EJB

Enterprise Java Beans

Préparé par
Pr. M.G. BELKASMI

Introduction

- Les EJB sont utilisés dans le cadre de l'architecture JEE (Java Enterprise Edition).
- Les EJB sont des objets distants utilisant RMI/IIOP

  → garantir l'usage des EJB via un quelconque code client.
- Ces composants permettent de mettre en œuvre la logique métier d'une application codée suivant une architecture logicielle 3 tiers (séparation entre présentation, métier et données).

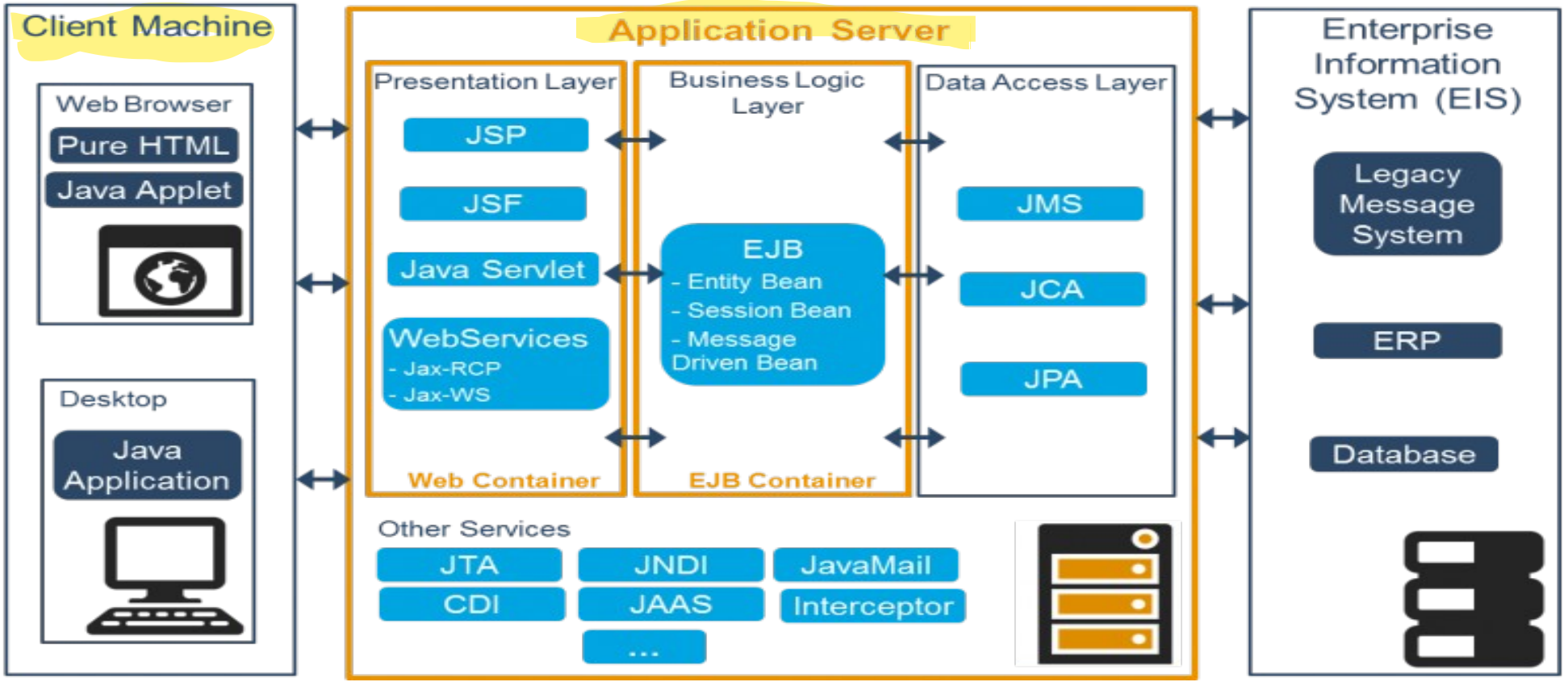
RMI : Remote Method Invocation
IIOP: Internet Inter-ORB Protocol (ORB : Object Request Broker)

Introduction

- La couche de présentation :
  - des clients légers (navigateur Web) : les servlets et les pages JSP sont utilisés pour mettre en œuvre cette couche logiciel. On parle bien entendu d'application Intranet/Internet.
  - des clients lourds : une application Java graphique (AWT et/ou SWING), du moins si cette partie est codée en Java.
- Dans les deux cas, la partie cliente des EJB utilisera l'API JNDI pour localiser les objets distants EJB.

