Les EJBs (Enterprise Java Beans)

Samia Bouzefrane
Maître de Conférences
Laboratoire CEDRIC
Conservatoire National des Arts et Métiers
292 rue Saint Martin
75141 Paris Cédex 03

samia.bouzefrane@cnam.fr
http://cedric.cnam.fr/~bouzefra

Bibliographie

EJB 2.0 : Mise en œuvre, Christophe Calandreau, Alain Fauré
Nader Soukouti, *Ed. Dunod*, 2002, ISBN : 2 10 004 729 9.

L'environnement J2EE : principes, fonctions, utilisation
P. Déchamboux, *École d'été sur les Intergiciels et sur la Construction d'Applications Réparties*, ICAR'2003,
http://sardes.inrialpes.fr/ecole/2003/

Enterprise java Bean
Lionel Seinturier, Université Pierre & Marie Curie, octobre 2003
http://www-src.lip6.fr/homepages/Lionel.Seinturier/middleware/ejb.pdf

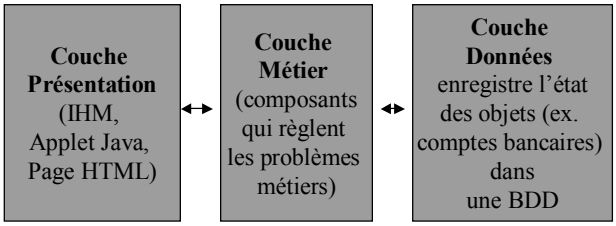
Programmation Java côté Serveur : **Servlets, JSP et EJB**,
Andrew Patzer, *Ed. Eyrolles*, 2000, ISBN : 2 212 09109 5.

Tutorial J2EE de Sun : http://java.sun.com/j2ee/1.4/docs/tutorial/doc/index.html

Architecture des Systèmes d'Informations

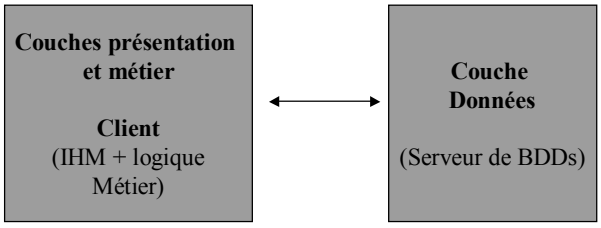
- accès aux données
- traitement des données
- présentation des données

Architecture en couches



Cette architecture sert à isoler la logique métier de l'interface graphique et à interdire un accès direct aux données.
Avantage : modifier une couche sans toucher aux autres.

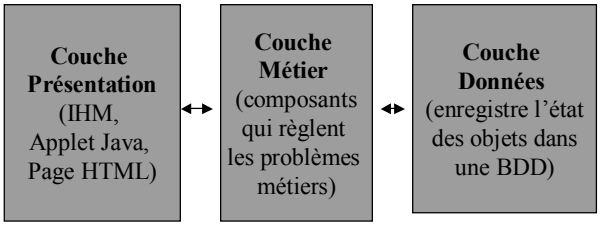
Architecture Client/Serveur (à deux niveaux)



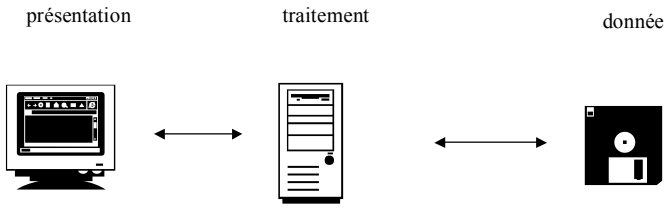
Client lourd

Architecture à 3 niveaux/1

Les 3 couches sont physiquement séparées : on peut même décomposer une couche en sous-couches (processus) => architecture à n niveaux



Architecture à 3 niveaux/2



client

Serveur d'applications: héberge des applications à base de composants EJB (classes java conformes au modèle EJB) et des composants Web (servlets, JSP)

SGBD : fournit un support de stockage pour les données de l'application

Middlewares et Architectures distribuées

Manipuler des objets => il est indispensable d'utiliser une infrastructure technique
Pour faire communiquer ces objets :
L'infrastructure doit :

- fournir des services de nommage
- fournir des services de sécurité.

➔ CORBA,RMI

Inconvénients d'utiliser des middlewares :

- portabilité côté serveur quasi-nulle
- le développeur gère le cycle de vie des objets
- le développeur doit optimiser l'accès et l'utilisation des ressources (exemples: connexion aux BDDs et manipulation de threads)

⏩ Ces tâches seront automatisées : elles sont plutôt déclarées au lieu d'être programmées.

