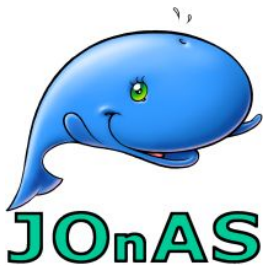




JOnAS & EasyBeans  
*EMN – 29 Novembre 2007*



# Plan

- Présentation Plate-forme Java EE™ 5
- Serveur d'applications JOnAS / EasyBeans
- EJB 3.0 (Description, Nouvelles fonctionnalités)
- Développer des applications EJB 3.0
  - Stateless
    - Gestion des intercepteurs, Debug
  - Stateful
  - Persistance
  - MDB
  - Migration Hibernate → EJB 3.0 JPA



Plate-forme JAVA EE

# Java EE™: Objectifs

## Java™ Platform, Enterprise Edition

- Successeur de la plate-forme J2EE (Java 2 Enterprise Edition)
- Simplifie le développement d'applications avec gestion de la sécurité, robustesse et l'interopérabilité.
- Support d'applications critiques :
  - Fourniture de services : *HA, passage à l'échelle, fiabilité, sécurité*
- Support de l'ensemble du cycle de vie d'une application
  - Composant, assemblage des applications, gestion du déploiement
- Libère le développeur du code non applicatif.
  - Transaction, persistance, sécurité, répartition de charge, distribution, gestion de pools (connexions (JDBC), instances, threads), cycle de vie => Serveur d'applications
- Applications portables
  - Fonctionne sur n'importe quel OS, JVM, DB & serveur d'applications.

# Serveurs d'applications J2EE/Java EE

## ■ Open Source

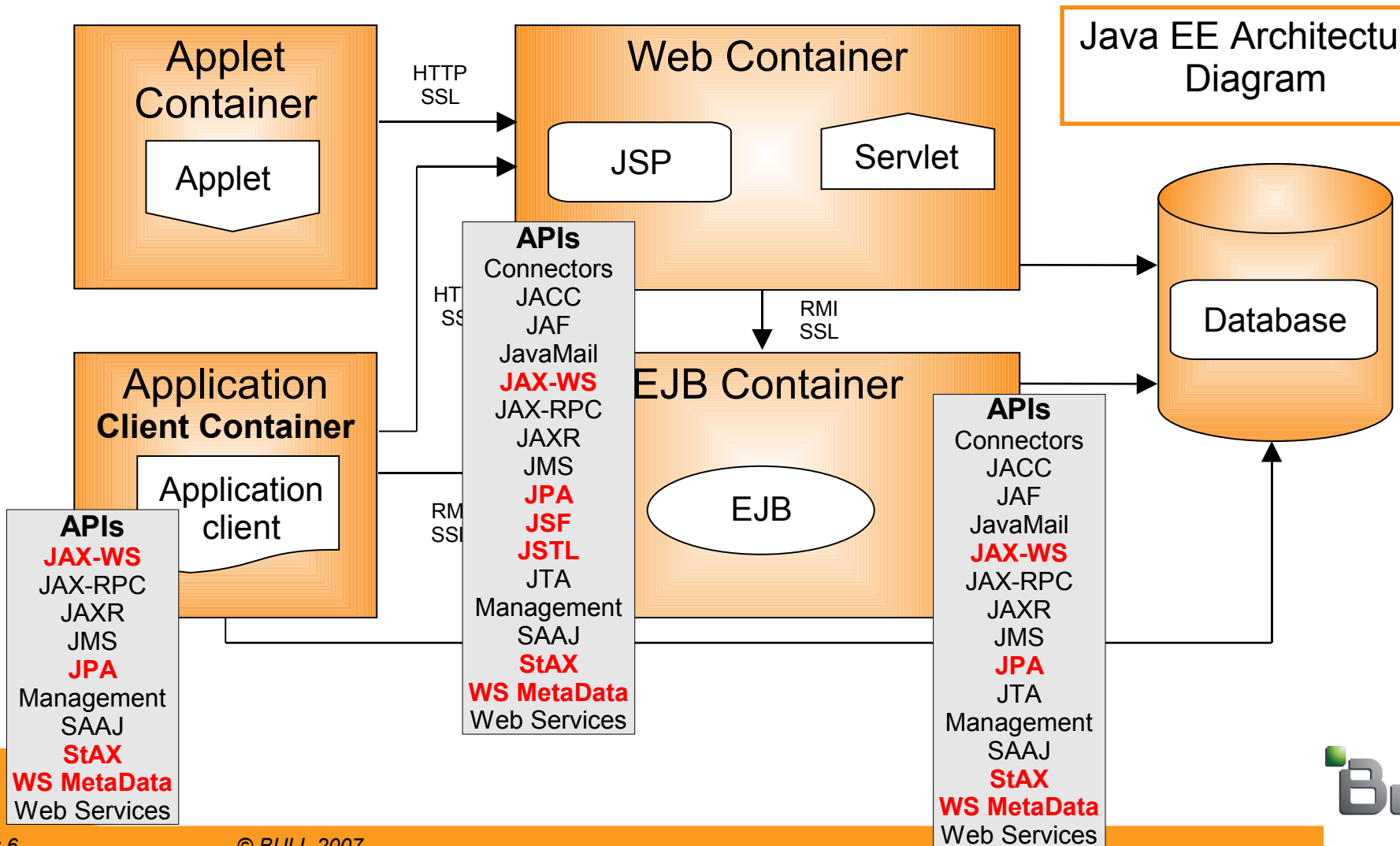
- JOnAS (LGPL)
- JBoss (LGPL)
- Geronimo (Apache)
- GlassFish (CDDL)

