CONCEPTION ET DEVELOPPEMENT D'APPLICATIONS INTERNET

CDAI 4 - Entreprise Java Beans (EJB)

Université Paris Dauphine

Master M2 MIAGE

Année 2014-2015

Bekhouche Abdesslem

Sobral Diogo

DAUPHINE UNIVERSITÉ PARIS

PLAN

1. INTRODUCTION
2. LES SESSIONS BEANS
3. LES ENTITYS BEANS
4. DES ENTITYS BEANS VERS JPA
5. JPA - UTILISATION DANS UNE SESSION EJB
6. DÉPLOIEMENT
7. LES MESSAGES DRIVEN BEANS
8. HISTORIQUE
9. CONCLUSION
10. WEBOGRAPHIE - BIBLIOGRAPHIE ET RÉFÉRENCE

1. INTRODUCTION

Limite de la programmation usuelle :

- tout est la la charge du programmeur
  - construction des différents modules
  - définition des instances
  - interconnexions des modules
- structure de l'application peu visible
  - ensemble des fichiers de codes nécessaire
- évolution / modification difficile
  - changement du mode de communication
  - évolution, ajout, suppression de fonctionnalités
  - modification du placement
- développement, génération des exécutables, déploiement
  - pas ou peu d'outils pour les applications réparties

1. INTRODUCTION

Programmation constructive ou par composition

- Motivation : réutilisation de logiciel
  - intégration de modules logiciels existants
  - construction d'applications réparties par assemblage de modules logiciels existants
  - programmation à gros grain ("programming in the large")

- Approche : description de l'architecture de l'application à l'aide d'un langage déclaratif
  - modèle de construction des composants
  composants : interfaces, attributs, implémentation
  - description des interactions entre composants (connecteurs)
  - description de variables d'environnement
  (placement, regroupement, sécurité, etc.)

1. INTRODUCTION

Qu’est-ce que qu'un EJB ?

- Définition usuelle des composants
  - module logiciel autonome pouvant être installé sur différentes plates-formes
  - qui exporte différents attributs, propriétés ou méthodes
  - qui peut être configuré
  - capable de s’auto-décrire

- Intérêt : être des briques de base configurables pour permettre la construction d’une application par composition

Quelques composants célèbres

- COM / DCOM, Java Beans,
  Enterprise Java Beans, Composants CORBA

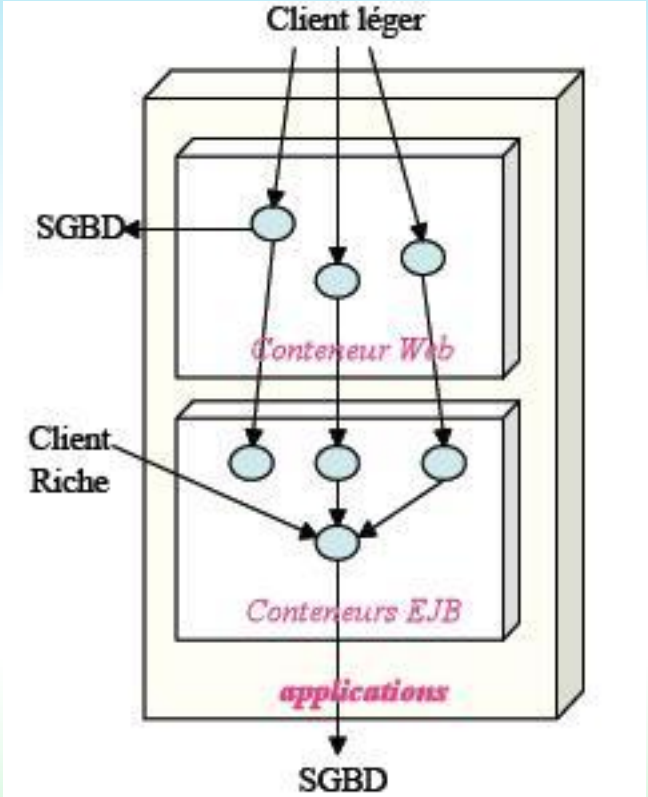
1. INTRODUCTION

- Modèle de composants pour le développement d'applications d'entreprises:

