Bases de données et environnements distribués
Chapitre V : Serveurs d'applications
Modèle de composants EJB

Éric Leclercq — révision Oct. 2017

Département IEM / Laboratoire LE2i

émail : Eric.Leclercq@u-bourgogne.fr
http://ludique.u-bourgogne.fr/leclercq
http://ufrsciencestech.u-bourgogne.fr

Plan du chapitre

Attention

Les exemples sont développés avec JDK 1.4 / J2EE 1.4

Architecture Logicielles

- Évolution des architectures logicielles :
  - modèle client-serveur.
  - les modèles objets distribués
  - de nombreuses limitations : problèmes de déploiement, de maintenance, de migration, etc.
- Vers des architectures applicatives multi-niveaux
  - Se concentrer sur les aspects métier des applications en découplant celle-ci des aspects présentation et accès aux données (non fonctionnels).
  - limiter l'impact des technologies intrusives souvent utilisées pour des besoins non fonctionnels
  - implanter les applications avec les serveurs d'applications pour assurer la *scalability*, la tolérance aux pannes etc.

Fonctionnalités d'un SA

Fonctionnalités généralement offertes par les serveurs d'applications :

- gestion des transactions
- sécurité et robustesse (reprise sur panne, redondance, etc.)
- équilibrage de charge
- implantation d'un modèle de composants COM ou EJB (Enterprise JavaBeans).
- administration, déploiement, déplacement des composants
- services d'annuaires évolués
- moteur de workflow
- etc.

Diversité des applications hébergée par un SA

Différents types d'applications clientes peuvent/doivent être développées dans des architectures de SI répartis :

- des clients spécifiques autonomes (y compris pour les tablettes et smartphones)
- des clients Java : applications autonomes, applet etc.
- des applications Web tenant compte des capacités du client

Discuter des avantages/inconvénients de chaque forme (de toute manière un choix arbitraire n'est pas envisageable)

Notion de composant

Un composant est un objet d'implémentation qui adhère à un modèle, il supporte un ensemble de méthodes standard qui lui permettent :

- d'être exécuté sur un serveur d'applications conforme au modèle
- d'invoquer des services proposés par le serveur d'applications : le mode transactionnel, la sécurité, la concurrence d'accès, l'indépendance à la localisation, etc.
- d'être facilement intégré à un environnement de développement (en phase de développement)

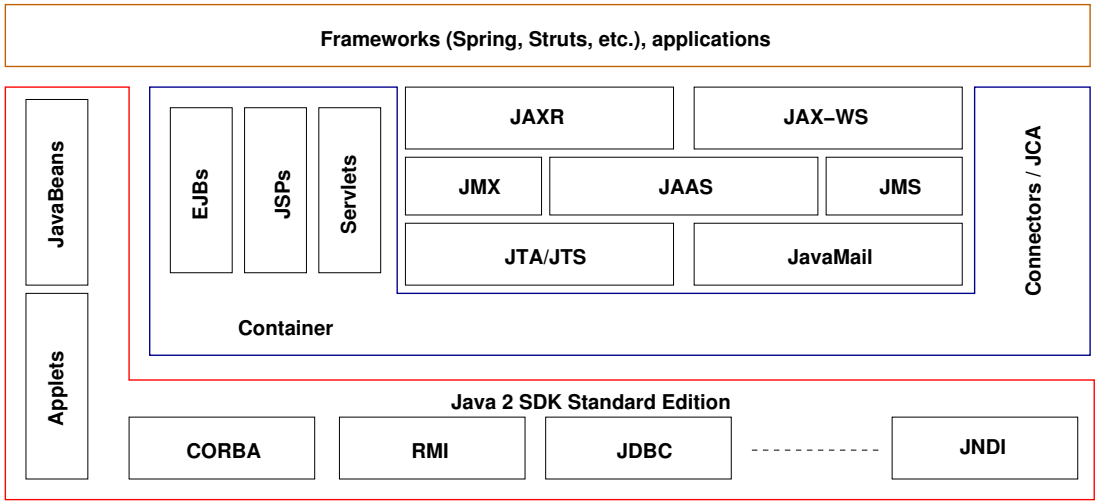
Les méthodes standard sont regroupées logiquement au moyen d'interfaces interfaces permettant ainsi aux composants d'être indépendants de leur implantation (relatif).

Modèles de composants

Il existe plusieurs modèles de composants cependant deux sont majoritaires :

- Le modèle COM (Component Object Model), issu de Microsoft et des technologies OLE (ActiveX). Indépendant du langage.
- Le modèle EJB (Enterprise JavaBean), spécification multi-éditeurs dont SUN et IBM sont à l'origine. Il repose sur le langage Java mais de plus en plus de langages sont proposés sur la JVM.
- Il existe aussi un modèle de composants pour CORBA (CORBA Component Model).

