Systèmes d'Information Avancés (et répartis)

Université Lyon 1
M1 Miage soir

L. Médini, avril 2008

Plan des cours

- Protocole HTTP et programmation serveur
- Architectures réparties
- Objets distribués
  - Javabeans (survol)
  - EJB 2
    - EJB session
    - EJB entités
    - Exemples et descripteurs de déploiement
  - EJB 3 : POJO et annotations
- Web services (SOA, WSDL, SOAP, UDDI)
- Projet

Les JavaBeans (1996)

- Définition
  - *Composants logiciels réutilisables* d'applications **non réparties** (en pratique, des classes Java)
- Structure
  - Les propriétés sont cachées et accessibles par des méthodes publiques
    public String getNom() et
    public void setNom(String valeur)
  - Les autres méthodes sont privées
- Utilisation depuis une JSP
  - Déclaration : *<jsp:useBean class="package.NomBean" id="testbean" scope="application" />*
  - Accès : *<%= testbean.getProperty0() %>*

Les JavaBeans (1996)

- Communications entre beans : le modèle événementiel
  - Principe des « écouteurs » (*listeners*) Java : un objet s'enregistre comme écouteur d'un certain événement (interface *java.util.EventListener*)
  - Un bean peut d'émettre un événement (classe *java.util.EventObject*), « capté » par le ou les écouteurs.
  - ⇒ Les beans peuvent être assemblés en applications
- Persistance : un bean doit pouvoir être sauvegardé et restitué (en pratique, il doit implémenter *java.io.Serializable*)
- Interface : un bean peut être manipulé *visuellement* dans un outil d'aide à la construction d'applications (dans ce cas, étendre *java.awt.Component*)
- Introspection
  - les propriétés et événements des beans sont découverts par *introspection* par l'outil de construction d'application (classe *XXXBeanInfo* qui implémente *java.beans.BeanInfo*)
  - La découverte des beans peut être réalisée par une instance de la classe *java.beans.Introspector*

Les Enterprise JavaBeans (1998)

- Définition
  - Composant transactionnel accessible à distance
  - Représente une partie de la logique métier d'une application
- Un EJB expose aux clients différents types d'interfaces
  - Fonctionnelles (EJB 1 et 2)
    - Interfaces « Home »
      - Gestion du cycle de vie (création/destruction des instances)
    - Interfaces métier
      - Méthodes mises à disposition par la classe d'implémentation

    → Disparition des interfaces Home en EJB 3.0
  - En fonction du type d'accès (depuis EJB 2.0)
    - Locales
    - Distantes

Les Enterprise JavaBeans

- Un EJB est accessible à travers un **serveur d'applications**
- Dans ce serveur, un **conteneur d'EJB** permet
  - D'exécuter les méthodes présentes dans la **classe d'implémentation** de l'EJB (pattern IoC)
  - De faire l'interface avec le serveur d'applications pour la transmission des appels de ces méthodes
  - De gérer les aspects distribués (accès local ou distant)
  - De fournir des services aux EJB
    - *Via* un objet `EJBContext` spécialisé pour chaque type de bean
    - Services fournis : aspects middleware distribués, gestion du cycle de vie, sécurité (accès par les interfaces rendues disponibles par le conteneur), transactions, persistence…

Les Enterprise JavaBeans

- Trois types de beans
  - Beans session
    - Représentent les processus métiers
    - Ne peuvent avoir qu'un seul client à un moment donné
  - Beans entités
    - Permettent d'accéder aux données persistantes (SGBD...)
    - Fournissent une représentation de ces données sous forme d'objets
  - Beans messages
    - Peuvent échanger des messages asynchrone par l'intermédiaire de Java Message Service (JMS)
- Les beans entités et messages ne sont en général pas accédés par les clients mais par d'autres beans