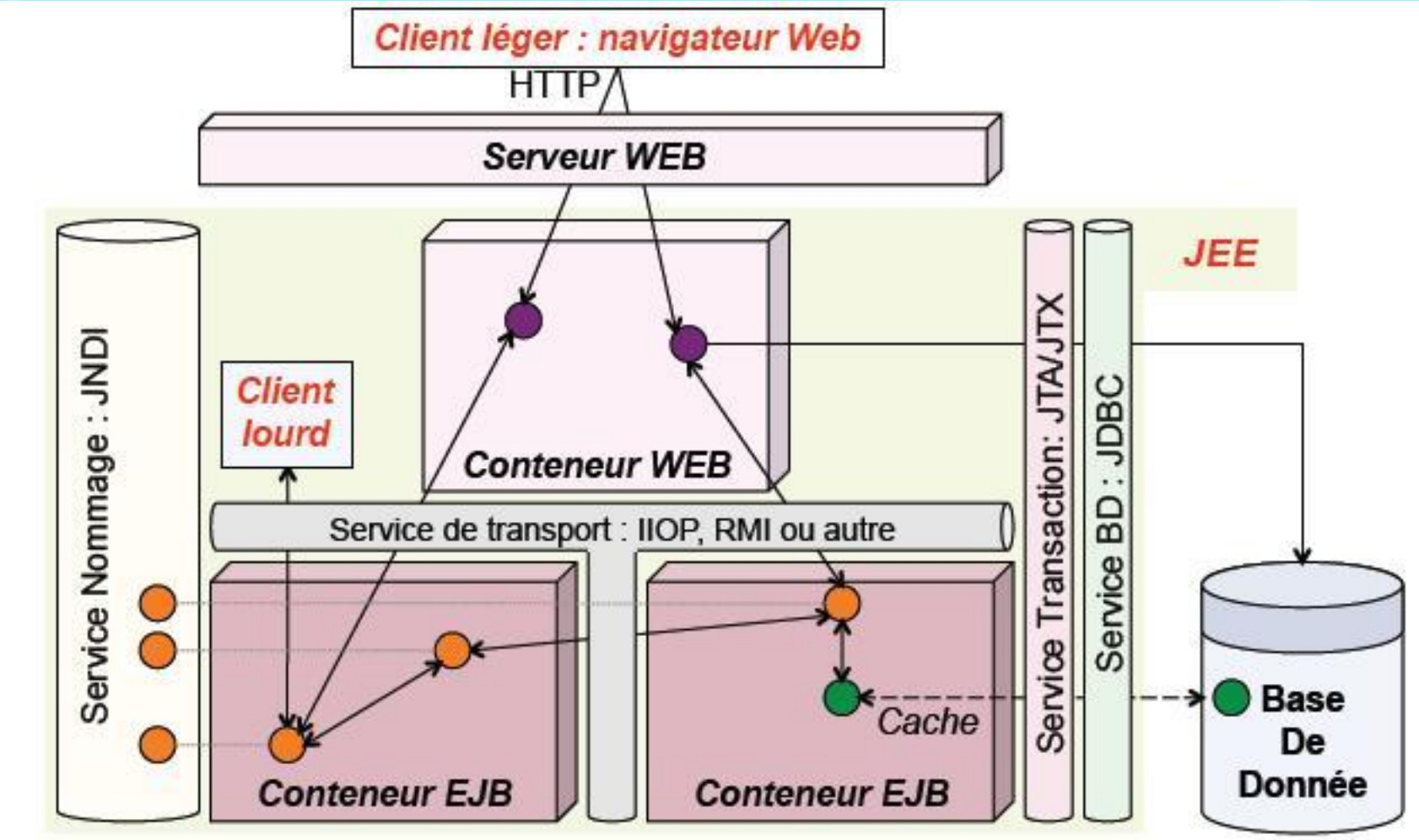
- Modèle de programmation:
  - par objet réparti (largement influencé par RMI);
  - par envoie de message