Aperçu de l'architecture



- JTA/JTS (Java Transaction API/Java Transaction Services)
- JCA (J2EE Connector Architecture)
- JMX (Java Management Extension) pour des applications de supervision d'applications
- JAAS (Java Authentication and Authorization Service) est une API de gestion de l'authentification et des droits d'accès.
- Eléments composant Java SE : `http://depinfo.u-bourgogne.fr/docs/j2sdk-1.6.30/docs/`

Les éléments du standard

Java 2 Enterprise Edition comprend la définition :

- des modes d'accès à un annuaire, aux bases de données
- du mode de dialogue entre les machines virtuelles Java.
- des interfaces qu'un composant doit présenter pour être réutilisable.

Un composant conforme aux services Java EE doit pouvoir s'exécuter dans n'importe quel environnement estampillé JEE (très théorique).

En réalité :

- de nombreux SA ne proposent qu'une une partie des services de Java EE
- peu parmi eux ont passé la totalité des tests de conformité Java EE
- les applications n'utilisent pas tous les services

Les spécifications sont donc implantées de manières inégales dans les SA et souvent en retard par rapport aux versions actualisées (travail long).

Les éléments du standard

- Les briques de base ont été élaborées par Sun.
- Au coeur de l'architecture, on trouve :
  - un intergiciel RMI/IIOP
  - des objets métiers qui sont des objets Java distribués les EJB (Enterprise Java Bean)
  - des services
- Il est possible aux éditeurs de soumettre des mises à jour des spécifications suivant un mécanisme bien défini.
- La force de Java EE tient surtout à son adoption par les grands éditeurs tels que BEA, IBM, ORACLE.

Version actuellement utilisé de Java EE : Java EE 6 même si la 7 est spécifiée. Documentation :
`http://java.sun.com/javaee/6/docs/tutorial/doc/`

Technologies et Services

Les technologies incluses dans Java EE, fournies sous formes d'API standards. Au niveau intergiciels on trouve :

- La communication entre objets distribués avec RMI
- L'intégration des objets CORBA avec JavaIDL et RMI/IIOP
- L'envoi de courriers électroniques avec JavaMail
- La communication asynchrone par messages entre objets distribués avec JMS (Java Message Service)
- La gestion des services Web avec JAX-WS ou JAX-RPC
- La recherche et l'enregistrement de configurations de ressources dans un annuaire avec JNDI (Java Naming and Directory Interface) ou annuaires de Web services avec JAXR

Technologies et Services

Au niveau de la gestion des données et de la persistance :

- L'accès aux bases de données avec JDBC
- La gestion des transactions avec JTA (Java Transaction API) et JTS (Java Transaction Service)
- La persistance avec JPA
- La création d'objets distribués transactionnels avec des EJB

Technologies et Services

Les services transversaux :

- La gestion des autorisation avec JAAS (Java Authentication and Authorisation Service
- Liaison Objets/Composants/Documents XML avec JAXB (Java Architecture for XML Binding)
- La gestion des interaction avec des systèmes propriétaires ou *legacy* avec JCA
- Le traitement XML Java API for XML Processing (JAXP), StAX (Streaming XML)
- La réalisation d'interfaces Web avec les pages JSP (Java Server Pages) et les servlets
- Framework web avec JSF (Java Server Faces)

Notion de conteneurs

La logique métier est implémentée dans les EJB qui sont des composants transactionnels côté serveur et accessibles à distance par différents types de clients.

- les EJB s'exécutent dans un serveur d'applications Java EE en proposant des services.
- Deux types de conteneurs sont généralement utilisés :
  - le conteneur Web est un environnement d'exécution pour les pages JSP et les servlets qui constituent une passerelle entre l'interface utilisateur et les EJB implémentant la logique métier.
  - le conteneur EJB est un environnement d'exécution pour les EJB. Le conteneur EJB héberge les composants métiers et leur fournit des services transversaux.
- Un composant EJB est conçu comme un ensemble réutilisable de services métiers et interagit avec tous les types de clients : servlets, JSP, application Java/RMI, CORBA, etc.

Offre logicielle

Conteneurs Web

- Apache TOMCAT (`http://tomcat.apache.org/`)
- Jetty (`http://www.eclipse.org/jetty/`)

Serveur d'application EJB :

- Apache Geronimo (`http://geronimo.apache.org`)
- GlassFish d'origine Sun Microsystems très bien intégré à NetBeans
- JBoss Enterprise Application Platform
- WebLogic Application Server, Oracle Corporation (developpé par BEA Systems)
- WebSphere Application Server, IBM
- Jonas (`http://jonas.ow2.org`)

Les spécifications EJB

- La spécification Enterprise JavaBeans 1.1 définit 2 types de beans : bean entité et bean session.
- La spécification Enterprise JavaBeans 2.0 introduit un troisième bean : le bean message. Il existe actuellement 3 types principaux de beans :
  - les beans sessions
  - les beans entités
  - les beans messages
- Les beans session proposent des beans avec ou sans états (statefull, stateless)
- Les beans entité proposent un persitance gérée par le conteneur ou le bean lui même (CMP et BMP)
- Les beans sessions et entité sont des composants distribués invoqués par des clients de manière synchrone (par invocation de méthodes)

Les beans session

- Un bean session représente un processus métier (ou une partie) : c'est une extension du processus du client dans le SA
- Deux types de beans sessions : les beans sessions sans état (stateless session) et les beans sessions avec état (statefull session).
- Un bean session ne peut avoir qu'un client à un instant donné
- Pour un bean session **sans état, plusieurs clients** peuvent être associés au même bean successivement : il représente un traitement fonctionnel, comme le calcul ou une demande de virement entre 2 comptes.