## ■ Commercial

- BEA WebLogic Server
- IBM Websphere Application Server
- Sun Java Application Server
- Oracle Application Server
- ...

# Architecture Java EE™

Java EE Architecture Diagram



# Définitions de quelques termes Java EE

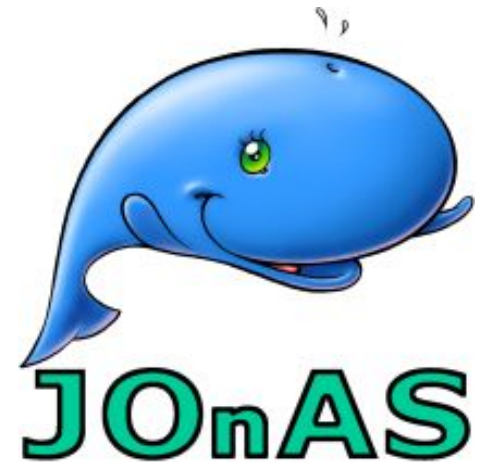
- ➔ Composants de présentation (pages HTML dynamiques; peuvent accéder aux ressources du serveur d'applications Java EE, par exemple les EJBs.):
  - **Servlet**: composant java générant des pages HTML
  - **JSP**: pages HTML intégrant du code Java.
- ➔ Composants métiers
  - **EJB**: Enterprise JavaBeans, composant java implantant la logique métier
    - **Session Bean**: attaché à un client, fourniture de méthodes métiers via son interface. Stateless ou Stateful (sans ou avec état).
    - **Message-driven Bean**: composant asynchrone, exécuté lors de la réception d'un message. Exemple : JMS (Java Message Service).
    - **Entities** : Objet de persistance. Plain Old Java Object (POJO) depuis la spécification EJB v3.0
- ➔ Conteneur
  - Structure d'accueil pour l'exécution des composants Java EE, s'interfaçant entre les composants et le serveur d'applications (et des services de celui-ci).

# Java EE : Modèle de développement

- Parties présentation et métier développées avec des composants Java
- Code utilisant le moins possible les domaines suivants :
  - transaction, persistance, sécurité
  - load balancing, pooling, distribution, ...
- Utilisation de ressources “logiques” via un environnement Java EE (accès aux EJB, DataSource, etc)
- => **Obtient des composants réutilisables et portables.**