- La plateforme a composant est basée sur du Java

- Ne gère pas l'hétérogénéité



1. INTRODUCTION



1. INTRODUCTION

- Un conteneur EJB est un composant coté serveur qui encapsule la logique métier d'une application

- On se focalise sur la logique applicative
  - Les services systèmes sont fournis par le conteneur
  - La logique de présentation est du ressort du client
  - Les enterprise beans sont portables d'un serveur d'application à un autre

But : simplifier le développement modulaire d’applications réparties

1. INTRODUCTION

- EJB : un composant
  - Uniquement Java
  - Décrit en xml ou par annotation Java

- Conteneur EJB : la plateforme à composant définie par SUN (aujourd'hui Oracle)
  - Uniquement Java
  - Programmation répartie
  - Orienté vers les serveurs d'entreprise
  - Dynamiquement adaptable
  - Liaison par appel de méthode et par envoie de message
  - Outils de génération dynamique de souches et squelettes
  - Nombreux services systèmes
  - Transaction, nommage, persistance, cycle de vie, transport

1. INTRODUCTION

Il existe trois types d'EJB :

- Session : effectue une tâche pour un client
  - Avec état : un état spécifique par client
  - Sans état : pas d'état spécifique

  (partagé entre clients ou non, dépend de l'implémentation)

- Entity : représente une entité métier persistante
  - Associé à un tuple d'une base de donnée
  - Gestion transparente de la persistance

- Message-Driven : un échangeur de messages asynchrones
  - Session bean sans état
  - Activé lors de la réception de messages

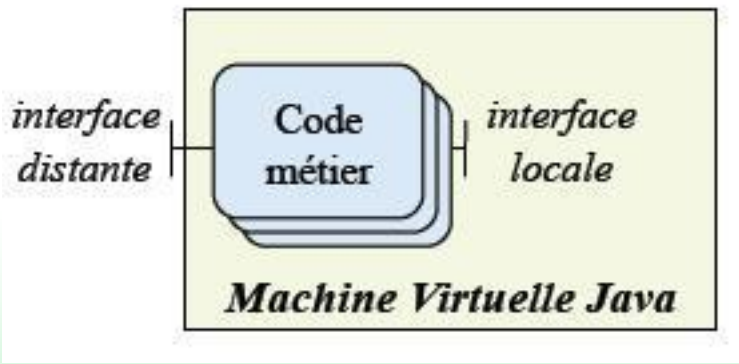
2. Les Sessions Beans

- Deux interfaces d'accès :
  - Interface locale : invocation uniquement dans la même JVM
  - Interface distante : invocation distante

```
import javax.ejb.Local;
@Local
public interface HelloLocal {
public String getHelloMessage();
}
```

```
import javax.ejb.Remote;
@Remote
public interface HelloRemote {
public String getHelloMessage();
}
```

- Un code métier :
  - Implantation des méthodes de l' interface



- Différence par rapport à RMI?
  - Gestion automatique du cycle de vie
  - Facilité d'accès aux autres services systèmes
  - **Un bean par client**

2. Les Sessions Beans

L'interface : rien de particulier

```
package test;
public interface HelloWorld {
public void sayHello();
}
```

Le code métier : **des annotations**

```
@Stateless(name="HelloWorld")
@Remote(HelloWorld.class)
public class HelloWorldBean implements HelloWorld {
public void sayHello() {
System.out.println("Hello, World!!!");
} }
```

2. Les Sessions Beans

Quelques remarques :

- Nommage : chaque recherche renvoie une nouvelle instance
  - name : nom EJB du bean – par défaut : le nom de la classe
  - mappedName : nom dans le service de nommage global (JNDI)