Les Enterprise JavaBeans

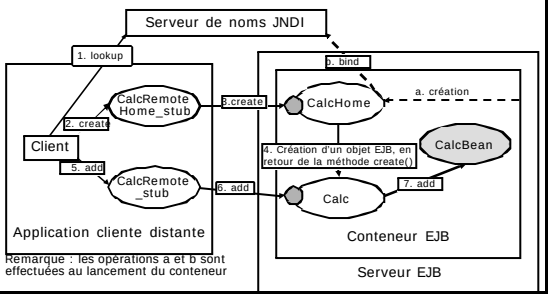
- Trois types de beans
  - Beans session
    - Représentent les processus métiers
    - Méthodes accessibles par le client
    - Ne peuvent avoir qu'un seul client à un moment donné
    - Deux types de Beans session
      - Avec états : c'est le même client qui réalise toutes les invocations (exemple : panier électronique)
      - Sans état : le Bean peut être utilisé successivement par plusieurs clients (exemple : calcul d'une distance)

Les Enterprise JavaBeans

- Historique
  - Mars 1998 : EJB 1.0
    - EJB session uniquement
  - Novembre 1999 : EJB 1.1
    - Sécurité, première version des EJB entités
  - Août 2001 : EJB 2.0
    - Interfaces locales et distantes, EJB messages, EJB-QL
  - Novembre 2003 : EJB 2.1
    - Modification de l'EJB-QL
  - Mai 2006 : EJB 3.0
    - Simplification du développement et du déploiement

Les EJB 2.0

- Fonctionnement du serveur et du client



Les EJB 2.0

- Cycle de vie
  - Pooling d'instances
    - Le conteneur gère un « pool » (réserve) d'instances qui lui évite de créer et détruire des objets pour chaque client
    - Le conteneur crée une instance de la classe d'implémentation d'un bean et la place dans le pool
      - À l'appel de la méthode create() par un client
      - S'il n'en a aucun de disponible
      - Dans la limite du nombre fixé par l'administrateur
    - Fonctionnement pour les différents types de beans
      - Les beans session sans état, une fois créés, restent opérationnels. Après utilisation, ils sont remis dans le pool ou détruits par le conteneur
      - Les beans session avec états sont désactivés (*ejbPassivate()*) et réactivés (*ejbActivate()*) en fonction des besoins du conteneur
      - Les beans entités sont également remis dans le pool après utilisation

EJB session

- Principes de base
  - Exemples de nommage pour un EJB session sans état Calc (calculatrice)
    - Interfaces home

```
public interface CalcHome extends EJBHome {
    public Calc create() throws RemoteException,
    CreateException; }
public interface CalcHomeLocal extends EJBLocalHome {
    public CalcLocal create() throws CreateException; }
```

    - Interfaces métier

```
public interface Calc extends EJBObject {
    public double add(double val1, double val2) throws
  RemoteException; }
public interface CalcLocal extends EJBLocalObject {
    public double add(double val1, double val2); }
```

EJB session

- Principes de base
  - Classe d'implémentation (nom : *CalcBean*)
    - Code de la logique métier
    - Implémente une interface spécifique au type d'EJB
      - *javax.ejb.SessionBean*
      - *javax.ejb.EntityBean*
      - *javax.ejb.MessageDrivenBean*

      } Dérivent de l'interface *javax.ejb.EnterpriseBean*
    - Contenu
      - Méthodes métiers : *add()*
      - Constructeur sans paramètre : *CalcBean()*
      - Méthode de création : *ejbCreate()*
      - Méthodes de gestion du cycle de vie : *ejbActivate()*, *ejbPassivate()*, *ejbRemove()*

EJB session

- Principes de base
  - Classe d'implémentation (nom : *CalcBean*)
    - Exemple

```
import javax.rmi.RemoteException;
import javax.ejb.SessionBean;
import javax.ejb.SessionContext;

public class CalcBean implements SessionBean {

   public double add(double val1, double val2) {
      return val1+val2; }
   public CalcBean() {}
   public void ejbCreate() {}
   public void setSessionContext(SessionContext sc) {}
   public void ejbActivate() {}
   public void ejbPassivate() {}
   public void ejbRemove() {}
}
```

EJB entités

- Principes généraux
  - Permettent l'accès aux données métier persistantes
    - BD
    - Système propriétaire
    - Système de stockage hétérogène
  - Chaque EJB entité représente un concept métier
    - Exemple : un compte en banque, un client, un achat…
  - ⇒ Il peut y avoir autant d'instances d'EJB entités que de données archivées