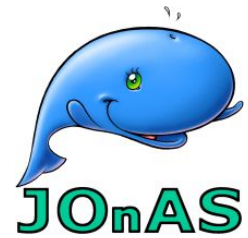


Serveur d'applications JOnAS



# Java Open Application Server :

## JOnAS (<http://jonas.objectweb.org>)



- ➔ Certifié J2EE 1.4 (Certification SUN) depuis février 2005
- ➔ Passage à l'échelle et haute disponibilité
  - Clustering (HTTP, RMI, DB), Failover (Réplication sessions HTTP)
  - Mécanismes d'optimisation (pools, cache, ...)
- ➔ Intégration en entreprise
  - Architecture Multi-tiers
  - Apache, LDAP, DBMSs, J2EE CA connectors, mainframes
  - Web Services
- ➔ Environnement de développement
  - Plug-ins Eclipse
- ➔ Administration
  - Console Web, commandes/scripts, API (JMX, JSR77)
- ➔ Offre de support (Bull)



# Historique JOnAS

## ■ Open Source depuis Juillet 1999

- Hébergé par le consortium international OW2
  - <http://www.ow2.org>
  - Initié par Bull, France Telecom et l'Inria



- Communauté d'utilisateurs et développeurs
    - > 200,000 JOnAS downloads (~ 5000 par mois)
    - ~ 500 inscrits sur la liste JOnAS
  - Applications opérationnelles
  - JOnAS est distribué sous licence LGPL (GNU Lesser General Public License)
    - >800 000 lignes de code OW2 (> 400 000 pour JOnAS)
- ## ■ Java EE 1.4 Certification Scholarship accordé par Sun



# JOnAS: Disponibilité et pré-requis

- Disponible sur <http://jonas.objectweb.org>
  - tgz ou exe (windows installer) packages:
    - JOnAS binaire ou source
    - JOnAS/Tomcat/Axis package
    - JOnAS/Jetty/Axis package
  - RPMs disponibles sur <http://www.jpackage.org>
- Pré-requis
  - JDK 1.4+ pour JOnAS 4 (Sun, IBM, BEA JVMs. OpenSource : GCJ, ...)
  - JDK 5+ pour JOnAS 5
  - Fonctionne avec le JDK 5 & 6
- Environnements supportés
  - Systèmes d'exploitations :
    - GNU/Linux, Solaris, AIX, HP-UX, Windows, etc.
  - BDD :
    - PostgreSQL, MySQL, Oracle, SQL Server, DB2, Access, Interbase, InstantDB, Sybase, Informix, HSQLDB, etc.

# JOnAS Admin console

The screenshot displays the JOnAS Administration console interface. On the left is a tree view of the domain structure, including Server JOnAS (jonas), Monitoring, Logging, Protocols, Services, Application Container, EJB Container, Web Container, Database, Resource, Transaction, Jms, Mail, Security, Deployment, Applications (EAR), EJB Modules (JAR), Web Modules (WAR), Resource Adapter Modules, Resources, Database (JDBC), JMS, Mail, Security, and MBeans.

The main content area shows the JOnAS Administration page for the AlarmRecord EJB. The breadcrumb path is: Domain (jonas) > Server JOnAS (jonas) > Services > EJB Container > alarm.jar. The page title is "JOnAS Administration" with a logo and a "Refresh" button. The current page is "AlarmRecord" under the "alarm.jar" EJB container.

Navigation tabs include: EJB Containers, alarm.jar, Statistics, EJBs, **AlarmRecord**, and Dependency. The "Action" section contains "Synchronize" and "Reduce instances" buttons.

The "EntityBean" section shows the following details:

<b>Name</b>	AlarmRecord
<b>File</b>	/D:/JONAS/JOnAS-4.1.4/work/apps/jonas/alarm_2004.10.01-16.33.49/alarm.jar

The "Naming and Classes" section shows the following details:

<b>EJB class</b>	org.objectweb.alarm.beans.AlarmRecordBean
<b>EJB JNDI Name</b>	alarmrecord
<b>EJB HomeClass</b>	org.objectweb.alarm.beans.AlarmRecordHome
<b>EJB RemoteClass</b>	org.objectweb.alarm.beans.AlarmRecord
<b>EJB LocalHomeClass</b>	
<b>EJB LocalClass</b>	

