Conservatoire national des arts et métiers

le cnam

Liban - Beyrouth

EJB Tutorial

Par: HARB Tony

Sommaire

Les EJB

Les Entreprise Java Bean ou EJB sont des composants serveurs donc non visuels.

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

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

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, ...

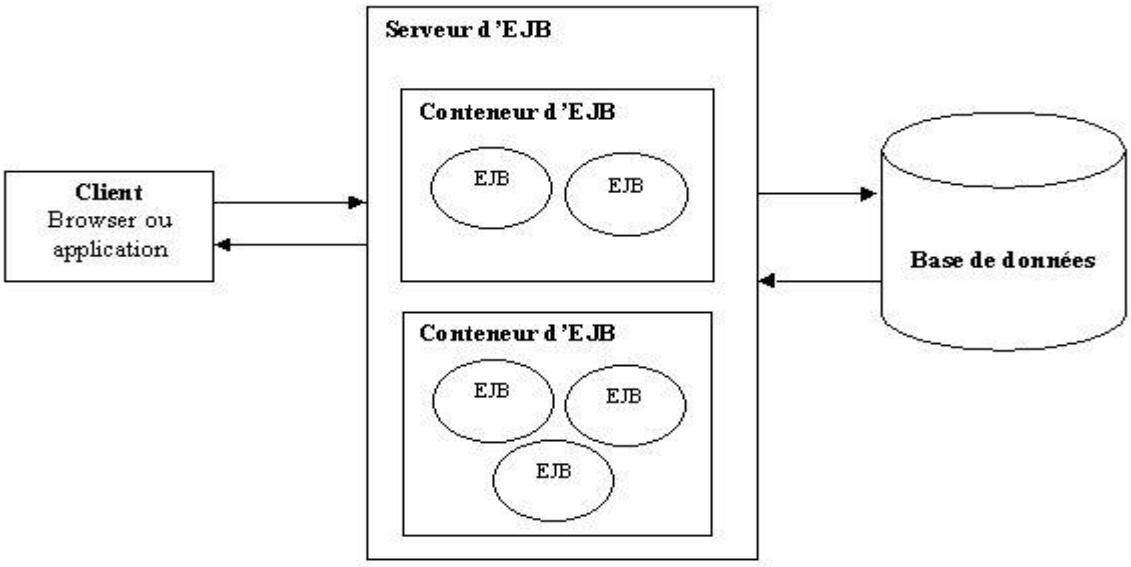
La présentation des EJB

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.

L'avantage c'est que l'EJB peut utiliser les services proposés par le conteneur 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.

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.

Les beans entités 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).

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 : 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.

✓ L'interface home permet de définir l'ensemble des services qui vont assurer 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

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.

```java
package com.jmdoudoux.ejb;

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

public interface MonPremierEJB extends EJBObject {
  public String message() throws 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 les clés primaires des deux beans sont identiques
- 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.

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.

```
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()

EJB sessions

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 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().

EJB sessions sans état

Il est inutile au serveur de sérialiser un EJB session sans état. Il suffit simplement de déclarer les méthodes ejbActivate() et ejbPassivate() sans traitements.

```
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. }
```

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.

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.

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.

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.

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.

