Chapitre 5 : Enterprise JavaBeans (Objets Transactionnels Distribués)

1 Définition

Enterprise JavaBeans (EJB) est une architecture de composants logiciels côté serveur écrits en Java pour la plateforme de développement Java EE. Tous les EJB peuvent évoluer dans un contexte transactionnel[1].

Les EJB facilitent la création d'applications distribuées pour les entreprises :

- Adoptés par l'industrie
- Développement rapide d'applications
- Portable facilement
- S'occupent du traitement métier de l'application
- Permettent aux développeurs de se concentrer sur les traitements orientés métiers
- Sont réutilisables, assemblables

EJB signifie deux choses :

- Une spécification
- Un ensemble d'interfaces

2 Conteneur

Un conteneur EJB (Enterprise JavaBeans) fournit un environnement d'exécution pour les beans enterprise sur un serveur d'applications. Le conteneur gère tous les aspects du fonctionnement des beans enterprise sur le serveur d'applications et fait office d'intermédiaire entre la logique métier utilisateur dans le bean et le reste de l'environnement du serveur d'application. Un ou plusieurs modules EJB, contenant chacun un ou plusieurs beans entreprise, peuvent être installés dans un conteneur unique. On peut dire que le conteneur est le composant distribué.

Les appels aux méthodes par les clients de l'EJB sont interceptés par le conteneur d'EJB.

[1] En informatique, et particulièrement dans les bases de données, une **transaction** telle qu'une réservation, un achat ou un paiement est mise en œuvre via une suite d'opérations qui font passer la base de données d'un état A — antérieur à la transaction — à un état B postérieur[1] et des mécanismes permettent d'obtenir que cette suite soit à la fois *atomique , cohérente, isolée* et *durable (ACID).*

Le conteneur d'EJB offre de nombreux services au bean enterprise, notamment :

- Cycle de vie du bean
- Accès au bean (communication à distance)
- Sécurité d'accès
- Accès concurrents
- Lancement, validation et annulation des transactions en fonction des besoins.
- Synchronisation des données dans les variables de l'instance d'un bean *entity* avec les éléments de données correspondants stockés de manière persistante.

3 Architecture d'un composant EJB

Tout composant EJB est constitué d'une classe de bean et deux interfaces : L'interface EJB Home et L'interface EJB Object. Les interfaces servent comme intermédiaire entre le bean et le client, puisque le client n'a pas un accès direct au bean.

3.1 L'interface EJB Home

C'est l'interface avec laquelle les clients peuvent créer, supprimer, et rechercher les beans. C'est au moment du déploiement du composant au serveur EJB que l'objet correspondant à cette interface sera créé par le conteneur.

3.2 L'interface EJB Object

C'est l'interface avec laquelle les clients font appel aux méthodes du bean. C'est aussi au moment du déploiement du composant au server EJB que l'objet correspondant à cette interface sera créé par le conteneur. on peut dire que c'est l'interface d'utilisation des beans.

EJB Home et EJB Object peuvent être des interfaces locales et des interfaces distantes.

3.3 Interface locale

- Si l'EJB n'a qu'une seule interface locale, il ne peut être utilisé que par les classes (EJBs) qui sont dans le même serveur
- Le développeur peut ne fournir aucune interface ; en ce cas, une interface locale est automatiquement créée, qui contient toutes les méthodes publiques de l'EJB