EJB entités

- Principes généraux
  - Données accessibles par des méthodes spécifiques
    - Mapping avec le support de persistance fait dans le descripteur de déploiement
    - Utilisation d'un langage de requêtes *ad hoc* : EJB-QL
  - Accessibles par plusieurs clients en même temps
    - Accès successifs sans transaction
    - Accès transactionnels simultanés
    - ⇒ Gestion des accès concurrents par le container

EJB entités

- Principes généraux
  - Ont une « identité »
    - Matérialisée par une classe de clé primaire (*Int*, *String*, *Object*)
  - Deux moyens de gérer la persistance
    - CMP : Container Managed Persistance
    - BMP : Bean Managed Persistance

EJB entités

- Persistance (gérée par le conteneur)
  - Rappels
    - un bean entité encapsule des données métier pour sécuriser leur utilisation
    - Chaque ensemble de données est représentée par un bean identifié (clé primaire)
    - Les données continuent d'exister après l'utilisation du bean
  - ⇒ Il faut pouvoir sauvegarder l'état d'un bean (sérialisation)
  - ⇒ La sauvegarde et la restauration des données sor des étapes critiques (gestion des transactions)

EJB entités

- Persistance (gérée par le conteneur)
  - États d'un bean entité
    - Inexistant
      - Levée d'une exception à partir de n'importe quelle méthode
      ⇒Doit être créé par une méthode newInstance(), suivie de setEntitycontext()
    - Dans le pool
      - Le bean est desactivé, et n'est associé à aucune donnée
      ⇒Peut être activé
    - Prêt
      - Le bean est associé à un objet entité déterminé (il connaît sa clé primaire)
      - Il peut exécuter des méthodes métier
      - Le conteneur appelle les méthodes *ejbLoad()* et *ejbStore()* pour gérer la synchronisation du bean avec le support de persistance

EJB entités

```
public interface Exemple extends EJBObject {

  public InfosExemple getInfosExemple() throws remoteException;
  public void setInfosExemple(InfosExemple ie) throws
remoteException;
  ...
}

public class InfosExemple implements java.io.Serializable {

  public final Integer champInt;
  public final String champString;

  public InfosExemple(Integer i, String s) {
    champInt = i;
    champString = s;
  }
}
```

EJB entités

```
public Collection findByCategorie(String categorie);

<entity>
 <display-name>ExempleEJB</display-name>
 <ejb-name>ExempleEJB</ejb-name>
 ...
 <abstract-schema-name>ExempleSN</abstract-schema-name>
 ...
<query>
 <query-method>
  <method-name>findByCategorie</method-name>
  <method-params>
   <method-param>java.lang.String</ method-param>
  </method-params>
 </query-method>
 <ejb-ql>
  select Object(a) from ExempleSN a where a.categorie = ?1
 </ejb-ql>
</query>
```

EJB entités

- Développement d'un bean CMP
  - L'interface home locale
    - Mêmes méthodes que l'interface distante *a priori*
    - Pas d'exception RemoteException

EJB entités

- Développement d'un bean CMP/BMP
  - La classe d'implémentation
    - Bean BMP : codage de la sérialisation dans cette classe
    - Bean CMP : classe déclarée abstraite pour que le conteneur puisse la sous-classer et implémenter les méthodes de gestion de la persistance
    - Contenu (BMP)
      - Déclaration des méthodes de gestion des champs
      - Méthodes métiers (déclarées dans les interfaces métier)
      - Constructeur sans paramètre (appelé par le conteneur)
      - Méthodes de création (déclarées dans les interfaces home)
      - Méthodes sans entités (déclarées dans les interfaces home)
      - Méthodes internes d'accès aux données (méthodes Select)
      - Méthodes de rappel (appelées par le conteneur)

EJB entités

- Développement d'un bean CMP
  - La classe d'implémentation
    - Déclaration des méthodes de gestion des champs

```
public abstract String getChampTexte();
public abstract void setchampTexte(String texte);
public abstract Integer getChampInt();
public abstract void setchampInt(Integer int);
```

EJB entités

- Développement d'un bean CMP
  - La classe d'implémentation
    - Déclaration des méthodes de gestion des champs

```
public abstract String getChampTexte();
public abstract void setchampTexte(String texte);
public abstract Integer getChampInt();
public abstract void setchampInt(Integer int);
```

    - Implémentation des méthodes métier

      Cf. beans session

    - Constructeur

      ...