Architecture J2EE (Java 2 Enterprise Edition)

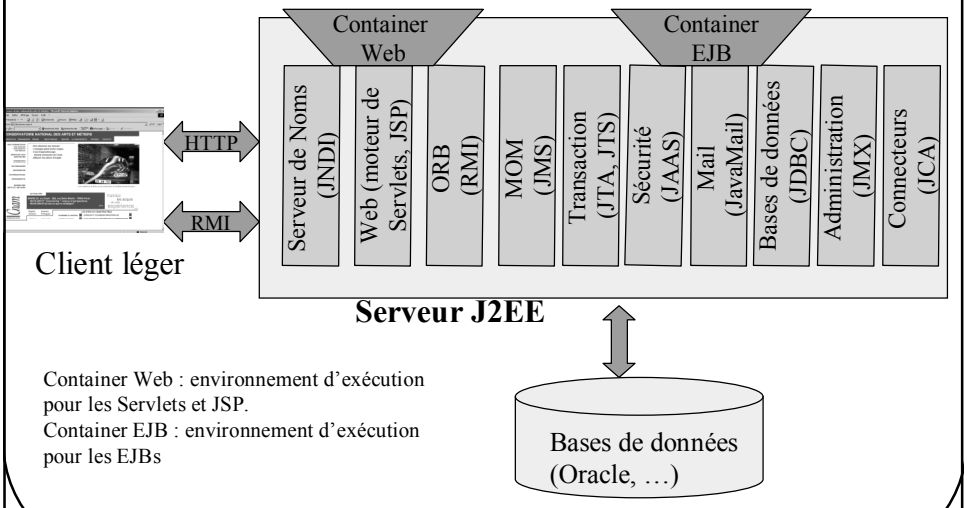
L'architecture J2EE de Sun a été proposée pour palier à ces inconvénients.

L'architecture J2EE définit :

- un middleware basé sur **RMI/IIOP**
- des objets Java distribués : **EJB** (Enterprise Java Beans)

⏩ L'architecture J2EE fournit des services techniques pour permettre au développeur de se concentrer sur la logique métier.

Plate-forme J2EE

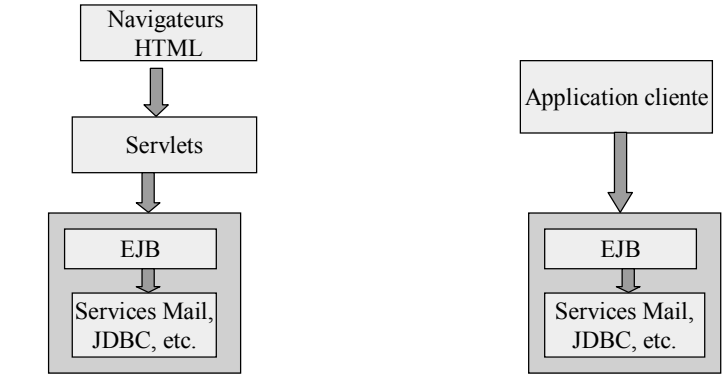


Container Web : environnement d'exécution pour les Servlets et JSP.
Container EJB : environnement d'exécution pour les EJBs

Fonctions couvertes par J2EE

- La communication entre objets distribués avec Java RMI et RMI/IIOP
- La création d'objets distribués transactionnels avec EJB
- La création et la recherche de références d'objets distants avec JNDI (Java Naming and Directory Interface)
- L'accès aux Bases de données avec JDBC (Java DataDase Connectivity)
- La gestion des transactions avec JTA (Java Transaction API) et JTS (Java Transaction Service)
- La communication asynchrone par messages avec JMS (Java Message Service)
- La réalisation d'interfaces graphiques Web avec les pages JSP (JavaServer Pages) et les Servlets
- L'intégration des objets CORBA avec JavaIDL
- l'envoi de courriers électroniques avec JavaMail
- La description du comportement des composants Java en XML

Configurations possibles pour une application J2EE



Client léger : navigateur Web

Client lourd

Les acteurs d'une application J2EE

Différents niveaux de responsabilité :

- Le fournisseur des EJBs : des composants métier réutilisables (par achat ou développement interne)
- L'assembleur d'applications : l'acteur qui construit une application à partir d'EJBs existants
- Le déployeur : l'acteur qui récupère l'application et qui la déploie sur un serveur d'applications
- L'administrateur : l'acteur qui contrôle et supervise le fonctionnement du serveur d'application
- Le fournisseur de serveurs : l'éditeur qui commercialise un serveur d'application