  Utile pour les clients lourds

- Interfaces d'accès :
  - @Remote : indique l'interface externe, plusieurs interfaces possibles
  - @Local : indique l'interface interne, plusieurs interfaces possibles
  - Par défaut : @Remote des interfaces directes

- Avec ou sans état :
  - Bean sans état (@Stateless) : état indépendant du client (attention, partage possible). Performance optimisé et réutilisable dans d'autre contexte.
  - Bean avec état (@Statefull) : état propre à chaque Bean. Spécifique à

2. Les Sessions Beans

Utilisation d'un Session Bean à partir d'un autre Session Bean

```
public interface Test {
public void test();
}

@Statefull(mappedName = "Test@Remote »)
@Remote(Test.class)
public class TestBean implements Test {
@EJB(name="HelloWorld")
HelloWorld hw;
public void test() { hw.sayHello(); }
}
```

2. Les Sessions Beans

Utilisation d'un Session Bean à partir d'un client Lourd

```
public static void main(final String[] args)
throws Exception {
Context ic = new InitialContext();
Test test = (Test)ic.lookup("Test@Remote");
test.test();
}
```

InitialContext : service de nommage (voir cours JNDI)
Le client ne peut utiliser que le nom JNDI, pas le nom du Bean.
Lancer le client : dans un conteneur client (avec Jonas : jclient au lieu de java)

2. Les Sessions Beans

Injection de ressources

- Accès à d'autres objets enregistrés dans le service de nommage

```
@Statefull(mappedName = "Test@Remote")
@Remote(Test.class)
public class TestBean implements Test {
@Resource(mappedName= "connexion_jdbc")
Object o;
}
```

2. Les Sessions Beans

Suppression de Session Bean :
Explicite après l'invocation d'une méthode marquée @Remove

@Remove
void del() { System.out.println("Va être supprimé"); }

Fonction destroy() : appelée juste avant toute suppression

- Après un appel explicite à une méthode marquée @Remove
- Lors du déchargement du Bean

Attention : *@Remove void destroy() { … }*
=>fonction appelée deux fois! ! Bug

2. Les Sessions Beans

Interception des changements d'état du cycle de vie

@PostConstruct
public void initialise() { ... initialise le Bean ... }

@PreDestroy
public void detruit() { ... destruction du Bean ... }

@PrePassivate
public void avantSwap() { ... Bean va être mis en swap ... }

@PostActivate
public void apresSwap() { ... Bean vient d'être rechargé ... }

2. Les Sessions Beans

Interception de méthodes métier

Utile pour debuggage, monitoring, changement de paramètres, évolution d'API

Méthode @AroundInvoke :

- Invoquée à la place de toute méthode métier
- Sauf si elle est annotée @ExcludeClassInterceptors

```
@AroundInvoke
public Object intercept(InvocationContext ic)
throws Exception {
System.out.println("*** intercept la méthode '" +
ic.getMethod().getName()
try { return invocationContext.proceed(); }
} finally { System.out.println("*** fin");
} }
```

2. Les Sessions Beans

Compilation : javac classique

Ajouter le .jar javaee dans le classpath

Packaging : dans un fichier .jar classique

/test/HelloWorld.class
/test/HelloWorldBean.class
/test/Test.class
/test/TestBean.class

Déploiement sous NetBeans : deployer le projet (clic droit -> "deploy")

3. Les Entity Beans

Entity Bean = tuple d'une base de donnée

Entity Bean : POJO

(Plain Old Java Object)

- Champs du POJO = colonne d'une table

En réalité, les EJB3 Entity a proprement parler n'existent plus au sein des EJB3.

La spécification EJB3 a aussi donné naissance à JPA (Java Persistence API) pour la gestion des Entity Beans.

API des Entity Bean : JPA (Java Persistance API)

- Unification des APIs Hibernate, TopLink…

Gestion (quasi-)transparente de la persistance

4. Des Entity Beans vers JPA

Les Bean Entity utilisés dans les version précédentes des EJB sont donc remplacé par des entités JPA.