EJB entités

- Développement d'un bean CMP
  - La classe d'implémentation
    - Méthodes de création
      - ejbCreateXxx() : initialise la création du bean
        - Met à jour les champs à partir des paramètres (utilise les *setters*)
        - Implémentation des méthodes déclarées dans les interfaces home
        - Type de la valeur de retour = type de la clé primaire
        - Doit retourner *null* (c'est le conteneur qui retrouve la clé primaire)

```
public Integer ejbCreateAvecUnParametre(Type1 param1)
  throws CreateException {
   if (param1==null) throw new CreateException;
   else setParam1(param1);
   return null; }
```

EJB entités

- Développement d'un bean CMP
  - La classe d'implémentation
    - Méthodes de création
      - ejbPostCreateXxx() : appelée une fois le support de persistance (BD) mis à jour
        - Traitements nécessaires pour finaliser la création du bean
        - Ne doivent pas obligatoirement comporter une clause *throw CreateException*
        - Type de retour void
        - Mêmes noms et paramètres que la méthode create() correspondante

```
public void ejbPostCreateAvecUnParametre(Type 1
  param1) { }
```

EJB entités

- Développement d'un bean CMP
  - La classe d'implémentation
    - Méthodes sans entité : ejbHomeXxx
      - Doivent être déclarées dans les interfaces Home
      - Permettent de faire des traitements de lots

        Ex : trouver combien de clients se sont connectés à une certaine date.
      - Peuvent utiliser des méthodes Finder ou des requêtes à la base
      - Peuvent nécessiter de nombreux accès aux données

      ⇒ À ne pas utiliser

EJB entités

- Développement d'un bean CMP
  - La classe d'implémentation

```
<query>
 <query-method>
  <method-name>
    ejbSelectObjetsCommeCa
  </method-name>
  <method-params>
   < method-param>java.lang.String</  method-param>
  </method-params>
 </query-method>
 <ejb-ql>
  select a.idDonnees from DonneesSN a where a.champCa = ?1
 </ejb-ql>
</query>
```

EJB entités

- Développement d'un bean CMP
  - La classe d'implémentation

```
public abstract class MonEJBBean implements javax.ejb.EntityBean {
  EntityContext ec = null;
  public void setEntitycontext(javax.ejb.EntityContext param) {
   ec = param; }
  public void unsetEntitycontext( ) {
   ec = null; }
  public void ejbActivate( ) { }
  public void ejbLoad( ) { }
  public void ejbStore( ) { }
  public void ejbRemove( ) { }
  public void ejbPassivate( ) { }

  […]
}
```

EJB entités

- Développement d'un bean CMP
  - La clé primaire
    - Permet d'identifier le bean de manière unique
    - Classe sérialisable
      - Dérivant de java.lang.Object (Integer, String…)
      - Spécifique au bean MonBeanId, mais la clé doit être composée d'un seul champ
      - Classe annexe sérialisable : méthodes à implémenter
        - Constructeur par défaut
        - public boolean equals(Object other)
        - public int hashcode ()
    - Le descripteur de déploiement doit indiquer
      - Le type de la classe (balise <prim-key-class>)
      - Le nom du champ (balise <primkey-field>)

Packaging

- Assembler tous les éléments dans un fichier ejb-jar
  - Interfaces métier et home (locales et/ou distantes)

Calc.class, *CalcLocal.class*, *CalcHome.class*, *CalsHomeLocal.class*

  - Classe d'implémentation
    CalcBean.class
  - Classe de clé primaire (beans entités)
    MyEntityBeanKey.class
  - Descripteur de déploiement
    ejb-jar.xml

Déploiement

- Réalisé par le serveur d'applications
  - Extraction du contenu du EAR/JAR
  - Génération du code déployé (code nécessaire à la communication du bean et de ses clients)
    - Objets EJB local et distant
      - Implémentent l'interface métier
      - Peuvent prendre en charge les fonctions de gestion de l'EJB (persistance, sécurité, transactions…)
    - Objets EJB Home local et distant
      - Implémentent l'interface home (méthode *create()*)
    - Stubs, skeletons, etc. en fonction du type de serveur
  - Enregistrement d'une référence à l'objet EJB Home dans un serveur de noms accessible via JNDI