Les beans session

- Pour un bean session **avec état, c'est le même client** qui réalise toutes les invocations
- Un bean session avec état possède un état conversationnel, c'est-à-dire un état résultant des interactions avec son client
- L'état représenté par un bean session est privé et est accessible par un client unique.

Exemple :

bean session avec état

- *un panier sur un site de commerce électronique possède deux attributs : un pour le nom du client et un pour la liste des articles sélectionnés par ce client.*
- *ce bean posséde une méthode `ajouterArticle()` que le client va invoquer pour ajouter un nouvel article dans le panier*

Les beans session

- Un bean entité représente un objet métier persistant :
  - une commande,
  - un article, ses caractéristiques et son stock,
  - un compte bancaire.
- Les attributs d'un bean entité peuvent être stockés dans une base de données ou dans tout autre support de persistance.
- Un bean entité peut être utilisé par plusieurs clients simultanément.
- La persistance d'un bean entité peut être gérée par le bean lui-même ou bien par le conteneur.

L'état représenté par un bean entité est partagé et transactionnel, c'est le point d'accès unique à ces données par différents clients.

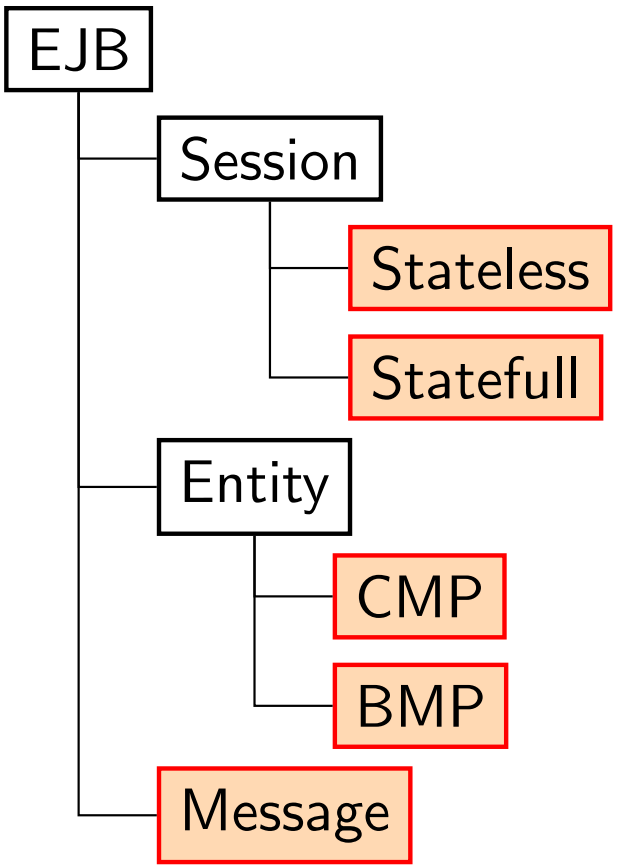
Les beans session

- **bean BMP (Bean Managed Persistency) :** les opérations de lecture et d'écriture sur le support de persistance doivent être codées dans le bean (par exemple du code JDBC)
- **bean CMP (Container Managed Persistency) :**
  - la persistance est gérée par le conteneur
  - seules les fonctionnalités de recherches doivent être codées dans le bean, (ou dans un objet DAO)
  - les autres opérations de lecture et d'écriture sur le support de persistance sont effectués automatiquement par le conteneur.

Les beans messages

- Les beans messages sont des beans qui consomment des messages de manière asynchrone, par l'intermédiaire de Java Messaging Service (JMS).
- Ils permettent d'implanter un processus métier dans avoir recours à un moteur de workflow
- Ils permettent de s'adapter à un environnement mobile, etc.

Synthèse



Interfaces pour l'implémentation des beans

- Les beans implémentent des interfaces qui spécifient les services qu'ils exposent aux clients
- Les clients utilisent ces interfaces pour manipuler les beans
- Il existent 2 types d'interfaces : les interfaces distances et les interfaces locales.
  - les interfaces distantes spécifient les méthodes accessibles depuis l'extérieur du SA
  - les interfaces locales spécifient les méthodes accessibles à l'intérieur du SA = cycle de vie
- Du point de vue implementation :
  - Les `HomeInterface - EJBHome` permettent de créer, détruire et recherche des objets du même type
  - Les `ComponentInterface - EJBOject - Remot` sont relatives au métier et permettent d'exécuter les méthodes.