Fournisseurs de serveurs d'applications J2EE

Offre commerciale

- IBM / WebSphere (n° 1)
- BEA / WebLogic
- Sun One
- Oracle 9i Application Server
- Et aussi Borland Entreprise Server, Macromedia / Jrun, SAP Web
- Application Server, Iona / Orbix E2A

Offre « open source »

- JBoss (n° 1)
- JOnAS
- EJB : OpenEJB, EJBean

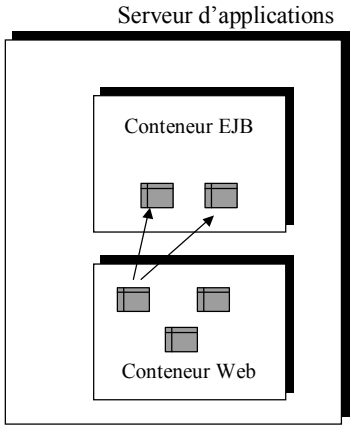
- Voir la liste des serveurs sur : http://java.sun.com/j2ee/licensees.html

Plus d'informations …

- http://java.sun.com/
- http://www.theserverside.com/
- http://developer.java.sun.com/developer/technicalArticles/J2EE/
- http://developer.java.sun.com/developer/onlineTraining/J2EE/
- http://www.triveratech.com/
- http://jonas.objectweb.org/

Serveur d'applications

Application J2EE =
- zéro, un ou plusieurs composants EJB
- zéro, un ou plusieurs composants Web
- reliés par un schéma d'assemblage



Composants Web/1

- Web Bean est un ensemble de :
  - JSP et/ou
  - Servlets et/ou
  - pages HTML

- Packagés dans un fichier archive .war

- Ce sont des composants qui :
  - implantent une logique de présentation simple pour des clients Web
  - servent de passerelle d'accès pour des composants EJB
  - peuvent implanter une logique de petits traitements

Composants Web/2

➢ Exemple :

Fichier.html

```
<html>
<head>
<title> Exemple </title>
</head>
<body>
<form method="POST"  action="/servlet/CalcServlet"  >
Addition de deux nombres : <br>
<input  type=text  name=Val1>
<br>
<input  type=text  name=Val2>
<br>
<input type=submit value="Additionner"  >
</body>
</html>
```


Fonctionnalités d'un container EJB/1

⏩ la connectivité entre les clients et les EJB : le connecteur gère les communications entre les clients et les EJB. Après le déploiement d'un EJB dans un serveur d'applications, le client peut invoquer les méthodes de cet EJB comme si elles étaient situées dans la même machine virtuelle, les communications sont gérées par le middleware sous-jacent.

⏩ la gestion de la persistance : les composants peuvent choisir de déléguer leur persistance au conteneur.

⏩ la gestion des transactions : les composants transactionnels peuvent déléguer la gestion de leurs transactions au conteneur.

Fonctionnalités d'un container EJB/2

⏩ la gestion de la sécurité : le conteneur assure l'application des politiques de sécurité déclarées mais non codées par le développeur.

⏩la gestion de la concurrence : les composants peuvent être invoqués par un seul client ou bien par plusieurs clients simultanément.

⏩la gestion du cycle de vie des composants : le conteneur assure la création et la destruction des instances des composants.

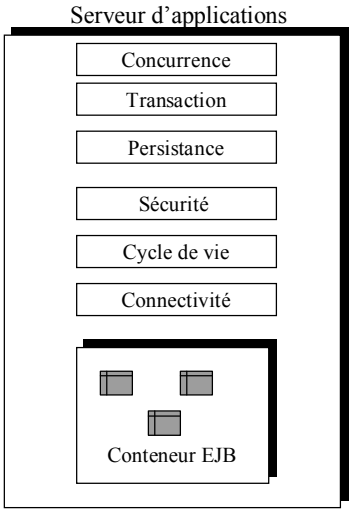
⏩ la création de réserves de connexions : l'obtention d'une connexion sur une base de données est coûteuse en termes de ressources, le nombre de connexions étant limité par le nombre de licences, le conteneur peut gérer une réserve de connexions.