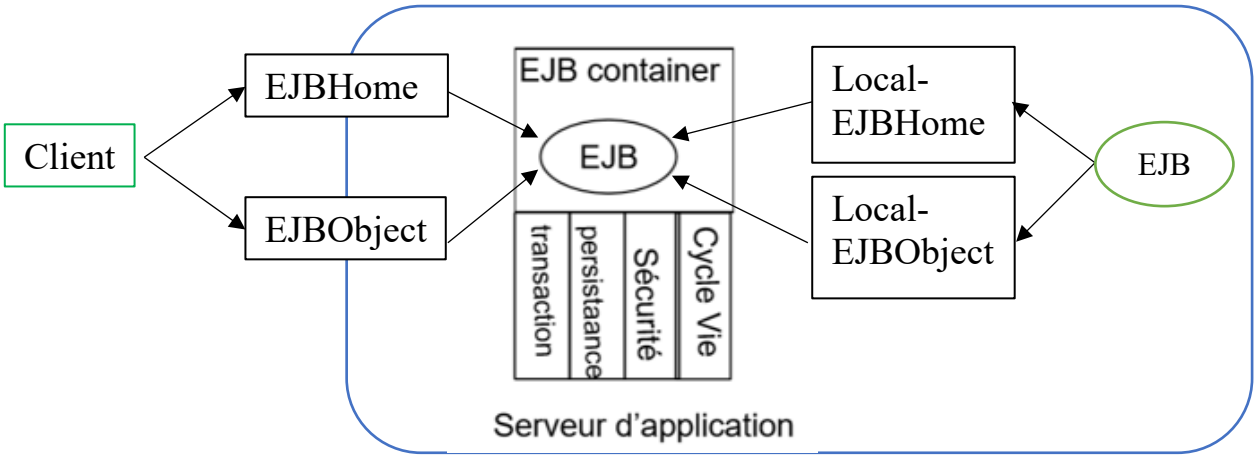

Figure 5.1- Architecture d'un composant EJB

3.4 Interface distante

- Indispensable si l'EJB peut être utilisé par des classes qui ne sont pas dans le même container (application distribuée)
- Pour manipuler un EJB à travers une interface distante, le serveur d'application utilisera RMI-IIOP, CORBA-IIOP, Messaging….

3.5 Classe du bean

C'est la classe du composant bean qui doit être fournie par le développeur du bean. Elle contient l'implémentation des méthodes publiées et non publiées à l'interface EJB Object.

Un descripteur de déploiement est un quatrième élément, c'est un fichier XML dont le développeur des beans, l'assembleur des beans, et le déployeur de l'application finale déclarent les attributs de transaction, de sécurité et des variables d'environnement.

4 Types de Beans

4.1 EJB session

Les EJB sessions sont des objets proposant des services à leur appelant. Ils proposent un certain nombre de méthodes écrites par le développeur. Il y a deux types d'EJB session : les EJB sessions ne conservant pas leur état entre deux appels (EJB dit « stateless »), et ceux le conservant (EJB dit « stateful »).

4.1.1 Bean sans état (Stateless)

La particularité principale d'un *Stateless Session Bean* est de ne pas conserver d'état entre les différents appels, après chaque appel, son état est réinitialisé. Donc ils sont partageables entre les clients.

- Exemple :

```
@Stateless
public class StatelessSessionBeanImpl implements StatelessSessionBean {

    public String sayHello() {
        return ("Hello!");
    }
}
```

4.1.2 Bean avec état(Stateful)

La particularité principale d'un Stateful Session Bean est de conserver son état entre différents appels de méthodes. donc ils ne sont pas partageables entre les clients, leur état n'est pas persistent (n'est pas stocké dans une base de données) mais il dure jusqu'à la terminaison d'une session client.

- Exemple

```
@Stateful
public class StatefulSessionBeanImpl implements StatefulSessionBean {

    private String user;

    public void login(String user) {
        this.user = user;
    }

    public String sayHello() {
        if (user == null) {
            return ("Hello world !");
        }
        return ("Hello " + user + " !");
    }
}
```

4.2 EJB *entité*

Les EJB *entité* sont des *beans* ayant majoritairement pour vocation d'être persistants, c'est-à-dire pouvant être stockés sur un support physique entre deux sessions. Chaque type de bean correspond à une table, et chaque instance de ce bean correspond à une entrée dans cette table. Si le serveur d'application tombe en panne, les beans entité peuvent être ré instanciés depuis la base de données.

Les EJB *entité* peuvent être de deux sortes : BMP (*Bean Managed Persistence*) ou CMP (*Container Managed Persistence*)

4.2.1 Bean Managed Persistence

Les EJB BMP sont des *beans* dont la persistance a dû être programmée par le développeur (ce dernier doit respecter un format pour la classe et les méthodes à implémenter sont imposées par la norme).

4.2.2 Container Managed Persistence

Les EJB CMP sont eux des *beans* dont la persistance est directement assurée par le conteneur d'EJB ; le mapping entre l'objet et son support de persistance est indiqué au conteneur via les fichiers descripteurs de déploiement. Le développeur, une fois le fichier de déploiement réalisé, n'a pas besoin d'écrire le code de persistance.

