation avec un client. Mais ces données ne sont pas persistantes : à la fin de l'échange avec le client, l'instance de l'EJB est détruite et les données sont perdues.

Les EJB session stateless ne peuvent pas conserver de telles données entre chaque appel du client.

Il ne faut pas faire appel directement aux méthodes create() et remove() de l'EJB. C'est le conteneur d'EJB qui se charge de la gestion du cycle de vie de l'EJB et qui appelle ces méthodes. Le client décide simplement du moment de la création et de la suppression du bean en passant par le conteneur.

Une classe qui encapsule un EJB session doit implémenter l'interface javax.ejb.SessionBean. Elle ne doit pas implémenter les interfaces home et remote mais elle doit définir les méthodes déclarées dans ces deux interfaces.

La classe qui implémente le bean doit définir les méthodes de l'interface remote. La classe doit aussi définir les méthodes ejbCreate(), ejbRemove(),

ejbActivate(), ejbPassivate et setSessionContext().

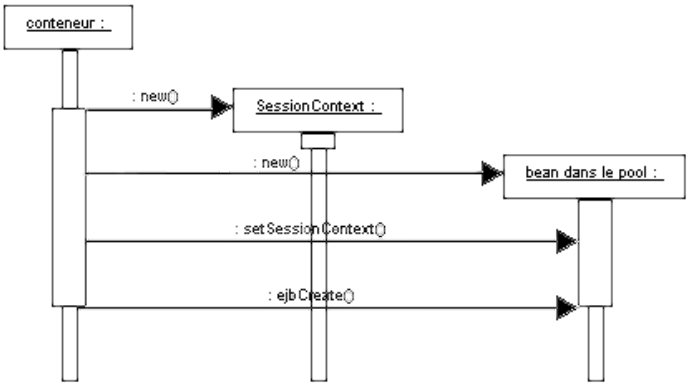
La méthode ejbRemove() est appelée par le conteneur lors de la suppression de l'instance du bean.

Pour permettre au serveur d'applications d'assurer la montée en charge des différentes applications qui s'exécutent dans ses conteneurs, celui-ci peut momentanément libérer de la mémoire en déchargeant un ou plusieurs beans. Cette action consiste à sérialiser le bean sur le système de fichiers et à le désérialiser pour sa remontée en mémoire. Lors de ces deux actions, le conteneur appelle respectivement les méthodes ejbPassivate() et ejbActivate().

59.2.1. Les EJB session sans état

Ce type de bean propose des services sous la forme de méthodes. Il ne peut pas conserver de données entre deux appels de méthodes. Les données nécessaires aux traitements d'une méthode doivent obligatoirement être fournies par le client en paramètre de la méthode.

Les services proposés par ces beans peuvent être gérés dans un pool par le conteneur pour améliorer les performances puisqu'ils sont indépendants du client qui les utilise. Le pool contient un certain nombre d'instances du bean. Toutes ces instances étant "identiques", il suffit au conteneur d'ajouter ou de supprimer de nouvelles instances dans le pool selon les variations de la charge du serveur d'applications. Il est donc inutile au serveur de sérialiser un EJB session sans état. Il suffit simplement de déclarer les méthodes ejbActivate() et ejbPassivate() sans traitements.



Le conteneur s'assure qu'un même bean ne recevra pas d'appel de méthode de la part de deux clients différents en même temps.

Exemple :

```
01. package com.jmdoudoux.ejb;
02.
03. import java.rmi.RemoteException;
04. import javax.ejb.EJBException;
05. import javax.ejb.SessionBean;
06. import javax.ejb.SessionContext;
07.
08.
09. public class MonPremierEJBBean implements SessionBean {
10.
11.   public String message() {
12.     return "Bonjour";
13.   }
14.
15.   public void ejbActivate() {
16.   }
17.
18.   public void ejbPassivate() {
19.   }
20.
21.   public void ejbRemove() {
22.   }
23.
24.   public void setSessionContext(SessionContext arg0) throws EJBException, RemoteException {
25.   }
26.
27.   public void ejbCreate() {
28.   }
29. }
```

59.2.2. Les EJB session avec état

Ce type de bean fournit aussi un ensemble de traitements grâce à ses méthodes mais il a la possibilité de conserver des données entre les différents appels de méthodes d'un même client. Une instance particulière est donc dédiée à chaque client qui sollicite ses services et ce tout au long du dialogue entre les deux entités.