Intergiciel explicite et implicite

- Les intergiciels étudiés jusqu'à présent sont utilisés de manière explicite par le programmeur
- Le code d'accès à l'API de l'intergiciel est mélangé avec le code métier.
- Pour un intergiciel implicite :
  - la déclaration des services nécessaires (transactions, persistance, sécurité) à l'objet distribué se fait dans un fichier de description distinct
  - un outil associé permet de générer un objet intercepteur d'appels .
  - cet objet intercepte les appels clients, exécute les services (transaction, sécurité, persistance) puis délègue ces appels à l'objet distribué

Intergiciels explicite et implicite

Dans l'architecture Java EE, le conteneur EJB intercepte les appels pour exécuter automatiquement les services de l'intergiciel.

- Le conteneur est donc une couche d'indirection/interception entre le code client et le bean, utilisation d'un intercepteur d'appels nommé objet EJB (`EJBObject`).
- Les objets sont dépendants du conteneur
- Le client traite donc avec l'objet EJB et non directement avec le bean.
- Le client fait appel à une fabrique d'objets EJB qui est chargée d'instancier et de détruire les objets EJB.
- La fabrique est appelée objet d'accueil ou `EJBHome`.
- Le composant ou bean EJB est donc un composant côté serveur entièrement géré et distribué par le conteneur.

Services du conteneur

Le conteneur fournit des services/comportement aux composants qui les déclarent :

- Persistance sans inclure du code SQL dans le code métier
- Transaction déclaratives sans utilise JTA
- Sécurité déclarative
- Mémoire cache, pools de connexions, activation
- Gestion des erreur et restauration des connexions
- etc.

C'est le rôle en partie du descripteur de déploiement et du conteneur qui s'interpose entre l'EJB et les couches de communication.

Notion de descripteur de déploiement

À un composant est associé un descripteur de déploiement, pour signifier les services non fonctionnels qu'il utilise :

- par exemple un EJB doit être persistant, sécurisé et accessible par plusieurs clients en même temps.
- le serveur (le conteneur) est responsable du traitement de ce descripteur et assure l'appel ou l'exécution de ces services.

Ainsi, un composant EJB est constitué d'une ensemble de classes Java et d'un fichier XML, fusionnés en une entité unique.

Bean session sans état

Objectif : comprendre les différents éléments mis en œuvre et les différentes étapes du processus de création du composant.

- Coté serveur :
  - Interface de l'objet EJB : `HelloWorld.java`
  - Bean, définissant les méthodes appelables : `HelloWorldBean.java`
  - Interface de fabrique du Bean : `HelloWorldHome.java`
  - Description de l'objet permettant son déploiement `ejb-jar.xml`
  - Description de l'accès au Bean : `jonas-ejb-jar.xml`
- Coté client :
  - Le programme client : `HelloClient.java`
  - Certaines portion du bean serveur

Bean session sans état

Code : HelloWorld.java

```
1  package helloWorld;
2
3  import java.rmi.RemoteException;
4  import javax.ejb.EJBObject;
5
6  public interface HelloWorld extends EJBObject{
7    public String sayHello() throws RemoteException;
8  }
```

Bean session sans état

Code : HelloWorldHome.java

```
1  package helloWorld;
2
3  import java.rmi.RemoteException;
4
5  import javax.ejb.CreateException;
6  import javax.ejb.EJBHome;
7
8  public interface HelloWorldHome extends EJBHome{
9    public HelloWorld create() throws CreateException
         , RemoteException;
10 }
```

Bean session sans état

Code : HelloWorldBean.java

```
1  package helloWorld;
2  import java.rmi.RemoteException;
3  import javax.ejb.EJBException;
4  import javax.ejb.SessionBean;
5  import javax.ejb.SessionContext;
```

Bean session sans état

Code : HelloWorldBean.java

```
1   package helloWorld;
2   import java.rmi.RemoteException;
3   import javax.ejb.EJBException;
4   import javax.ejb.SessionBean;
5   import javax.ejb.SessionContext;
6   public class HelloWorldBean implements SessionBean{
7     protected SessionContext ctx;
8     private static final long serialVersionUID = 1L;
9     public String sayHello(){
10      return "Hello, world !";
11    }
```

Bean session sans état

Code : HelloWorldBean.java

```
1  package helloWorld;
2  import java.rmi.RemoteException;
3  import javax.ejb.EJBException;
4  import javax.ejb.SessionBean;
5  import javax.ejb.SessionContext;
6  public class HelloWorldBean implements SessionBean{
7    protected SessionContext ctx;
8    private static final long serialVersionUID = 1L;
9    public String sayHello(){
10     return "Hello, world !";
11   }
12   public void ejbCreate() {}
13   public void ejbActivate() throws EJBException, RemoteException {}
14   public void ejbPassivate() throws EJBException, RemoteException {}
15   public void ejbRemove() throws EJBException, RemoteException {}
16   public void setSessionContext(SessionContext arg0)
17     throws EJBException, RemoteException {
18     ctx = arg0;
19   }
20 }
```