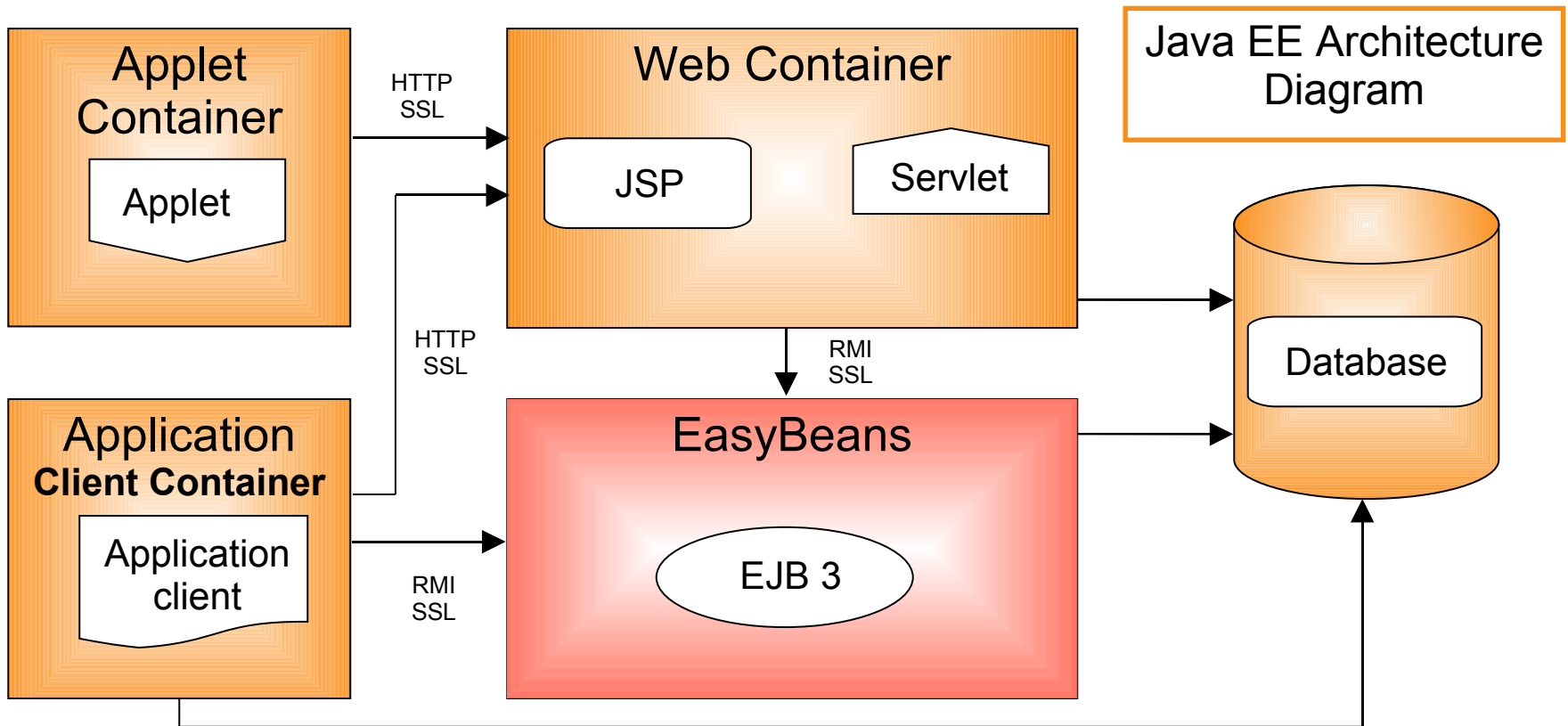
The "Entity Configuration" section shows the following details:

<b>Persistency</b>	Container-Managed Persistency
<b>Passivation Timeout</b>	0
<b>Lock Policy</b>	container-serialized
<b>Shared</b>	false
<b>Prefetch</b>	false
<b>Initial Pool size</b>	0
<b>Max number of instances</b>	0

The "Memory Management" section shows the following details:

<b>Current Pool size</b>	1
<b>Used in TX</b>	0
<b>Used outside TX</b>	0
<b>Unused but ready</b>	3

# JOnAS EJB3 : EasyBeans



# EasyBeans

- <http://www.easybeans.net> & <http://www.easybeans.org>
- Projet démarré en Décembre 2005
- Conteneur Léger et modulaire
- Peut être intégré aux serveurs JOnAS, Jetty & Tomcat
- Mode standalone
- Mode OSGi (Apache Felix, Eclipse Equinox, Knopflerfish)
- Pré-requis
  - JDK 5+
- Fonctionnalités : archives “exploded”, rechargement de classes, facilité de déploiement, facilité dans le lancement des clients, etc.