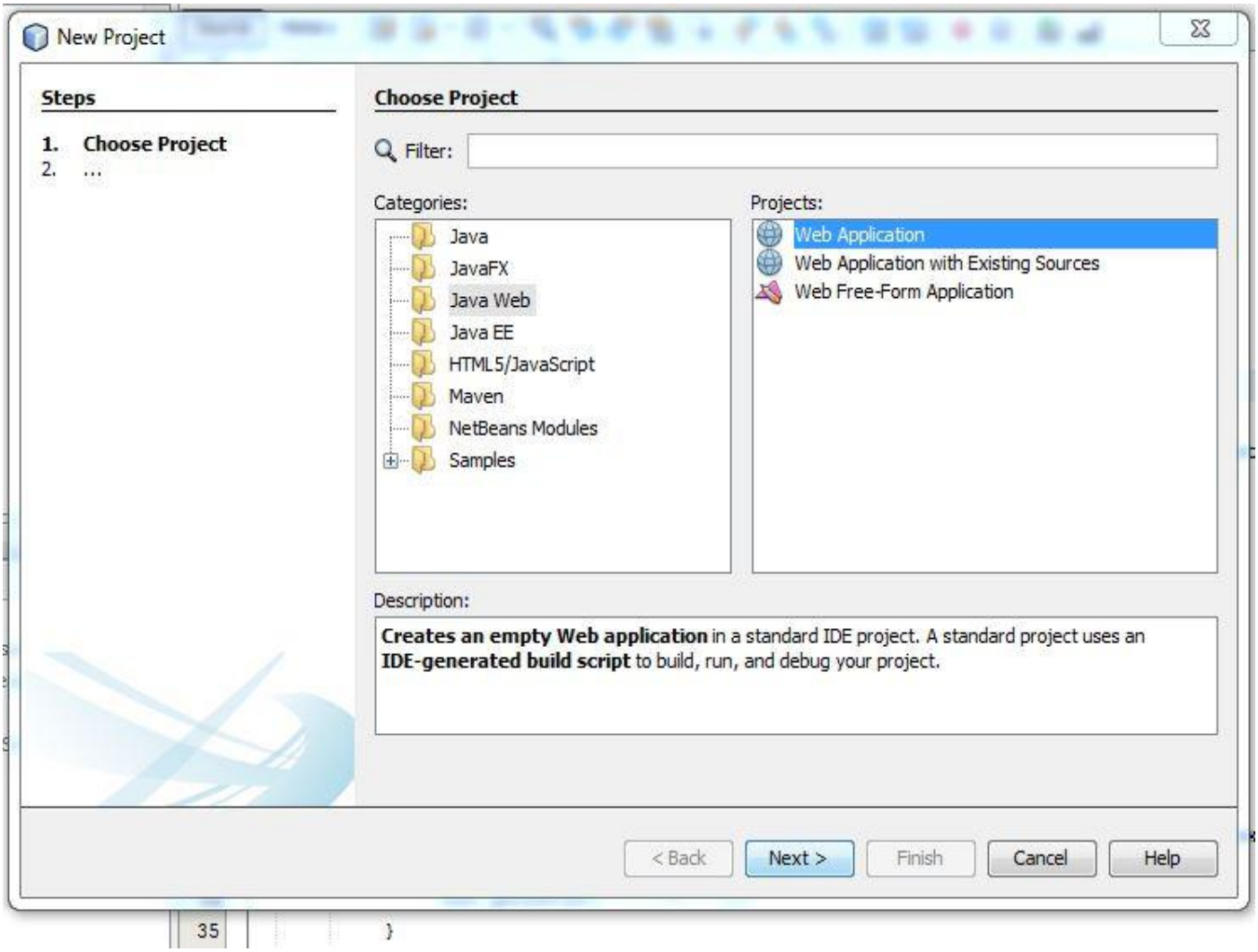
Exemple sur Netbeans

Pour poursuivre cet exemple, il faut installer quelques softwares :

Software or Resource	Version Required
NetBeans IDE	7.2, 7.3, 7.4, 8.0, Java EE version
Java Development Kit (JDK)	version 7 or 8
GlassFish Server Open Source Edition	3.x, 4.x