- A titre de comparaison, ces entités subissent de grandes modifications :
  - Très forte influence d'Hibernate
  - Annotation de type @Entity
  - Nécessite un constructeur par défaut (ou pas de constructeur)
  - Doit implémenter Serializable pour une utilisation distante

- Instanciation par l'opérateur **new**
- Primary Key : @Id permet de déclarer une clé primaire
- Mapping des champs par défaut :
- Annotation @Basic

4. Des Entity Beans vers JPA

Exemple de base

```
@Entity
@Table(name = "FILMS")
public class Film implements java.io.Serializable {
@Id @GeneratedValue(strategy = GenerationType.AUTO)
private int id;
private String name;

public int getId() { return id; }
public void setId(int id) { this.id = id; }

public String getName() { return name; }
public void setName(final String name) { this.name = name }
}
```

4. Des Entity Beans vers JPA

- @Basic ou rien : indique qu'un champs est persistant
- Tous les champs sont persistants
- *Changer le nom de la colonne de la BD :*
  @Column(name="nomBD") int nomJava;
- Table associée au tuple : @Table(name="FILMS")
  Pas nécessaire de de mettre tous les champs de la table
- Clé primaire obligatoire : type primitif ou composé
  @GeneratedValue(strategy=?) : indique comment sont générés les clés
  Auto, Identity, Sequence, Table

4. Des Entity Beans vers JPA

Les jointures entre EntityBean

Cardinalité des relations :

- 1-1
- 1-n (ex. : une commande contient n lignes)
- n-1 (ex. : plusieurs ligne de commandes peuvent concerner le même produit)
- n-n (ex. : un cours comporte +sieurs étudiants qui suivent +sieurs cours)

Dénomination : OneToOne, OntToMany, ManyToOne, ManyToMany

Principe : un unique propriétaire de la relation
Propriétaire : tuple qui possède la clé externe

4. Des Entity Beans vers JPA

Exemple de jointure : *« un film est joué dans des salles »*

```
create table FILMS (
id integer primary key auto_increment,
name VarChar(256)
);

create table SALLES_PROG (
id integer primary key auto_increment,
salle_id integer,
film_id integer,
foreign key (salle_id) references SALLES(id),
foreign key (film_id) references FILMS(id)
);
```

film_id : clé étrangère ! propriétaire : SALLES_PROG

4. Des Entity Beans vers JPA

Exemple de jointure

Dans le Bean Film :

```
@OneToMany(mappedBy = "film", fetch=FetchType.EAGER,
cascade=CascadeType.ALL)
private List<SalleProg> salles;
```

Dans le bean SalleProg (propriétaire) :

```
@ManyToOne
@JoinColumn(name = "film_id")
private Film film;
```

"film" est le champ de jointure chez le propriétaire

4. Des Entity Beans vers JPA

Complément sur les jointures

Fetch : façon de gérer la jointure (aussi paramètre des @Basic)

- FetchType.EAGER : au plus tôt (nécessaire si Serializable)
- FetchType.LAZY : au plus tard (inutilisable avec Serializable)

Cascade : cascade des opérations entre les beans

- CascadeType.ALL : toute opération propagée
- CascadeType.MERGE : merge entre deux beans (voir + loin dans le cours)
- CascadeType.PERSIST : Film devient persistant ! List<SalleProg> aussi
- CascadeType.REFRESH : rechargement à partir de la base
- CascadeType.REMOVE : suppression en cascade

4. Des Entity Beans vers JPA

Utilisation des EntityBeans : dans des SessionBeans

L'EntityManager : gestionnaire de persistance

```
public interface Cinema{ Film findFilm(int id); }


@Stateless
public class CinemaBean implements Cinema {
@PersistenceContext
private EntityManager em;
...
}
```

4. Des Entity Beans vers JPA

Etat d'un Entity Bean

- Attaché : géré par l'entity manager
  Modification répercutée dans la BD
  - Tout Entity Bean renvoyé par l'entity manager

- Détaché : non géré par l'entity manager
  Modification non répercutée dans la BD
  - Nouveau Bean
  - Copie de Bean (après sérialisation)

4. Des Entity Beans vers JPA

- Une instance d'EntityManager est associée avec un contexte de persistance. Ce contexte est un ensemble d'instances d'entités pour lequel pour n'importe quelle entité, il existe une unique instance de l'entité.