Fonctionnalités d'un container EJB/3

6 services fournis par le serveur d'applications au conteneur EJB :

- transaction
- persistance
- sécurité
- cycle de vie
- concurrence
- connectivité

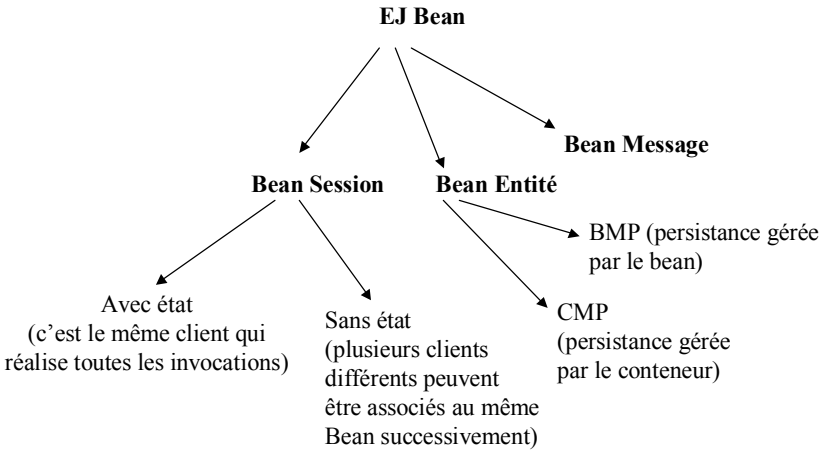
Ces services sont intégrés dés le départ à la plate-forme.



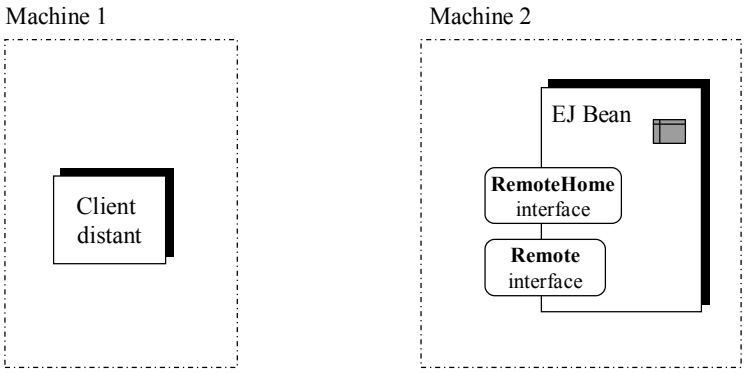
Composants EJB

- Composants applicatifs de J2EE (code métier)
- Potentiellement répartis et transactionnels
- se focalisent sur la logique applicative
- sont portables d'un serveur d'application à un autre
- Trois profils
  - **Session** : instances dédiées à un contexte d'interaction d'un client particulier
  - **Entité** : instances partagées représentant les données de l'entreprise
  - **Orienté message** : instances neutres réagissant à l'arrivée de messages asynchrones
- Gérés par le « container » EJB

Les EJ Beans/1

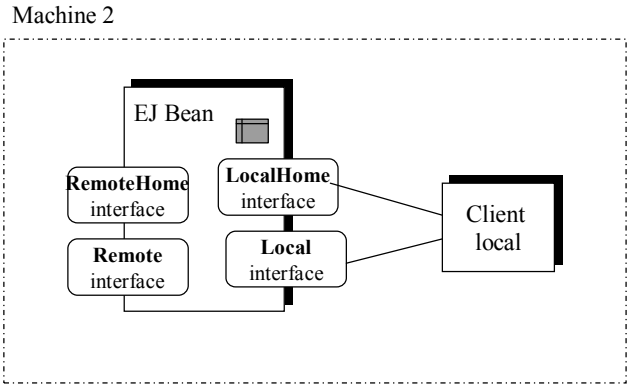


Les EJ Beans/2



Chaque EJ Bean fournit deux interfaces d'accès **distant**
- **`Remote`** : les services « métiers » (méthodes) fournis par le bean
- **`RemoteHome`** : interface de gestion du composant (création, recherche, destruction d'instances)

Les EJ Beans/3



+ éventuellement deux interfaces d'accès **local** (meilleure performance pour les clients hébergés dans le même conteneur).
- **`Local`** : les services « métiers » (méthodes) fournis par le bean
- **`LocalHome`** : interface de gestion du composant (création, recherche, destruction d'instances)

Les Beans Session/1

- Session dont la durée de vie est liée à celle de son client, c'est une prolongation du processus client dans un serveur d'application
- Meurt lorsque le client n'en a plus besoin (d'où l'idée de session)
- Bean à durée de vie plutôt courte
- Un Bean Session est créé par son client, utilisé et supprimé ensuite par son client

1. Bean Session sans état (Stateless session bean)
   - C'est un bean léger
   - Ne préserve pas d'état d'un appel à un autre
   - Deux instances quelconques d'un tel bean sont équivalentes
   - Bean sans variable d'instance
   - Exemple: demande de virement entre deux comptes, services de calcul, services de recherche d'informations dans une BDD
2. Bean Session avec état (stateful session bean)
   - C'est un bean lourd, effectue des opérations pour le compte du client
   - Gère un état en mémoire (objet avec état) pour maintenir l'état du client
   - Exemple: un panier sur un site de commerce électronique avec 2 attributs, nom du client et les articles sélectionnés

Les Beans Session/2

Quand utiliser un Bean Session ?