Bean session sans état

Le bean implémente aussi les méthodes spécifiques du `SessionBean` :

- `ejbCreate()` : appel lors de la création du bean session
- `ejbActivate()` : appel lors de l'activation du bean (après sa création, avant son utilisation)
- `ejbPassivate()` : appel lors de la désactivation du bean (après la fin de son utilisation)
- `ejbRemove()` : appel lors de la suppression du bean session

Bean session sans état

Code : HelloWorldHome.java

```
1  package helloWorld;
2
3  import java.rmi.RemoteException;
4  import javax.ejb.CreateException;
5  import javax.ejb.EJBHome;
6
7  public interface HelloWorldHome extends EJBHome
8  {
9    public HelloWorld create()
10           throws CreateException, RemoteException;
11 }
```

Bean session sans état

Code : ejb-jar.xml

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <ejb-jar>
3    <description>HelloWorld deployment descriptor</description>
4    <display-name>HelloWorld</display-name>
5    <enterprise-beans>
6      <session>
7        <description>HelloWorld deployment descriptor</description>
8        <display-name>HelloWorld</display-name>
9        <ejb-name>HelloWorld</ejb-name>
10       <home>helloWorld.HelloWorldHome</home>
11       <remote>helloWorld.HelloWorld</remote>
12       <ejb-class>helloWorld.HelloWorldBean</ejb-class>
13       <session-type>Stateless</session-type>
14       <transaction-type>Container</transaction-type>
15     </session>
16   </enterprise-beans>
17   <assembly-descriptor>
18     <container-transaction>
19       <method>
20         <ejb-name>HelloWorld</ejb-name>
21         <method-name>*</method-name>
22       </method>
23       <trans-attribute>Required</trans-attribute>
24     </container-transaction>
25   </assembly-descriptor>
26 </ejb-jar>
```

Bean session sans état

Code : jonas-ejb-jar.xml

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <jonas-ejb-jar>
3    <jonas-session>
4      <ejb-name>HelloWorld</ejb-name>
5      <jndi-name>myHelloWorld</jndi-name>
6    </jonas-session>
7  </jonas-ejb-jar>
```

Déploiement

- Réalisé au moyen d'un fichier `.jar`
- Les `.war` (Web ARchive) sont utilisés pour les applications web (jsp et servlet)
- Les `.ear` (Enterprise ARchive) contiennent des fichiers `jar` et `war`
- Peut être créé manuellement par la commande `jar cvf Hello.jar *` exécutées dans le répertoire contenant les fichiers `.class` et les fichiers `.xml`
- Structure du répertoire à packager :

```
META-INF/MANIFEST.MF
META-INF/ejb-jar.xml
helloWorld/HelloWorldBean.class
helloWorld/HelloWorldHome.class
helloWorld/HelloLocalHome.class (*)
helloWorld/HelloLocal.class (*)
helloWorld/HelloWorld.class
```

Bean session sans état

Code : HelloClient.java

```
1  package client;
2  import helloWorld.HelloWorld;
3  import helloWorld.HelloWorldHome;
4  import javax.naming.InitialContext;
5  import javax.naming.Context;
6  import javax.rmi.PortableRemoteObject;
7
8  public class HelloClient{
9    public static void main(String[] args){
10     Context initialContext = null;
11     try{ initialContext = new InitialContext();}
12     catch (Exception e){
13       System.err.println("Cannot get initial context for JNDI: " + e);
14       System.exit(2);
15     }
16     try{
17       Object objref = initialContext.lookup("myHelloWorld");
18       HelloWorldHome home = (HelloWorldHome)PortableRemoteObject.narrow(
19           objref,
20           HelloWorldHome.class);
21       HelloWorld myHelloWorld = home.create();
22       String message = myHelloWorld.sayHello();
23       System.out.println(message);
24     } catch (Exception e){
25       System.err.println(" Erreur : " + e);
26       System.exit(2);
27     }
28   }
29 }
```

Pour aller plus loin : OSGi les EJB 3

mais aussi gestion de la persistance et des transactions
A suivre...

Bean et transactions

- Java Transaction API (JTA) est une des APIs de JEE.
- JTA spécifie les interfaces permettant de gérer les transactions distribuées (globale) ou locales
- Un transaction distribuée est réparties sur plusieurs gestionnaires transactionnels (de ressources)
- L'implémentation de JTA est fournie par le serveur application ou par le SGBD.
- Pour les applications stand-alone ou les servlets on s'appuiera sur le gestionnaire de transactions du serveur d'applications

Bean et transactions

Rappel sur les classes principales de JTA :

- `UserTransaction` : permet de définir et d'interagir avec une transaction au moyen des méthodes usuelles `begin()`, `commit()`, `rollback()`, `getStatus()`
- `TransactionManager` : offre la possibilité de mettre en pause et de redémarrer une transaction (similaire à `UserTransaction`
- `Status` : pour récupérer le statut de la transaction.