- Le cycle de vie des instances est lié a ce contexte :
  - pas d'identité persistante (pas associée au contexte de persistance).
  - "instance gérée" est une instance avec une identité persistante associée avec un contexte de persistance.
  - "instance détachée" est une instance ayant une identité persistante mais qui n'est plus liée au contexte de persistance.
  - "instance supprimée" est une instance ayant une identité persistante, associée avec le contexte de persistance mais qui est programmée pour être supprimée de la base de données.

4. Des Entity Beans vers JPA

L'EntityManager : attachement et détachement

- void merge(Object entity) : synchronise l'état persistant du bean
  Utilisable sur un bean détaché. Ne rattache pas le bean.
  Utile lorsque le Bean est modifié chez le client et renvoyé au serveur.

- void persist(Object entity) : rend l'entité persistante et attachée
  Utilisable sur un nouveau Bean (après un new)

```
Film createFilm(String name) {
Film film= new Film();
res.setName(name);
em.persist(film); // attache le bean + rend persistant
return film; // la copie est bien sûr détachée
}
```

4. Des Entity Beans vers JPA

Principales méthode d'un EntityManager

- Object find(Class cl, Object key) : Trouve un EntityBean à partir de sa clé rimaire
- boolean contains(Object entity) : Vrai si entity est attaché à l' EntityManager
- Query createQuery(String qlString) : Création d'une requête dans le langage « query language »
- Query createNamedQuery(String name) : Création d'une requête nommée
- Query createNativeQuery(String sqlQuery) : Création d'une requête dans le langage SQL
- void remove(Object entity) : Supprime l'entity de la base
- void refresh(Object entity) : Recharge le bean à partir de la base

4. Des Entity Beans vers JPA

Exemple d'utilisation

Trouver un film à partir de sa clé primaire :

```
public Film findFilm(int id) {
return em.find(Film.class, Integer.valueOf(id));
}
```

Supprimer un film :

```
public void removeFilm(int id) {
em.remove(findFilm(id));
}
```

4. Des Entity Beans vers JPA

Le langage « Query Language » : langage de requête proche de SQL

- Sélection à partir du nom du Bean
- Paramètres indiqués par :nom-du-parm
- Positionnement des requêtes dans l'objet Query
- Demande d'une liste de résultats à partir de l'objet Query

Exemple :

```
public Film findFilmByName(String name) {
Query q = em.createQuery(
"select f from Film where f.name = :fname");
q.setParameter("fname", name);
List<Film> res = q.getResultList();
return res.size() == 0 ? null : res.get(0);
}
```

4. Des Entity Beans vers JPA

Exemple - Recuperer tous les pays dont le nom est 'France' :

Sans parametre :

```
String queryText = "from Country where name='France'";
Query query = em.createQuery(queryText);
Country country = query.getSingleResult();
```

Avec parametre :

```
String queryText = "from Country where name=:countryName";
Query query = em.createQuery(queryText);
query.setParameter("countryName", "France");
Country country = query.getSingleResult();
```

4. Des Entity Beans vers JPA

Requête nommée : définie avec le Bean

```
@Entity
@Table(name = "FILMS")
@NamedQueries({
    @NamedQuery(name = "findAllFilms",
                query = "select f from Film f"),
    @NamedQuery(name = "findFilmByName",
                query = "select f from Film f WHERE f.name = :fname")
})
public class Film implements java.io.Serializable { ...
```

Utilisation dans le SessionBean :

```
void findFilmByName() {
Query q = em.createNamedQuery("findFilmByName"); ...
```

4. Des Entity Beans vers JPA

Interception des changements d'état du cycle de vie

Autour de la création (em.persist) :