- Pas de besoin spécifique de partage de données entre les clients

1. Bean Session sans état (Stateless session bean)
   - Pour des tâches génériques
   - Pour consulter en **lecture seule** des données persistantes
   - Efficaces et faciles à implémenter
   - Les données sont passées comme paramètres de la méthode
2. Bean Session avec état (stateful session bean)
   - L'état du Bean représente l'état de l'interaction entre le client et le Bean
   - Le Bean doit conserver de l'information entre deux invocations du client
   - Dédié à un client pendant toute sa durée de vie
   - Le même Bean est utilisé pour servir tous les appels du même client

Les Interfaces métier Beans Session

Définissent les interfaces que le Bean peut rendre au client. Elles sont locales ou distantes.

1. Interface métier distante (**`Remote`**)
   - Accessible par des composants locaux ou distants au Bean
   - Hérite de l'interface `javax.ejb.EJBObject`
   - Les méthodes de l'interface lèvent l'exception `RemoteException`

2. Interface métier locale (**`Local`**)
   - Accessible uniquement par les composants situés sur la même machine que le Bean
   - Hérite de l'interface `javax.ejb.EJBLocalObject`
   - Pas d'exception `RemoteException`

Interface Remote d'un Bean Session

Exemple.

```
import javax.ejb.EJBObject;
import java.rmi.RemoteException;

public interface Calc extends javax.ejb.EJBObject {
    public double add (double val1, double val2) throws RemoteException;
}
```

Interface Local d'un Bean Session

Exemple.

```
import javax.ejb.EJBLocalObject;

public interface CalcLocal extends javax.ejb.EJBLocalObject {
    public double add(double val1, double val2) ;
}
```


Les Interfaces de gestion du cycle de vie

Locales ou distantes, gèrent la création, la recherche et la suppression des EJ Beans.

1. Interface Home distante (**`RemoteHome`**)
   - Gère le cycle de vie du Bean
   - Hérite de l'interface `javax.ejb.EJBHome`
   - Les méthodes de l'interface lèvent les exceptions `RemoteException` et `CreateException`

2. Interface Home locale (**`LocalHome`**)
   - Gère le cycle de vie du Bean
   - Hérite de l'interface `javax.ejb.EJBLocalHome`
   - Pas d'exception `RemoteException`

3. Méthodes possibles : `create` (création d'instances de bean, retourne l'interface `Remote` ou `Local` selon que l'interface est locale ou distante)

   Plusieurs méthodes `create` peuvent être définies avec plusieurs signatures

Interface Home d'un Bean Session

Exemple.

```
import javax.ejb.EJBHome;
import javax.ejb.CreateException;
import ja.rmi.RemoteException;


public interface CalcHome extends javax.ejb.EJBHome {
    public Calc create() throws CreateException, RemoteException;
}
```


Interface HomeLocal d'un Bean Session

Exemple.

```
import javax.ejb.EJBLocalHome;
import javax.ejb.CreateException;

public interface CalcLocalHome extends javax.ejb.EJBLocalHome {
    public CalcLocal create() throws CreateException;
}
```

Implémentation du Bean Session

Classe Java :

- qui définit les méthodes spécifiées dans les interfaces `Remote` et `RemoteHome`
- qui implémente l'interface `javax.ejb.SessionBean`
- la classe d'implémentation de `javax.ejb.SessionBean` ne déclare pas l'implémentation des interfaces

Method Summary (Méthodes de l'interface SessionBean)	
`void ejbActivate()`	The activate method is called when the instance is activated from its « passive » state.
`void ejPassivate()`	The passivate method is called before the instance enters the « passive » state.
`void ejbRemove()`	A container invokes this method before it ends the life of the session object.
`void setSessionContext()`	Sets the associated session context

Développement

Fournir des méthodes pour les interfaces `Remote` et `RemoteHome`

- même méthodes que dans l'interface `Remote`
- une méthode `ejbCreate` pour chaque `create` de l'interface `RemoteHome`
- même profil que `create`
- retourne `void`

Exemple de classe d'implémentation/1

```
import javax.ejb.SessionBean;
import javax.ejb.SessionContext;
import javax.ejb.CreateException;
public class CalcBean implements SessionBean {
       SessionContext sessionContext;

//constructeur
public CalcBean() {}

// méthodes de l'interface Home
public void ejbCreate() throws CreateException { };
```