JNDI : Java Naming and Directory Interface

Architecture 3-tiers Java EE



Topologie des composants EJB

On compte trois "familles" (catégories) de composants EJB :

- Les EJB **Session Beans** sont utilisés pour représenter les données d'un client et les comportements qui leur sont associés.
  - ≠ session : un conteneur de servlets ne gère pas l'activation/passivation de vos sessions. Le conteneur d'EJB oui. Plus la gestion de la sécurité et de la notion de transactions.
- Les EJB **Entity Bean** permettent de représenter une donnée persistante ayant une existence au niveau d'un support de stockage telle une base de données relationnelle.
- Les EJB **Messages Beans** permettent la mise en œuvre d'applications basées sur un traitement asynchrone de messages.
  - Faut s'appuyer sur un service JMS (Java Messages Service). Ils sont apparus avec le standard EJB 2.0.

Topologie des composants EJB

Les EJB sessions, deux sous-catégories existent :

- Les EJB Session avec état (**stateful**) :
  - maintient des données pour un client en mémoire (caddie virtuel).
  - il y a donc autant d'objets Session avec états instanciés que d'utilisateurs.
  - En cas de défaut de mémoire, le conteneur d'EJB sait passiver les EJB momentanément inutilisés et les activer le cas échéant.
- Les EJB Session sans états (**stateless**) :
  - utilisés pour représenter un client le temps d'un traitement.
  - Seul le comportement de l'EJB nous intéresse.
  - le conteneur peut gérer un pool de composants EJB stateless

Topologie des composants EJB

Les EJB entités, deux sous-catégories :

- Les EJB entités CMP (Container Managed Persistance) délèguent la persistance des données au conteneur d'EJB. Aucun code JDBC n'est à fournir.
- Les EJB entités BMP (Bean Managed Persistance) prennent en charge l'accès aux bases de données. Le conteneur décide malgré tout des moments auxquels l'EJB se synchronise avec la base.

Mise en œuvre d'un EJB 2.0

Tout EJB est, au moins, constitué d'une unique classe, des interfaces distantes (tout comme avec RMI) :

- Une interface qualifiée "d'interface de fabrique" :
  - permet la construction d'un composant EJB (factory).
- Une interface qualifiée "d'interface métier" :
  - permet de définir les services que propose le composant EJB.
- Le bean à proprement parlé qui implémente (**dans une certaine mesure**) les deux interfaces précédentes.
- Le descripteur de déploiement (structuré en XML):
  - ne fait pas réellement partie du bean
  - contient des informations utiles pour le conteneur d'EJB qui va recevoir le bean (nature, aspects liés à la sécurité, gestion des transactions.)

Exemple :un compte bancaire "*Account*"

- Nous allons considéré un bean permettant de représenter un compte bancaire (associé à un utilisateur).
- Nous considérerons que ce bean n'est pas persistant.
- Nous appellerons ce bean "*Account* "

Exemple :un compte bancaire "*Account*"

L'interface de fabrique *AccountHome*

- Étend *EJBHome*, qui dérive elle-même de l'interface *Remote* (définie elle dans le package *java.rmi.**).
- L'interface *EJBHome* est, bien entendu, stockée dans le package *javax.ejb*
- Le nom d'une méthode de création est obligatoirement *create* (Surcharge possible).
- l'invocation de cette méthode peut échouer :
  - soit pour de raisons liées au réseaux (*RemoteException*)
  - soit pour de histoire de sécurité (*CreateException*)

Exemple :un compte bancaire "*Account*"

L'interface de fabrique *AccountHome*

```
import java.rmi.*;
import javax.ejb.*;

public interface AccountHome  extends  EJBHome {
    public Account create()
                throws RemoteException, CreateException;
}
```

Exemple :un compte bancaire "*Account*"

L'interface métier *Account*

- Étend *EJBObject* dérivant de *Remote*.
- définit les méthodes métiers que va exposer le bean d'entreprise.
- Conventionnellement, cette interface porte le nom du bean. Elle pourrait aussi être nommée *AccountRemote.*