- @PrePersist
- @PostPersist

Lors du chargement à partir de la base (em.find, Query.getResultList)

- @PostLoad

Autour des mises à jour (modification de champs, em.merge)

- @PreUpdate
- @PostUpdate

Autour de la suppression (em.remove)

- @PreRemove
- @PostRemove

5. JPA - Utilisation dans un EJB session

- L'entity manager est injecté par IoC dans les EJB.
- Rien d'autre a faire que de déclarer qu'on en a besoin :

```
import javax.ejb.Stateless;
@Stateless
public class CountryBean implements CountryRemote {

@PersistenceContext
private EntityManager em;

public List<City> listCities(String countryName) {
Query query = em.createQuery("from City where
country.name = :contryName");
query.setParameter("countryName", countryName);
List<City> cities = query.getResultList();
return cities;
}
}
```

6. Déploiement

Le descripteur de déploiement

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.0" xmlns="http://java.sun.com/xml/ns/persistence" xmlns:
xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http:
//java.sun.com/xml/ns/persistence http://java.sun.
com/xml/ns/persistence/persistence_2_0.xsd">

<persistence-unit name="ZZ_TEST-ejbPU" transaction-type="JTA">
      <exclude-unlisted-classes>false</exclude-unlisted-classes>
      <properties>
         <property name="javax.persistence.jdbc.driver" value="com.mysql.jdbc.Driver"
/>
         <property name="javax.persistence.jdbc.url"
                        value="jdbc:mysql://localhost:3306/test" />
         <property name="javax.persistence.jdbc.user" value="root" />
         <property name="javax.persistence.jdbc.password" value="bison32" />
      </properties>
      <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>
   </persistence-unit>
</persistence>
```

7. Les Messages Driven Bean

Message driven Bean : interaction par messagerie
MOM : Message Oriented Middleware

Deux modes de communication

- N vers 1 : Queue
- N vers M : Topic

Message Driven Bean : repose sur la spécification JMS
JMS : Java Message Service

7. Les Messages Driven Bean

Principe des Message Driven Bean

- Consomme des messages asynchrones
- Pas d'état (toutes les instances d'une même classe de MDB équivalentes)
- Peut traiter les messages de clients
- 1 seule méthode métier (onMessage)
  - Paramètres imposés
  - Pas de valeur de retour
  - Pas d'exception

Quand utiliser un MDB

- Éviter appels bloquants
- Découpler clients (producteurs) et serveurs (consommateurs)
- Besoin de fiabilité : protection crash serveurs

7. Les Messages Driven Bean

La spécification JMS (java.sun.com/jms)

Queue : file de discussion (un seul consommateur)
Topic : sujet de discussion (diffusion)

ConnectionFactory : usine à connexions vers queue/topic
Connection : connexion vers queue/topic

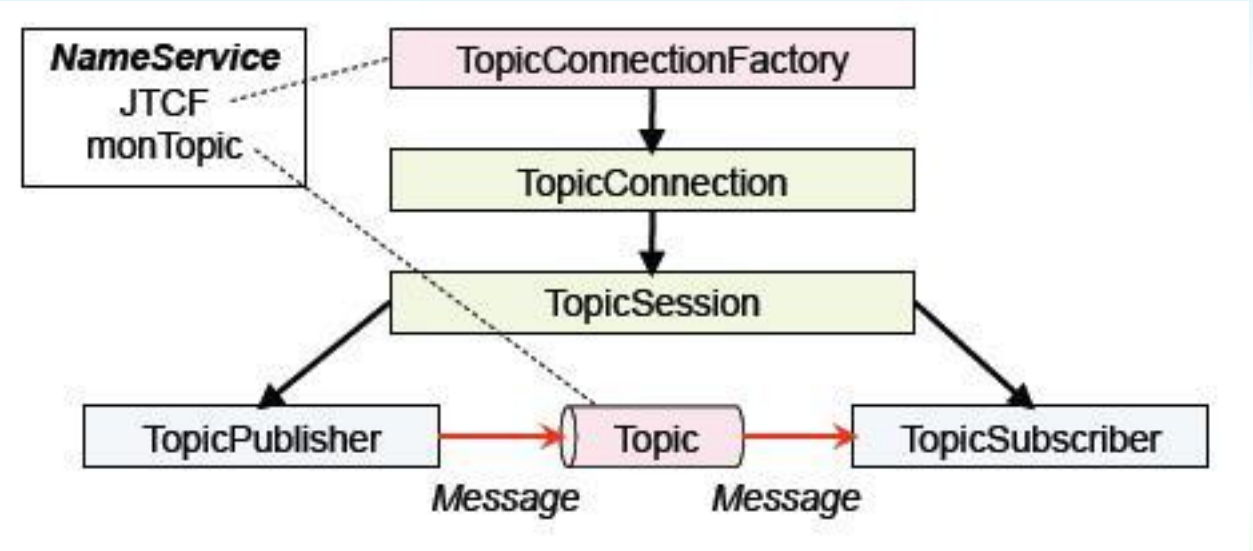
Session :

- Création d'émetteur et récepteur
- Peut être rendue transactionnelle

7. Les Messages Driven Bean

Architecture Java Message Service

(pour le queue, remplacer Topic par Queue, Publisher par Sender, Subscriber par Receiver)



7. Les Messages Driven Bean

Développement d'un Message Driven Bean