Exemple de classe d'implémentation/2

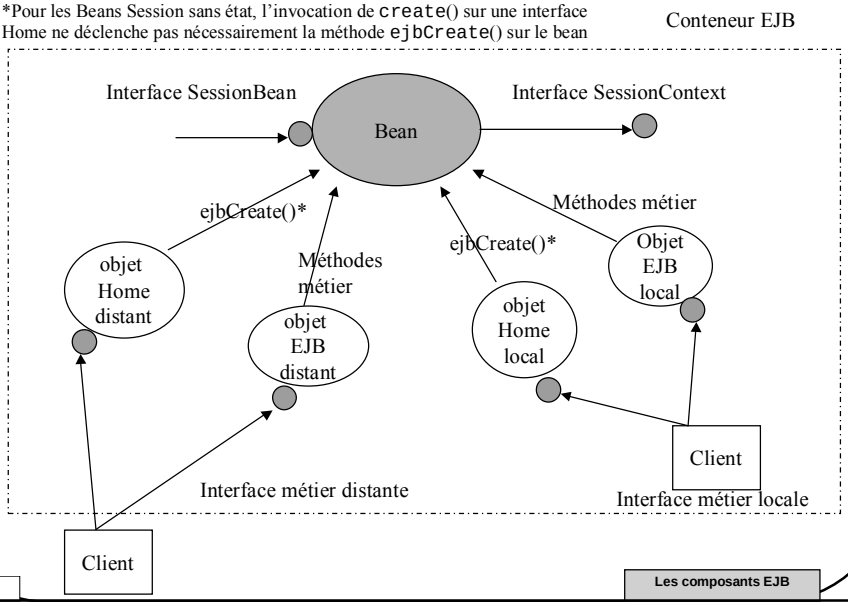
```
// méthodes de l'interface SessionBean
public void ejbRemove() { }
//public void ejbActivate() { } non utilisee par un Bean sans etat
//public void ejbPassivate() { } non utilisee par un Bean sans etat
public void setSessionContext (SessionContext sessionContext
sessionContext) {this.sessionContext  = sessionContext;}

// méthode de l'interface Remote
public double add(double val1, double val2)
       {      return val1+val2; }

} // fin de CalcBean
```

Structure d'un Bean Session

*Pour les Beans Session sans état, l'invocation de `create`() sur une interface Home ne déclenche pas nécessairement la méthode `ejbCreate`() sur le bean



Client EJB

Développement côté client :

1. Rechercher l'interface `Home` du Bean par son nom via JNDI
JNDI : API accès services nommage
(LDAP, CORBA COSNaming, DNS, RMI registry, etc.)

2. Accéder au Bean
L'interface `Home` permet d'accéder aux instances existantes du Bean ou d'en créer de nouvelles
=> on récupère une référence sur une interface `Remote`

3. Invocation du Bean

Classe Java du Client EJB

```
import javax.rmi.*;
import javax.naming.*; import javax.ejb.*;
public class ClientCalc {
   CalcHome calcHome; // reference sur l'objet Home distant
   Calc myCalc; //ref sur l'objet EJB distant
   try {// obtention du contexte initial : ref du service de nommage
     Context ctx = new InitialContext();
     //recherche de l'interface Home distante
     Object ref= ctx.lookup("Calc");
    calcHome = (CalcHome)PortableRemoteObject.narrow(ref,CalcHome.class);
    // ref: souche d'acces a l'interface Home de Calc
    // creation d'un EJB Calc
    myCalc = calcHome.create(); double somme = 0;
    //invocation de la methode metier add()
 somme = myCalc.add(Integer.parseInt(args[0]),Integer.parseInt(args[1]));
     System.out.println(somme);
   } catch(Exception e) {e.printStackTrace(); }   }
```


Couche Présentation/1

- Exemple :

Fichier.html

```
<html>
<head>
<title> Exemple </title>
</head>
<body>
<form method="POST"  action="/servlet/CalcServlet"  >
Addition de deux nombres : <br>
<input  type=text  name=Val1>
<br>
<input  type=text  name=Val2>
<br>
<input type=submit value="Additionner"  >
</body>
</html>
```

Couche Présentation/2

CalcServlet.java

```
public class CalcServlet extends HttpServlet {
  private static final String CONTENT_TYPE="text/html";
  private calcHome CalcHome;
```

// a l'initialisation de la servlet, on recupere la reference de l'interface Home de Calc

```
  public void init() throws ServletException {
       try {
        Context ctx=new InitialContext();
        Object ref=ctx.lookup("Calc");
        calcHome = (CalcHome)PortableRemoteObject.narrow(ref,
        CalcHome.class);
    } catch (Exception e) {e.printStackTrace();}
   } // fin de init()
```


Couche Présentation/3

CalcServlet.java (suite)