Exemple :un compte bancaire "*Account*"

L'interface métier *Account*

```
import java.rmi.*;
import javax.ejb.*;

public interface Account extends EJBObject {

    public void add(double value) throws RemoteException;
    public void sub(double value) throws RemoteException;

}
```

Exemple :un compte bancaire "*Account*"

La classe d'implémentation du bean

```
import java.rmi.*;
import javax.ejb.*;

public class AccountBean implements SessionBean {
    private double currentValue;

    public void add(double value) { this.currentValue += value; }
    public void sub(double value) { this.currentValue -= value; }

    public AccountBean() {}
    public void ejbCreate() {  this.currentValue = 0; }
    public void ejbRemove() {}
    public void ejbActivate() {}
    public void ejbPassivate() {}
    public void setSessionContext(SessionContext sc) {}
}
```

Exemple :un compte bancaire "*Account*"

La classe d'implémentation du bean

- il n'y a pas d'erreur ou d'oublie
  - chaque méthode ***create*** de l'interface locale (home) se doit d'être renommée **ejbCreate** dans la classe de bean.
  - elle doit avoir le même prototype (à l'exception du type de retour) que son homologue dans l'interface locale.
  - Les méthodes **ejbActivate** et **ejbPassivate** sont utilisées pour sauver, momentanément, l'état de l'objet.

Exemple :un compte bancaire "*Account*"

La classe d'implémentation du bean

- Il ne s'agit à priori pas d'un objet distant. Comment cela marche t-il ?
  - On a pas à coder l'intégralité du code nécessaire au bean.
  - JEE permet de spécifier des choses en rapport aux transactions et à la sécurité. Ces informations ne seront pas codées, mais spécifiées dans le descripteur de déploiement (le fichier XML).
  - Lors du déploiement le serveur se chargera de générer le code manquant.

Exemple :un compte bancaire "*Account*"

La classe d'implémentation du bean

- Lors du déploiement, deux nouvelles classes vont être générées :
  - La **classe de fabrique** : elle implémentera l'interface de fabrique et sera à proprement parlé une classe d'objets distants. Seule une instance sera utilisée à partir de cette classe. Cette instance sera enregistrée dans un service de noms et servira à construire les beans, tout en garantissant le support de sécurité et de transaction.

Exemple :un compte bancaire "*Account*"

La classe d'implémentation du bean

- Lors du déploiement, deux nouvelles classes vont être générées :
  - La **classe de façade** : elle implémentera l'interface métier distante. En fait c'est une instance de cette classe que vous manipulerez réellement. Cette instance prendra en charge les aspects de sécurité et si vous êtes autorisé, elle invoquera la méthode souhaitée sur votre classe de bean. C'est pour cette raison que votre classe de bean n'est pas une classe d'objets distants : elle est accédée au sein de la JVM du conteneur d'EJB par un objet de façade.

Instanciation et manipulation d'un EJB

```
import javax.rmi.*;
import javax.naming.*;
//...
// Récupération du context initial JNDI
    Context initial = new InitialContext();

// Localisation, à partir du contexte JNDI, de l'unique
// instance de construction pour ce type de bean (d'EJB).
    AccountHome home = (AccountHome)PortableRemoteObject.narrow(
                        initial.lookup("Account"), AccountHome.class );

// Création d'un bean session pour l'utilisateur à partir
// de la fabrique
Account acc = home.create();

// Utilisation de l'EJB
acc.add(200);
acc.sub(100);
```

Instanciation et manipulation d'un EJB

- On commence par localiser l'objet de fabrique via JNDI.
  - L'enregistrement de l'objet distant de fabrique à été réalisé au moment du déploiement de l'application sur le serveur.
  - Cet objet est la seule instance de la classe de fabrique.
  - Toute personne souhaitant obtenir un EJB de ce type doit passer pas cette instance.
  - Conventionnellement, c'est le nom du bean qui est choisit pour l'enregistrement JNDI.
- Demander une nouvelle instance du bean considéré.
  - Soit tout se déroule bien, soit une exception sera générée.
- Une fois l'objet disponible, (il s'agit de la façade), invoquer le comportement (les méthodes) requis.