Bean et transactions : déclarer les transactions

Mode impératif : pour les EJB, on parle de Bean Managed Transaction et le développeur utilise directement sur l'API JTA pour définir et contrôler les transactions.

Extrait de BMT en EJB 3

```
1   @Stateless
2   @TransactionManagement(TransactionManagementType.BEAN)
3   public class UserServiceImpl {
4      UserDAO userDAO;
5      public void update(User user) throws Exception {
6         InitialContext context = new InitialContext();
7         // service du ServApp
8         UserTransaction transaction = (UserTransaction)context.lookup("UserTransaction
               ");
9         try {
10           transaction.begin();
11        userDAO.update(user);
12           transaction.commit();
13        } catch (Exception up) {
14           transaction.rollback();
15           throw up;
16        }
17     }
18  }
```

Bean et transactions : déclarer les transactions

Mode déclaratif : le développeur indique, via des annotations, les éléments transactionnels de sa méthode.

Extrait de CMT en EJB 3

```
1   @Stateless
2   public class UserServiceImpl {
3      UserDAO UserDAO;
4      @Resource SessionContext context;
5      @TransactionAttribute(TransactionAttributeType.REQUIRED)
6      public void update(User user) throws Exception {
7         try {
8        userDAO.update(user);
9         } catch (Exception up) {
10           context.setRollbackOnly();
11           throw up;
12        }
13     }
14  }
```

Attention au comportement vis à vis du rollback sur exception `http://www.ibm.com/developerworks/java/library/j-ts1/j-ts1-pdf.pdf` et `http://blog.xebia.fr`

Principes

- Les EJB 3 sont présents depuis la spécification Java 1.5
- La spécification est définie dans la JSR 220 et contient trois éléments décrits de manière séparée `https://jcp.org/en/jsr/detail?id=220` :
  - Le noyau
  - Le mécanisme de persistance (*persistence provider*) comme par exemple :
    - Hibernate EntityManager `http://www.hibernate.org`
    - Apache OpenJPA `http://openjpa.apache.org/`
    - TopLink Essentials https ://glassfish.dev.java.net/javaee5/persistence/
    - Eclipse Link `http://www.eclipse.org/eclipselink/`
  - Les fonctionnalités annexes
- Motivations :
  - EJB 2.x trop complexes
  - les développeurs utilisaient des outils additionnels comme XDoclet (Attribute Oriented Programming) `http://xdoclet.sourceforge.net` Hibernate pour la persistance : `http://www.hibernate.org`

Avantages

- Les descripteurs de déploiement ne sont plus nécessaires : tout peut être réalisé avec des annotations
- Les beans de type CMP (*Container Managed Persistence*) ont été simplifiés et utilisent les mécanismes développés dans Hibernate et JDO
- Certaines bonnes pratiques ont été définies par défaut par exemple, le modèle de transaction
- L'utilisation des exceptions et leur interception ont été réduites
- L'héritage est maintenant permis, un beans peut étendre un code de base
- Les requêtes SQL natives sont supportées via le langage EJB-QL (*Query Language*) et ses extensions

Nouvelles fonctionnalités

- Annotations ou méta-données peuvent être utilisées pour définir les éléments non fonctionnels.
- Exemple : pour un stateless session bean, l'annotation `@Stateless` est déclarée au niveau de la classe du bean
- Ajoute des *Business Interceptors* :
  - permet au développeur d'intercepter les méthodes métier
  - pour modifier les paramètres ou les valeurs de retour
  - utilisable pour de la traçabilité ou de l'analyse de performance.
- *Lifecycle Interceptors* pour effectuer des actions liées au cycle de vie des beans. Les callback EJB 2 `ejbActivate()` sont définis au moyen d'annotation `@PostActivate` par exemple posée sur une méthode qui sera appelée par le conteneur.

Injection de dépendances

L'injection de dépendances (*Dependency injection*) permet de demander au conteneur d'injecter une ressources au lieu de la créer ou de la rechercher.

- Avec les EJB le code suivant était nécessaire :

Couplage et dépendances

```
1   try {
2   Object o = new InitialContext().lookup("java:comp/env/ejb/MyEJB");
3   myBean = PortableRemoteObject.narrow(o, MyInterface.class);
4   } catch (NamingException e) {
5   ....
6   }
```

- Avec les EJB 3 :

Injection de dépendances

```
1   @EJB private MyInterface myBean;
```

Exemple de Bean

Le bean `HelloWorld` est divisé en deux parties : l'interface métier et la classe qui implémente l'interface.