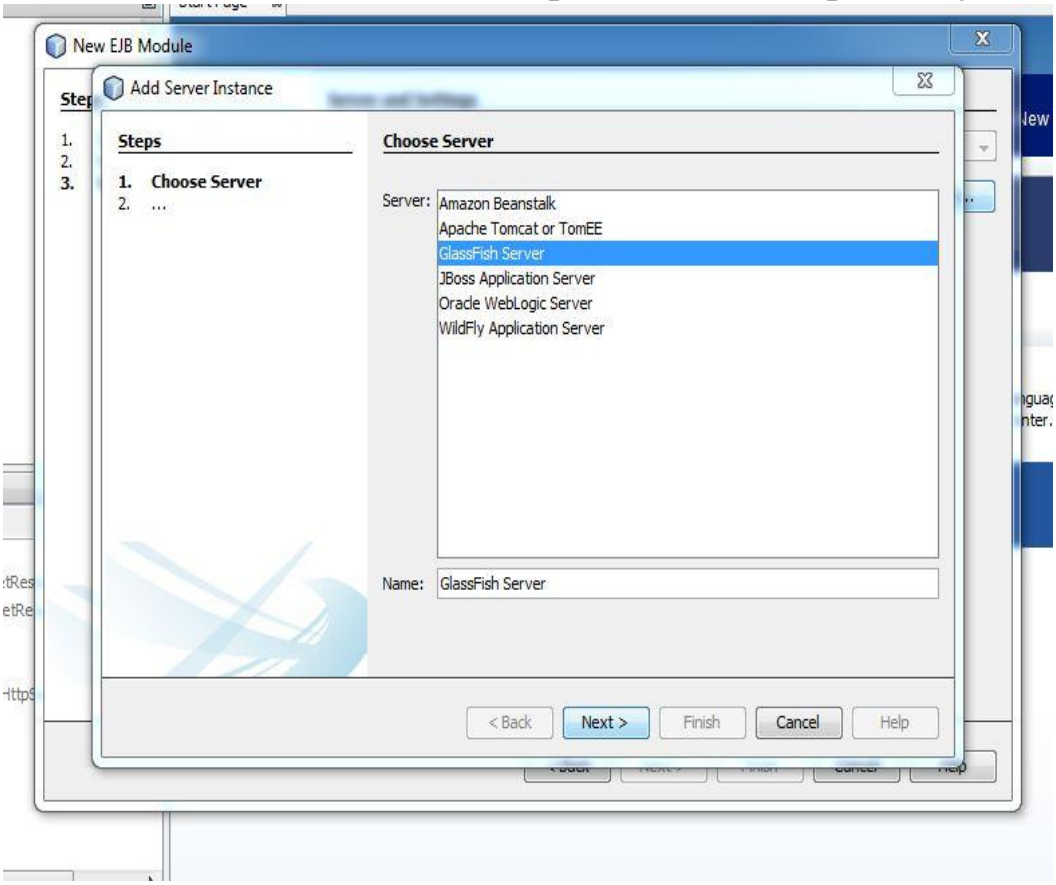
Cet exemple permet de calculer la somme des deux entiers.

1- Créer une web application (new Project)



2- Cliquer sur Next et renommer le projet

3- Choisir le serveur GlassFish ou presser sur Add pour l'ajouter.



4- Modifier le body de la page index.jsp :