4.3 EJB message

Depuis la norme EJB 2.0 (2015) , cette architecture propose un troisième type de composant : les EJB message permettant de déclencher un processus côté serveur applicatif lors de la publication d'un message asynchrone. Pour ces composants, le client ne s'adresse pas directement aux composants mais publie un message sur un réceptacle JMS[2] (queue ou topic) configuré sur le serveur applicatif qui va alors déclencher l'activation par ce serveur d'une instance de l'EJB concerné pour pouvoir traiter ce message.

5 Évolution EJB

A partir de la spécification EJB2.0, les interfaces locales sont introduites qui sont Local-EJBHome et Local-EJBObject permettant une manipulation des beans non par des clients distants mais par d'autres beans appartenant au même serveur, ce qui optimise les appels à ces beans et augmente la performance.

De la version 1.0 à la version 2.1, un EJB était accompagné d'un ou plusieurs fichiers de déploiement écrits en XML qui permettait au serveur applicatif de déployer correctement l'objet au sein d'un conteneur. C'était notamment dans ces fichiers de déploiement que le développeur avait la possibilité de préciser le cadre transactionnel dans lequel l'objet allait s'exécuter. Depuis la version 3.0, le modèle EJB utilise le principe d'annotation Java (meta-données) pour spécifier toute la configuration et les propriétés transactionnelles de l'objet. Le fichier de code source de l'EJB se suffit à lui-même.

Depuis la version 3.0 de la spécification EJB, la notion de bean BMP/CMP n'existe plus : les EJB entité sont directement liés à la base de données via un mapping objet-relationnel. Ce mapping est défini soit dans un fichier de configuration XML, ou directement dans le code Java en utilisant des annotations. Cette nouvelle interface de programmation des EJB entité est appelée *Java Persistance API (JPA)*.

[2] L'interface de programmation Java Message Service (JMS) permet d'envoyer et de recevoir des messages de manière asynchrone entre applications ou composants Java. JMS permet d'implémenter une architecture de type MOM (message oriented middleware).

5.1 EJB 3

La plate-forme Java EE propose de mettre en oeuvre les couches métiers et persistance avec les EJB. Particulièrement intéressants dans des environnements fortement distribués, jusqu'à la version 3, leur mise en oeuvre est assez lourde sans l'utilisation d'outils tels que certains IDE. La version 3 des EJB vise donc à simplifier le développement et la mise en oeuvre des EJB qui sont fréquemment jugés trop complexes et trop lourds à mettre en oeuvre.

Cette simplification est rendue possible notamment par :

- Utilisation des annotations
- Mise en oeuvre de valeurs par défaut qui répondent à la plupart des besoins (configuration par exception)
- Descripteur de déploiement est facultatif
- Utilisation de POJO et de JPA pour les beans de type entity
- Injection de dépendances côté serveur mais aussi côté client (l'interface Home qui gérait le cycle de vie est abandonnée) qui remplace l'utilisation directe de JNDI

5.2 POJO

Les classes et les interfaces des EJB 3.0 sont de simples POJO ou POJI : ceci simplifie le développement des EJB. Par exemple, l'interface Home n'est plus à déclarer.

Il est toujours possible d'implémenter les interfaces SessionBean, EntityBean et MessageDrivenBean mais le plus simple est d'utiliser les annotations définies : @Stateless, @Stateful, @Entity ou @MessageDriven

```
@Remote
public interface Convertisseur {
  public CompteBancaire find(int id);
}
```

```
@Stateless
public class ConvertisseurBean
       implements Convertisseur {
...
}
```

Figure 5.2- Interface distante et leur implémentation dans une *Bean stateless* en utilisant les annotations

Il est possible de définir une interface métier pour l'EJB ou de laisser générer cette interface lors du déploiement. Dans le premier cas, il n'est plus nécessaire qu'elle implémente l'interface EJBObject ou EJBLocalObject mais il faut simplement utiliser les annotations définies : @Remote ou @Local.

Dans le second cas, ces annotations doivent être utilisées dans la classe d'implémentation pour permettre de déterminer l'interface générée. Il est possible de définir une interface locale et/ou distante pour un même EJB.

- Exemple

```
@Remote
@Stateless
public class HelloWorldBean {
  public String saluer(String nom)
  {
    return "Bonjour "+nom;
  }
}
```

Il n'est pas recommandé de laisser les interfaces être générées par le conteneur pour plusieurs raisons :

- Les interfaces générées exposent par défaut toutes les méthodes de l'EJB
- L'interface est utilisée par le client pour invoquer l'EJB
- Le nom des interfaces générées utilise le nom de l'implémentation de l'EJB