# Pourquoi choisir EJB 3.0 / EasyBeans

- Utilisation de standards Java EE
  - Spring = framework non standard
- EasyBeans fonctionne dans plusieurs serveurs
  - JOnAS, Tomcat, Jetty, ...
- Utilisation de la manipulation de bytecode (avec la bibliothèque ASM d'OW2)
  - Gagne en rapidité
  - Augmente performance
- Peut être embarqué facilement
- Disponible en tant que bundles OSGi

# En savoir plus ?

## EasyBeans

Open Source & Lightweight EJB3 Container

[Project](#) [Demos](#) [Documentation](#) [Downloads](#) [Support](#) [News](#) [Developers](#) [Wiki](#)

### :: What is EasyBeans ?

EasyBeans is an open-source [Enterprise Java Beans \(EJB\)](#) container hosted by the [OW2 consortium](#). The License used by EasyBeans is the [LGPL](#).

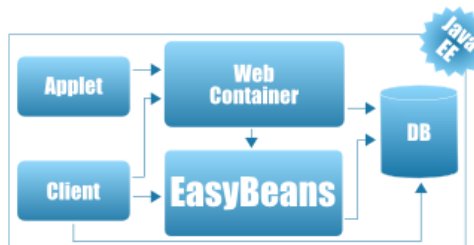
EasyBeans main goal is to ease the development of Enterprise Java Beans. It uses some new architecture design like the bytecode injection (with [ASM ObjectWeb tool](#)), [IoC](#), [POJO](#) and can be embedded in [OSGi](#) bundles or other frameworks ([Spring](#), [Eclipse](#) plugins, etc.).

It aims to provide an EJB3 container as specified in the [Java Platform Enterprise Edition \(Java EE\)](#) in its fifth version. It means that Session beans (Stateless or Stateful), Message Driven Beans (MDB) are available on EasyBeans.

The new persistence layer used by EJB 3.0 is now called Java Persistence API (or JPA). It replaces the CMP (Container Managed Persistence) model used by EJB 2.x. The default persistence provider used in EasyBeans is Hibernate Entity Manager or Apache OpenJPA but other JPA providers have been tested like for example Oracle TopLink Essentials.

### Recent News

**10.17.2007** :: 1.0 RC 1 Released  
**08.29.2007** :: EasyBeans On OSGi  
**05.16.2007** :: 1.0 Milestone 6 Released  
[Read More >>](#)



### :: Why EasyBeans ?

For a long time, EJB (Enterprise Java Beans) has been considered as a complex technology. Many developers were not using EJBs when developing Java EE applications and then use partials J2EE servers like [Apache Tomcat](#). With the new Java EE 5 version, and EJB 3.0 which is part of this specification, one of the focus was on EoD (Ease of Development) So, Ease of Development was also the main item used to design the EJB3 container of [JOnAS application server](#). All is done in the JOnAS EJB3 container to develop EJBs in an easy way.

Some people said that the EJB container in JOnAS was too tied to the application server. An answer to these people was done by extracting the EJB container of the JOnAS core and making it available as a lightweight container. Also, this allows to use this JOnAS EJB3

container in other application servers, in standalone mode, bundle it for OSGi platform, in [JUnit/TestNG](#), etc. This is why JOnAS EJB3 container was named EasyBeans and has its own web site.

### :: Link with JOnAS J2EE application server.



JOnAS 4.x is a J2EE 1.4 certified application server and it runs with the JDK 1.4. EasyBeans and EJB 3.0, as part of Java EE 5, have features requiring a new JDK version, the JDK 5.0.

EasyBeans is then available for JOnAS 4.x with a resource adapter, you only have to use a JDK 5.0. But it's easy to disable EasyBeans (if developers want to switch from JDK 5.0 to JDK 1.4) by removing this resource adapter (.rar file).