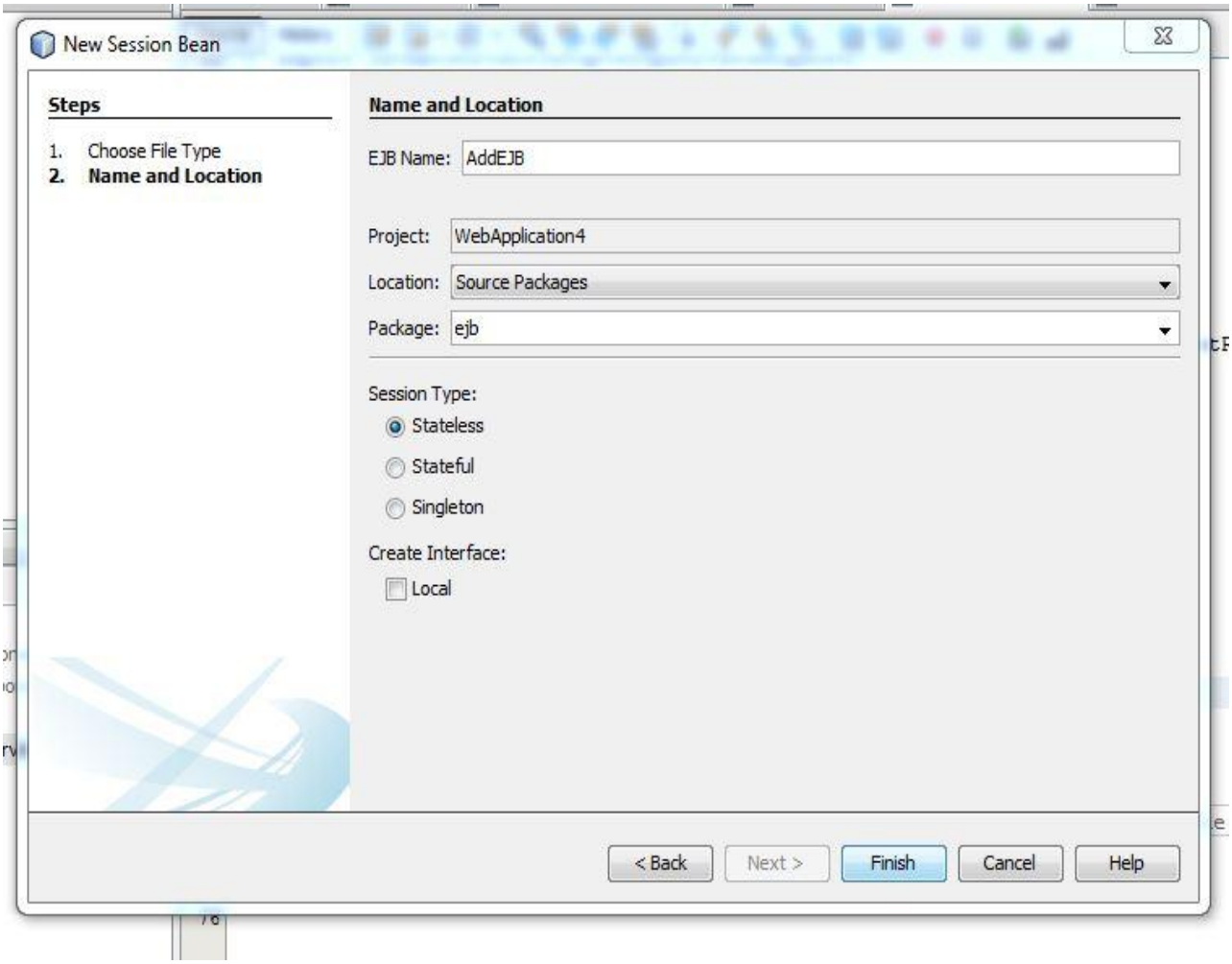
```
<form action="MyServlet" method="post">
        <h1>Hello World!</h1> <br>
        Username: <input type="text" name="username"> <br>
        Password <input type="password" name="pass"> <br>

    <input type="submit" method="method1" value="Login">
```

5- Right click sur le projet → New →Servlet et renommer la « MyServlet »

6- Ajouter ce code au Servlet :

```
@EJB
    AddEJB obj;
    protected void processRequest(HttpServletRequest request,
HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        try (PrintWriter out = response.getWriter()) {
            int i=Integer.parseInt(request.getParameter("n1"));
            int j=Integer.parseInt(request.getParameter("n2"));
            obj.setI(i);
            obj.setJ(j);
            obj.add();
            int k = obj.getK();
            out.println("addition using EJB is: "+k);
        }
```

7- L'instance AddEJB est une SessionBean stateless qui doit être créée. Right click → New → SessionBean

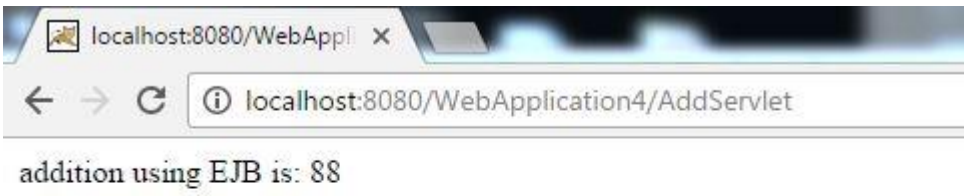


8- Ajouter le code :

```
private int i,j,k;

   public int getI() {
       return i;
   }

  public void setI(int i) {

       this.i = i;
   }
   public int getJ() {
       return j;
   }
   public void setJ(int j) {
       this.j = j;
   }
   public int getK() {
       return k;
   }
   public void setK(int k) {
       this.k = k;
   }
   public void add(){
   k = i + j ;
   }
```

9- Right-click sur la webapplication → build→ Run→ browser ouvre et la page jsp s'affiche.

10- Presser sur Add :



Bibliographie :

Developpons en Java : www.jmdouxdoux.fr

www.youtube.com