Empaquetage et déploiement des applications

- les composants EJB se doivent d'être empaquetés dans des archives Java pour pouvoir être déployés sur un serveur d'applications JEE.
- Les EJB sont stockés dans une archive d'objet métier (d'extension "*.jar*").
- Un descripteur de déploiement est bien entendu requis : les outils de développement que vous utiliserez savent générer ces fichiers. Cette archive doit de plus être stockée dans une archive d'entreprise d'extension "*.ear*" (Enterprise ARchive).
- Au final, c'est ce fichier que vous déploierez sur le serveur d'applications.

Développement d'EJB 3
cas d'un EJB Session

- La création d'un EJB Session est largement simplifiée par la spécification EJB 3 :
  - il n'y a pas de contrainte d'héritage
  - aucune méthode particulière à implémenter.
    Le développement d'un EJB 3 Session se fait en trois étapes:
    - Définition de l'interface du composant
    - Implémentation du composant
    - Ajout des annotations EJB 3

Développement d'EJB 3
cas d'un EJB Session

- Définition de l'interface du composant

```
package com.et;

public interface PremierEJB3 {
    public String ditBonjour(String aQui);
}
```

Développement d'EJB 3
cas d'un EJB Session

- Implémentation du composant

```
package com.et;

public class PremierEJB3Bean implements PremierEJB3 {

    public String ditBonjour(String aQui) {
        return "Bonjour " + aQui + " !!!";
    }

}
```

Développement d'EJB 3
cas d'un EJB Session

Ajout des annotations EJB 3

- Code de l'interface après l'ajout de l'annotation @Remote :

```
package com.et;
import javax.ejb.Remote;

@Remote
public interface PremierEJB3 {

    public String ditBonjour(String aQui);
}
```

Développement d'EJB 3
cas d'un EJB Session

Ajout des annotations EJB 3

- Code de la classe après l'ajout de l'annotation @Stateless :

```
package com.et;
import javax.ejb.Stateless;

@Stateless
public class PremierEJB3Bean implements PremierEJB3 {
    public String ditBonjour(String aQui) {
        return "Bonjour " + aQui + " !!!";
    }
}
```

Développement d'EJB 3
cas d’un EJB Session

Tester l'EJB Session

Le code de l’EJB étant écrit, on peut le tester.

- il faut d'une part déployer le projet EJB
- et d'autre part écrire une application cliente.

Développement d'EJB 3
cas d'un EJB Session

- **Tester l'EJB Session :** la classe de test suivante pour invoquer

```
package com.et;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;

public class ClientPremierEJB3 {
    public static void main(String[] args) {
     try{
         Context context = new InitialContext();
         PremierEJB3 beanRemote =
                 (PremierEJB3)context.lookup("PremierEJB3Bean/remote");
         System.out.println(beanRemote.ditBonjour("ClientPremierEJB3"));
     }catch(NamingException e){
         e.printStackTrace();
     }
    }
}
```

Développement d'EJB 3
cas d'un EJB Entity

- La spécification EJB 3 revoit très largement le développement des Entity Beans.
  - Exercice : refaire l'EJB Account en version EJB3
- Les EJB Entity sont décrits dans une spécification complémentaire nommée JPA (Java Persistence API)

Développement d'EJB 3
cas d’un EJB Entity

les principaux apports de la JPA sont :

- Simplification du code via l'approche '**POJO**' (Plain Old Java Object) :
  - un EJB Entity consiste en une classe Java standard
    - pas de contrainte d'héritage,
    - pas de méthode particulière à implémenter

Développement d'EJB 3
cas d’un EJB Entity

les principaux apports de la JPA sont :

- Fonctionnalités de mapping objet-relationnel plus riches :
  - précédemment le mapping objet-relationnel n'était pas standardisée
  - JPA propose d'utiliser les annotations pour définir le mapping.
  - JPA aborde de façon complète et pragmatique le problème de persistance
  - La spécification JPA standardise l'utilisation d'un fichier, nommé **persistence.xml**

Développement d'EJB 3
cas d’un EJB Entity

les principaux apports de la JPA sont :

- Utilisation possible en dehors d'un serveur d'applications JEE : JPA peut être utilisée dans une application cliente, la présence d'un conteneur d'EJB n'est plus nécessaire.
  - Des solutions intermédiaires sont aussi possibles : déploiement dans Tomcat d'une application à base de Servlets et JSP utilisant JPA.