```
@MessageDriven(activationConfig = {
@ActivationConfigProperty(
propertyName = "destination",
propertyValue = "topic_rigolo"),
@ActivationConfigProperty(
propertyName = "destinationType",
propertyValue = "javax.jms.Topic")})

public class Mdb implements MessageListener {
public void onMessage(Message inMessage) {
System.out.println(((TextMessage)msg).getText());
}
}
```

7. Les Messages Driven Bean

Exemple d'émetteur (Bean)

```
@Resource(name="rigolo", mappedName="topic_rigolo")
Topic topic;
@Resource(name="factory", mappedName="JTCF")
TopicConnectionFactory factory;
TopicSession session; TopicPublisher sender;

public void publish(String value) {
TopicConnection tc = factory.createTopicConnection();
session = tc.createTopicSession(false,
Session.AUTO_ACKNOWLEDGE);
sender = session.createPublisher(topic);
TextMessage msg = session.createTextMessage();
msg.setText("MDB: " + value);
sender.publish(msg); }
```

8. Historique

Différence majeure entre version 2 et version 3

Présence de Home (Usine) dans les Session Beans

- Usine enregistrée dans le service de nommage
- Obtention d'un bean via l'usine (home.create())

Pas d'EntityManager

- Gestion de la persistance via la Home de l'EntityBean (homeFilm.findByName("Les Vikings"))

Entity Bean non sérializable

- Pas de notion de DTO (Data Transfert Object)
- Entity Bean toujours attaché

8. Historique

Version 2 :

- beaucoup de XML
- Difficile à maintenir

Version 3 :

- beaucoup d'annotation
- Maintenance simplifiée
- Possibilité d'utiliser du XML à la place…

9. Conclusion

Enterprise Java Bean

- Composants Java
- Programmation répartie
- Facile à mettre en oeuvre
- Très bonne amélioration de l'ergonomie avec la version 3

Fonctionnement interne

- Relativement compliqué (quatre cours pour l'étudier!)
- Demande de bonnes base

10.Webographie - Bibliographie et référence

- JEE Specification : java.sun.com/products/j2ee
- Enterprise Java Beans Specification 1.1, 2.0, 2.1 et 3.0 : java.sun.com/products/ejb
- Mastering Enterprise JavaBeans and the Java 2 Platform Enterprise Edition - Ed Roman Wiley
- Mastering Enterprise JavaBeans II - Ed Roman Wileyz
- EJB Fondamental - Ed Roman Eyrolles
- EJB3 Des concepts a l'ecriture du code - SupInfo Ed Dunod
- EJB3 in Action - Manning Publications