Les données conservées par le bean sont stockées dans les variables d'instances du bean. Les données sont donc conservées en mémoire. Généralement, les méthodes proposées par le bean permettent de consulter et mettre à jour ces données.

Dans un EJB session avec état il est possible de définir plusieurs méthodes permettant la création d'un tel EJB. Ces méthodes doivent obligatoirement commencer par ejbCreate.

Les méthodes ejbPassivate() et ejbActivate() doivent définir et contenir les éventuels traitements lors de leur appel par le conteneur. Celui-ci appelle ces deux méthodes respectivement lors de la sérialisation du bean et sa désérialisation. La méthode ejbActivate() doit contenir les traitements nécessaires à la restitution du bean dans un état utilisable après la désérialisation.

Le cycle de vie d'un ejb avec état est donc identique à celui d'un bean sans état avec un état supplémentaire lorsque celui-ci est sérialisé. La fin du bean peut être demandée par le client lorsque celui-ci utilise la méthode remove(). Le conteneur invoque la méthode ejbRemove() du bean avant de supprimer sa référence.

Certaines méthodes métiers doivent permettre de modifier les données stockées dans le bean.

59.3. Les EJB entité

Ces EJB permettent de représenter et de gérer des données enregistrées dans une base de données. Ils implémentent l'interface EntityBean.

L'avantage d'utiliser un tel type d'EJB plutôt que d'utiliser JDBC ou de développer sa propre solution pour mapper les données est que certains services sont pris en charge par le conteneur.

Les beans entités assurent la persistance des données en représentant tout au partie d'une table ou d'une vue. Il existe deux types de bean entité :

- persistance gérée par le conteneur (CMP : Container Managed Persistence)
- persistance gérée par le bean (BMP : Bean Managed Persistence).

Avec un bean entité CMP (container-managed persistence), c'est le conteneur d'EJB qui assure la persistance des données grâce aux paramètres fournis dans le descripteur de déploiement du bean. Il se charge de toute la logique des traitements de synchronisation entre les données du bean et les données dans la base de données.

Un bean entité BMP (bean-managed persistence), assure lui-même la persistance des données grâce à du code inclus dans les méthodes du bean.

Plusieurs clients peuvent accéder simultanément à un même EJB entity. La gestion des transactions et des accès concurrents est assurée par le conteneur.

59.4. Les outils pour développer et mettre en oeuvre des EJB

La mise en oeuvre des EJB requiert un conteneur d'EJB généralement inclus dans un serveur d'applications et un IDE pour être productif.

59.4.1. Les outils de développement

Plusieurs EDI (Environnement de Développement Intégré) open source permettent de développer et de tester des EJB notamment Eclipse et Netbeans. Netbeans est d'ailleurs celui qui propose le plus rapidement une implémentation pour mettre en oeuvre la dernière version des spécifications relatives aux EJB.

59.4.2. Les conteneurs d'EJB

Il existe plusieurs conteneurs d'EJB commerciaux mais aussi d'excellents conteneurs d'EJB open source notamment Glassfish, JBoss ou Jonas.

59.4.2.1. JBoss

JBoss est un serveur d'applications Java EE open source écrit en Java.

Il peut être téléchargé sur www.jboss.org.

Pour l'installer, il suffit de décompresser l'archive et de copier son contenu dans un répertoire , par exemple : c\jboss

Pour lancer le serveur, il suffit d'exécuter la commande :

java -jar run.jar

Les EJB à déployer doivent être mis dans le répertoire deploy. Si le répertoire existe au lancement du serveur, les EJB seront automatiquement déployés dès qu'ils seront insérés dans ce répertoire.