Couplage et dépendances

```
1  package org.objectweb.easybeans.tutorial.helloworld;
2  public interface HelloWorldInterface {
3  void helloWorld();
4  }
```

Code métier

```
1  package org.objectweb.easybeans.tutorial.helloworld;
2  public class HelloWorldBean implements HelloWorldInterface {
3  public void helloWorld() {
4  System.out.println("Hello␣world␣!");
5  }
6  }
```

Ni la communication RMI, ni la définition du bean ne figurent à cette étape !

Exemple de Bean (la définition du bean)

Le bean sera un stateless session bean, la classe sera annotée avec `@Stateless`, l'interface doit être déclarée `remote` en utilisant l'annotation `@Remote`.

Code métier

```
1   package org.objectweb.easybeans.tutorial.helloworld;
2   import javax.ejb.Remote;
3   import javax.ejb.Stateless;
4
5   @Stateless
6   @Remote(HelloWorldInterface.class)
7   public class HelloWorldBean implements HelloWorldInterface {
8    public void helloWorld() {
9     System.out.println("Hello␣world␣!");
10  }
11  }
```

Pour déployer, compiler les classes `HelloWorldInterface` et `HelloWorldBean`, créer un répertoire `ejb3s/helloworld.jar/` et placer les classes dedans. Elles seront chargées et déployées.

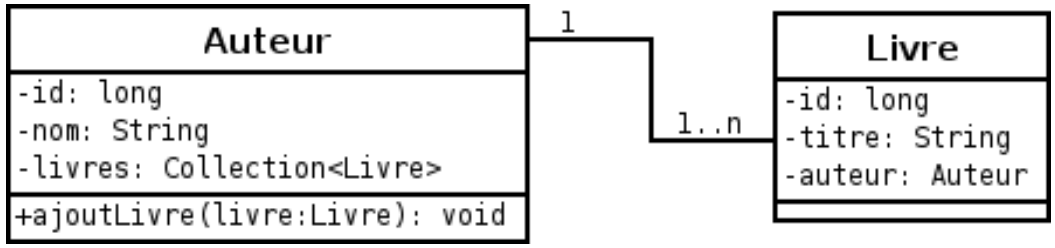
Exemple client

Code métier

```
1  package org.objectweb.easybeans.tutorial.helloworld;
2  import javax.naming.Context;
3  import javax.naming.InitialContext;
4
5  public final class Client {
6  private static final String JNDI_NAME =
7    "org.objectweb.easybeans.tutorial.helloworld.HelloWorldBean"
8    + "_" + HelloWorldInterface.class.getName() + "@Remote"
9
10 private Client() {
11 }
12 /**
13 * Main method.
14 * @param args the arguments (not required)
15 * @throws Exception if exception is found.
16 */
17 public static void main(final String[] args) throws Exception {
18   Context initialContext = new InitialContext();
19   HelloWorldInterface businessItf =
20      (HelloWorldInterface) initialContext.lookup(JNDI_NAME);
21   System.out.println("Calling␣helloWorld␣method...");
22   businessItf.helloWorld();
23 }
24 }
```

Pas de `create()`, pas de transtypage...

Par l'exemple (d'après Florent Benoit - Bull)



Code de l'entity Bean

```
1   @Entity
2   @NamedQuery(name = "tousLesAuteurs", query = "select␣o␣FROM␣Auteur␣o")
3   public class Auteur implements Serializable {
4    private static final long serialVersionUID = 827953655456812069 5L;
5    private long id;
6    private String nom = null;
7    private Collection<Livre> livres;
8    public Auteur() {
9      livres = new ArrayList<Livre>();
10      }
11  /**
12   * Constructeur utilise pour initialiser cet entity bean.
13   * @param nom - le nom de l'auteur.
14   */
15      public Auteur(final String nom) {
16          this();
17          setNom(nom);
18      }
```

Par l'exemple (d'après Florent Benoit - Bull)

Code de l'entity Bean

```
1       @OneToMany(mappedBy = "auteur", fetch = FetchType.EAGER, cascade = CascadeType.
            ALL)
2       public Collection<Livre> getLivres() {
3           return livres;
4       }
5       public void ajoutLivre(final String titre) {
6           Livre livre = new Livre();
7           livre.setTitre(titre);
8           livre.setAuteur(this);
9           livres.add(livre);
10      }
11      public void setLivres(final Collection<Livre> livres) {
12          this.livres = livres;
13      }
14      public String getNom() {
15          return nom;
16      }
17      public void setNom(final String nom) {
18          this.nom = nom;
19      }
20      @Id
21      @GeneratedValue(strategy = GenerationType.AUTO)
22      public long getId() {
23          return this.id;
24      }
25      public void setId(final long id) {
26          this.id = id;
27      }
28  }
```