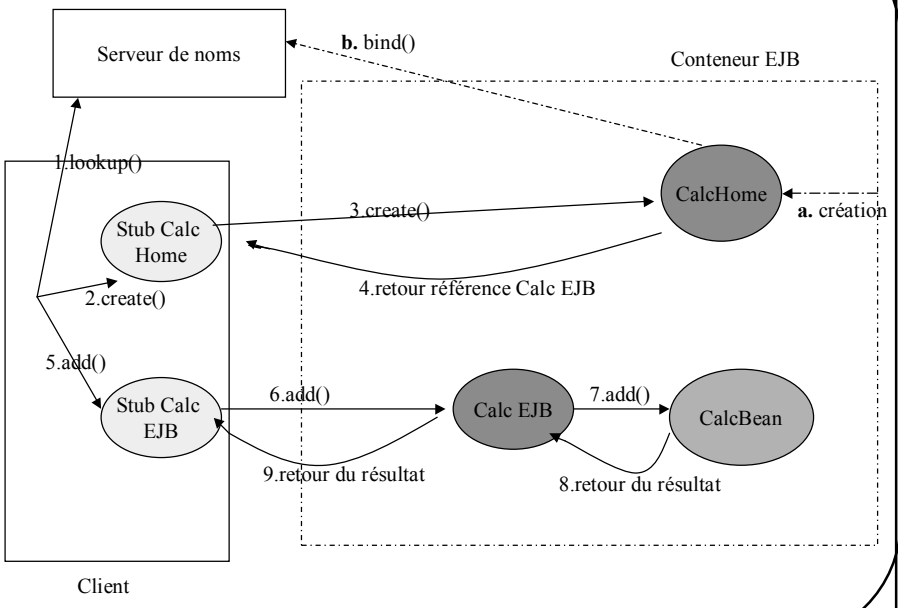
```
  public void doPost(HttpServletRequest requete, HttpServletResponse
reponse) throws ServletException, IOException {
       reponse.setContentType(CONTENT_TYPE);
       PrintWriter sortie=reponse.getWriter();
       Double d1=new Double(requete.getParameter("Val1"));
       Double d2=new Double(requete.getParameter("Val2"));
       double val1= d1.doubleValue();
       double val2=d2.doubleValue();
       // creation d'un EJB Calc
       myCalc = calcHome.create();
       //invocation des methodes metier add()
       double res=myCalc.add(val1, val2);
       sortie.println("<html><head> <title> Resultat </title> </head>
<body>");
```

Couche Présentation/4

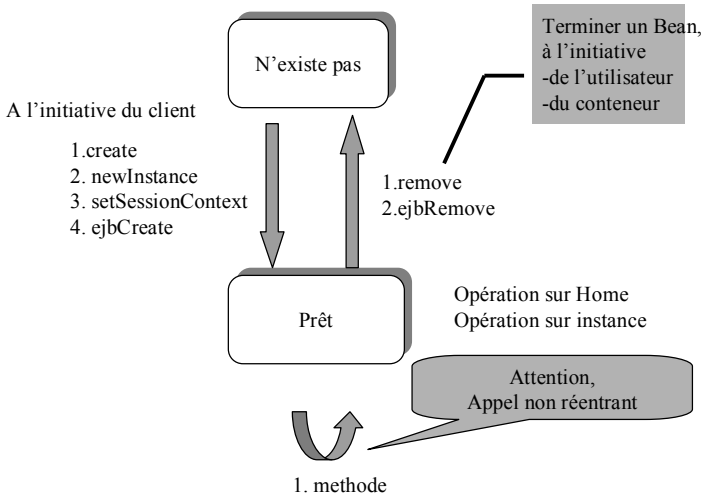
CalcServlet.java (suite)

```
 sortie.println("<p> Résultat de l'Opération " + val1 + "+" + val2 +"="+
"<b>"+res +"</b>"+ "<p>");
 sortie.println("</body></html>");
} // fin de doPost
} // fin de la servlet
```