59.5. Le déploiement des EJB

Un EJB doit être déployé sous forme d'une archive jar qui doit contenir un fichier qui est le descripteur de déploiement et toutes les classes qui composent chaque EJB (interfaces home et remote, les classes qui implémentent ces interfaces et toutes les autres classes nécessaires aux EJB).

Une archive ne doit contenir qu'un seul descripteur de déploiement pour tous les EJB de l'archive. Ce fichier au format XML doit obligatoirement être nommé ejb-jar.xml.

L'archive doit contenir un répertoire META-INF (attention au respect de la casse) qui contiendra lui-même le descripteur de déploiement.

Le reste de l'archive doit contenir les fichiers .class avec toute l'arborescence des répertoires des packages.

Le jar des EJB peut être inclus dans un fichier de type EAR.

59.5.1. Le descripteur de déploiement

Le descripteur de déploiement est un fichier au format XML qui permet de fournir au conteneur des informations sur les beans à déployer. Le contenu de ce fichier dépend du type de beans à déployer.

59.5.2. Le mise en package des beans

Une fois toutes les classes et le fichier de déploiement écrits, il faut les rassembler dans une archive .jar afin de pouvoir les déployer dans le conteneur.

59.6. L'appel d'un EJB par un client

Un client peut être une entité de toute forme : application avec ou sans interface graphique, un bean, une servlet ou une JSP ou un autre EJB.

Un EJB étant un objet distribué, son appel utilise RMI.

Le stub est une représentation locale de l'objet distant. Il implémente l'interface remote mais contient une connexion réseau pour accéder au skeleton de l'objet distant.

Le mode d'appel d'un EJB suit toujours la même logique :

- obtenir une référence qui implémente l'interface home de l'EJB grâce à JNDI
- créer une instance qui implémente l'interface remote en utilisant la référence précédemment acquise
- appel de la ou des méthodes de l'EJB

59.6.1. Un exemple d'appel d'un EJB session

L'appel d'un EJB session avec ou sans état suit la même logique.

Il faut tout d'abord utiliser un objet du type InitialContext pour pouvoir interroger JNDI. Cet objet nécessite qu'on lui fournisse des informations dont le nom de la classe à utiliser comme fabrique et l'url du serveur JNDI.

Cet objet permet d'obtenir une référence sur le bean enregistré dans JNDI. A partir de cette référence, il est possible de créer un objet qui implémente l'interface home. Un appel à la méthode create() sur cet objet permet de créer un objet du type de l'EJB. L'appel des méthodes de cet objet entraîne l'appel des méthodes de l'objet EJB qui s'exécute dans le conteneur.

Exemple :

```
01. package testEJBClient;
02.
03. import java.util.*;
04. import javax.naming.*;
```

```
05.
06. public class EJBClient {
07.
08.   public static void main(String[] args) {
09.     Properties ppt = null;
10.     Context ctx = null;
11.     Object ref = null;
12.     MonPremierBeanHome home = null;
13.     MonPremierBean bean = null;
14.
15.     try {
16.       ppt = new Properties();
17.       ppt.put(Context.INITIAL_CONTEXT_FACTORY, "org.jnp.interfaces.NamingContextFactory");
18.       ppt.put(Context.PROVIDER_URL, "localhost:1099");
19.       ctx = new InitialContext(ppt);
20.       ref = ctx.lookup("MonPremierBean");
21.       home = (MonPremierBeanHome) javax.rmi.PortableRemoteObject.narrow(ref,
22.         MonPremierBeanHome.class);
23.       bean = home.create();
24.       System.out.println("message = " + bean.message());
25.       bean.remove();
26.     }
27.     catch (Exception e) {
28.        e.printStackTrace();
29.     }
30.   }
31. }
```

59.7. Les EJB orientés messages

Ces EJB sont différents des deux types d'EJB précédents car ils répondent à des invocations de façon asynchrone. Ils permettent de réagir à l'arrivée de messages fournis par un M.O.M. (Middleware Oriented Messages).