Le Bean Livre

Code de l'entity Bean

```
1   @Entity
2   @NamedQuery(name = "tousLesLivres", query = "select␣o␣FROM␣Livre␣o")
3   public class Livre implements Serializable {
4    private static final long serialVersionUID = 7870634228111717036L;
5    private long id;
6    private Auteur auteur;
7    private String titre;
8    public Livre() {}
9    public Livre(final String titre, final Auteur auteur) {
10          setTitre(titre);
11          setAuteur(auteur);
12   }
13   @ManyToOne
14   @JoinColumn(name = "Auteur_id")
15   public Auteur getAuteur() {
16          return auteur;
17   }
18    public void setAuteur(final Auteur auteur) {
19       this.auteur = auteur;
20   }
21  ...
22   @Id
23   @GeneratedValue(strategy = GenerationType.AUTO)
24   public long getId() {
25       return this.id;
26   }
27   public void setId(final long id) {
28          this.id = id;
29   }
```

Le Bean Livre

Définir le gestionnaire de persistence

```
1  <persistence xmlns="http://java.sun.com/xml/ns/persistence"
2               xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3               xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
4  ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd">
5
6  <persistence-unit name="entity">
7     <provider></provider>
8     <jta-data-source>jdbc_1</jta-data-source>
9     <properties>
10    <property name="hibernate.hbm2ddl.auto" value="create-drop"/>
11    </properties>
12 </persistence-unit>
13 </persistence>
```

Utiliser un entity bean

- L'accès aux entity bean est réalisé à travers un entity manager
- L'entity manager peut être accédé par un stateless session bean

Facade via SB

```
1  @Remote
2  public interface SessionFacadeRemote {
3      void init();
4      List<Auteur> listeDesAuteurs();
5      List<Livre> listeDesLivres();
6  }
```

- L'interface est utilisée pour accéder au bean depuis un client distant (`@Remote`)

Utiliser un entity bean

- L'injection de ressource ou de dépendance utilisée dans le bean stateless pour injecter un entity manager

Facade via SB

```
1   @Stateless
2   public class SessionFacade implements SessionFacadeRemote {
3       @PersistenceContext
4       private EntityManager entityManager = null;
5       public void init() {
6           Auteur balzac = new Auteur("Honore de Balzac");
7           Livre pereGloriot = new Livre("Le Pere Goriot", balzac);
8           balzac.getLivres().add(pereGloriot);
9           Livre lesChouans = new Livre("Les Chouans", balzac);
10          balzac.getLivres().add(lesChouans);
11
12          // Enregistrement (seulement l'auteur car il y a l'attribut Cascade est on).
13          entityManager.persist(balzac);
14
15          // Livres victor hugo
16          Auteur hugo = new Auteur("Victor Hugo");
17          hugo.ajoutLivre("Les Miserables");
18          hugo.ajoutLivre("Notre-Dame de Paris");
19          // Enregistrement
20          entityManager.persist(hugo);
21
22          // should be done by the server
23          entityManager.joinTransaction();
24      }
```

Utiliser un entity bean

Facade via SB

```
1       @SuppressWarnings("unchecked")
2        public List<Auteur> listeDesAuteurs() {
3            Query auteurs = entityManager.createNamedQuery("tousLesAuteurs");
4            return auteurs.getResultList();
5        }
6
7        @SuppressWarnings("unchecked")
8        public List<Livre> listeDesLivres() {
9            return entityManager.createNamedQuery("tousLesLivres").getResultList();
10       }
11  }
```

Client final

Facade via SB

```
1   Context initialContext = new InitialContext();
2     SessionFacadeRemote facadeBean = (SessionFacadeRemote) initialContext.lookup(
          jndiName);
3
4     facadeBean.init();
5
6   // Recuperation de la liste des auteurs
7   List<Auteur> auteurs = facadeBean.listeDesAuteurs();
8
9   // Liste par auteur des livres ecrits par ceux-ci.
10  if (auteurs != null) {
11      for (Auteur auteur : auteurs) {
12          System.out.println("Liste␣de␣livres␣pour␣l'auteur␣dont␣le␣nom␣est␣'" +
                auteur.getNom() + "'␣:");
13          Collection<Livre> livres = auteur.getLivres();
14          if (livres == null) {
15              System.out.println("-␣Aucun␣livre.");
16          } else {
17              for (Livre livre : livres) {
18                  System.out.println("-␣Livre␣ayant␣pour␣titre␣'" + livre.getTitre() +
                        "'.");
19              }
20          }
21      }
22  } else {
23      System.out.println("Aucun␣auteur.");
24  }
```

Références

- Java EE 5, Antonio Goncalves, Eyrolles, 340 pages, 3eme édition, 2011
- J2EE 1.4, James L. Weaver , Kevin Mukhar , Jim Crume, Eyrolles, 640 pages
- J2EE, Jérôme Molière, Eyrolles, 220 pages, 2eme édition
- Le cours de Claude Duvallet : `brice.lecomte.free.fr/cours/master2/concepWeb/COURS-EJB.pdf`
- Le guide : `http://julien.coron.free.fr/languages/Java/EJB/` de Julien Coron pour développer votre premier EJB sur Jonas, nouvelle URL : `http://julien.coron.chez.com/languages/Java/EJB/`.