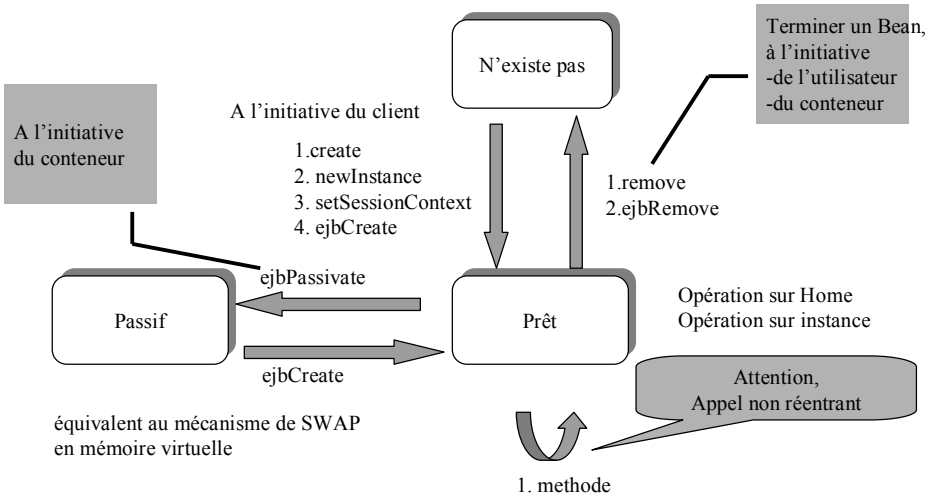

Intéraction entre le client et le Bean



Cycle de vie d'un Bean Session sans état



Cycle de vie d'un Bean Session avec état



Spécifications des propriétés du Bean

Un Bean est défini grâce à ses propriétés :
-Pour un Bean Session, il faut indiquer s'il est avec ou sans état.
-Pour un Bean Entité, il faut indiquer si la persistance est gérée par le conteneur ou par le programmeur.
-Pour les deux types de Beans, il faut indiquer si la gestion des transactions sera gérée par le conteneur ou par le programmeur.

Les propriétés du Bean sont fournies dans un fichier XML appelé descripteur de déploiement

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD Enterprise
JavaBeans 2.0//EN" "<http://java.sun.com/dtd/ejb-jar_2_0.dtd>">
<ejb-jar>
   <enterprise-beans>
     <session>
      <display-name>Calc</display-name>
      <ejb-name>Calc</ejb-name>
      <home>rep.CalcHome</home>
      <remote>rep.Calc</remote>
      <ejb-class>rep.CalcBean </ejb-class>
        <session-type>Stateless </session-type>
        <transaction-type>Container</transaction-type>
      </session>
```


Spécifications des propriétés du Bean

```
</enterprise-beans>
    <assembly-descriptor>
       <container-transaction>
              <method>
                     <ejb-name>Calc</ejb-name>
                     <method-name>*</method-name>
              </method>
              <trans-attribute>Required</trans-attribute>
       </container-transaction>
     </assembly-descriptor>
</ejb-jar>
```

Déploiement du Bean/1

Une fois que le Bean est défini, il faut le déployer dans un serveur d'application. Il faut alors assembler tous les éléments du Bean dans un fichier JAR qui va contenir :

- Les interfaces du Bean
- La classe du Bean et les classes auxiliaires
- Le descripteur de déploiement.

Les serveurs d'application offrent des interfaces graphiques, appelées consoles d'administration qui permettent de rééditer le descripteur de déploiement. Une fois l'objet déployé, ce dernier enregistre la référence à l'objet EJBHome dans un serveur de noms accessibles via JNDI.

Calc.jar = { Calc.class, CalcLocal.class, CalcHome.class,CalcHomeLocal.class, CalcBean.class, ejb-jar.xml }

A partir du fichier JAR, le serveur est capable d'extraire son contenu et de rendre le Bean utilisable par les clients, càd, générer des composants nécessaires à la communication (Stub et Skeleton) entre le Bean et ses clients (on parle de code déployé), on peut trouver :

-l'objet EJB qui implémente l'interface métier
-L'objet EJBHome qui implémente l'interface Home.

Déploiement du Bean/2

Packaging des applications

Une application EJB = 1 archive **.ear** = { 1 descripteur XML de l'application,
1 archive .war par composant Web,
1 archive .jar par composant EJB }

Une archive **.war** = { 1 descripteur XML du composant Web,
JSP ou servlet.class }

Une archive **.jar** = { 1 descripteur XML du Bean,
les fichiers .class du Bean }

Restrictions des EJBs et Gestion des ressources

Restrictions des EJBs
-les EJBs ne doivent pas manipuler des threads
-les EJBs ne doivent pas effectuer des opérations d'entrée/sortie
-les EJBs ne doivent pas manipuler les sockets serveur
-les EJBs ne doivent pas charger des librairies écrites en code natif (C, C++)

Gestion des ressources
Les serveurs EJB gèrent une charge importante tout en gardant un bon niveau de performance. Les EJBs mettent en œuvre deux techniques pour gérer un grand nombre de Beans (donc de clients):les pools d'instance et le partage de ressources entre les Beans
Un pool d'instances des EJBs est une réserve d'instances créées à l'avance: technique utilisée par les Beans Session sans état, les Beans Message et Entité.