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DD Enterprise JavaBeans
2.0//EN" "http://java.sun.com/dtd/ejb-jar_2_0.dtd">

<ejb-jar>
 <description>Descripteur de déploiement du bean Calc</description>
 <display-name>Calc</display-name>
 <enterprise-beans>
  <session>
   <description>Calculatrice simple</description>
   <display-name>Calc</display-name>
   <ejb-name>Calc</ejb-name>
   <home>calcul.CalcHome</home>
   <remote>calcul.Calc</remote>
   <ejb-class>calcul.CalcBean</ejb-class>
   <session-type>Stateless</session-type>
   <transaction-type>Container</transaction-type>
  </session>
 </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>
```

```
<!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>
  <entity>
   <ejb-name>ExempleBean</ejb-name>
   <home>com.demo.ExempleHome</home>
   <remote>com.demo.ExempleObject</remote>
   <ejb-class>com.demo.ExempleBean</ejb-class>
   <persistence-type>Container</persistence-type>
   <prim-key-class>java.lang.String</prim-key-class>
   <reentrant>False</reentrant>
   <abstract-schema-name>ExempleSN</abstract-schema-name>
   <cmp-field>
    <field-name>ExempleId</field-name>
   </cmp-field>
   <cmp-field>
    <field-name>ExempleData</field-name>
   </cmp-field>
   <primkey-field>ExempleId</primkey-field>
   <query>
    <query-method>
     <method-name>findByCategorie</method-name>
     <method-params>
      <method-param>java.lang.String</  method-param>
     </method-params>
    </query-method>
    <ejb-ql>select Object(a) from ExempleSN a where a.categorie = ?1</ejb-ql>
   </query>
   <query>...</query>
  </entity>
 </enterprise-beans>
 <assembly-descriptor>
  ...
```

Les EJB 2.0

- Programmation du client
  - Tâches à effectuer
    - Contact du serveur de noms
    - Récupération d'une référence sur l'interface home distante du bean
    - Création d'un objet EJB pour accéder au bean
    - Invocation des méthodes métier

```
import javax.naming.InitialContext;
import javax.naming.Context;
import javax.rmi.PortableRemoteObject;

public class CalcClient {
 public static void main(String args[]) {
  try {
   // Obtention du contexte initial par défaut
   Context initialContext = new InitialContext();

   //Recherche de l'interface home de distante de l'EJB
   Object objref = initialContext.lookup("Calc");
   CalcHome home = (CalcHome)PortableRemoteObject.narrow(objref,
  CalcHome.class);

   // Création et utilisation du bean
   Calc myCalc = home.create();
   System.out.println("3 + 2 = " + myCalc.add(3., 2.));

  } catch (Exception e) {
    e.printStackTrace(); }
 }
}
```

Les Enterprise JavaBeans

- Conclusion
  - Les EJB 2 sont des composants
    - Sûrs
    - Pratiques
    - Puissants
    - Extensibles
    - ... mais pas simples
  - ⇒ Nécessitent une modélisation approfondie de l'application
  - ⇒ Ne pas « sortir » du framework de génération

Les EJB 3.0

- Principe
  - Conserver/augmenter la puissance des EJB 2
    - Mêmes mécanismes sous-jacents
    - Intégration de fonctionnalités JEE5
  - Simplifier l'utilisation pour le développeur
    - Utilisation de POJOs
    - Annotations Java
    - Injection de dépendances
    - Utilisation de l'API Persistence 1.0 (mapping objet/relationnel)

Les EJB 3.0

- Simplifications
  - Disparition des interfaces Home et de leurs méthodes dans la classe d'implémentation
  - Utilisation d'annotations dans des classes Java standard (POJO)
  - Injection de dépendances (par *setters*) par le conteneur
  - Méthodes *callback interceptors* (appelées par des annotations) pour la gestion du cycle de vie
  - Classes *interceptor* permettant de regrouper tous les traitements liés au cycle de vie (POA)

Les EJB 3.0

- Classe d'implémentation
  - Classes Java standard (POJO)
  - Déclaration : `public` (mais pas `final` ni `static`)
  - Constructeur sans argument
  - Annotations permettant
    - L'injection de dépendances
    - L'appel des méthodes *callbacks interceptors*