The next major version of JOnAS (JOnAS 5.x / Java EE 5) will provide EasyBeans by default for its EJB3 container.

As EasyBeans and JOnAS have two separated codebase, the projects can have a different lifecycle and then EJB3 container can be upgraded without upgrading the whole platform.





EJB 3.0 (sous ensemble de Java EE 5)

# EJB 3.0 : Description

- Inclus dans Java EE 5
- JSR 220
  - <http://www.jcp.org/en/jsr/detail?id=220>
- 2 parties
  - Core (Session Beans, MDB)
  - Persistence (Entities, Persistence provider)
- EJB 3.0 core :
  - Branchement du fournisseur de persistance
  - Différentes implantations :
    - Hibernate Entity Manager, Apache OpenJPA, Oracle TopLink, ,...

# Pourquoi les EJB 3.0 ?

- EJB 2.x : Trop complexe
- Des projets émergent pour simplifier le développement :
  - XDoclet : utilisation de tags javadoc en remplacement des fichiers XML et des interfaces.
    - <http://xdoclet.sourceforge.net>
  - Hibernate : Simplification du modèle de persistance.
    - <http://www.hibernate.org>
- Faciliter le développement:
  - Ease of Development (EoD)
  - Utilisation de métadonnées : les annotations
- Agrandir la communauté de développeurs Java EE

# EJB2 vs EJB3 : Simplicité du développement

- Les descripteurs de déploiement deviennent optionnels.
  - Utilisation des annotations (métadonnées)
- Simplification de la persistance.
  - CMP des EJB 2.0 remplacée par JPA (proche du modèle Hibernate/JDO)
- Un ensemble de valeurs par défaut (TX par défaut Required)
- Réduction de l'utilisation de certaines exceptions
  - exemple : RemoteException
- Interface Home (pour le cycle de vie) n'est plus requise
- Les interfaces « callback » ne sont plus obligatoires. Plus besoin d'implanter `javax.ejb.SessionBean`
- Autorisation de l'héritage
- Amélioration du langage EJB-QL : requêtes SQL natives

# EJB3 : Les nouveautés

- Annotations / Métadonnées
- Intercepteurs
  - Chaque méthode métier peut être interceptée par le développeur
- Intercepteurs pour la notification du cycle de vie
  - Définis par des annotations / métadonnées
  - Remplace les méthodes `ejbXXX()` des EJB 2.1 comme `ejbActivate()`
- Injection de dépendance
  - Annotations utilisées pour l'injection : Ressources, EJBs, ...
- Persistance :
  - POJO (Plain Old Java Object) like
  - Gestionnaire de persistance
  - Possibilité d'être notifié à différents moments du cycle de vie

# Définition de Beans EJB 3.0 [1/2]

**@Remote**

```
public interface HelloWorld extends Remote {  
    String hello() throws RemoteException;  
}
```

Interface

**@Stateful**

**@Stateless**

```
public class HelloWorldBean implements HelloWorld {  
    public String hello() throws RemoteException {  
        return "Hello World !";  
    }  
}
```

Implementation

# Définition de Beans EJB 3.0 [2/2]

```
@MessageDriven(activateConfig={
    @ActivationConfigProperty(propertyName="destinationType", propertyValue="javax.jms.Queue"),
    @ActivationConfigProperty(propertyName="destination", propertyValue="jms/StockQueue")
})
public class MyMDB implements MessageListener {
    public void onMessage(Message message) {
        ...
    }
}
```

Message Driven Bean

# Métadonnées EJB 3.0

Default = REQUIRED

```
@Stateless public class MyBean implements MyItf {  
    @TransactionAttribute(REQUIRES_NEW)  
    public void someMethod() { ... }  
}
```

Transaction

```
@Stateless public class MyBean implements MyItf {  
    @RolesAllowed("EasyBeans")  
    public void someMethod () {...}  
}
```

Securité

# Métadonnées pour la persistance

```
@Entity
@Table(name="EMPLOYEES")
public class Employee {
```

```
    private int id;
    private String name;
```

```
    @Id
    public int getId() { return id; }
    public void setId(final int id) { this.id = id; }
```

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

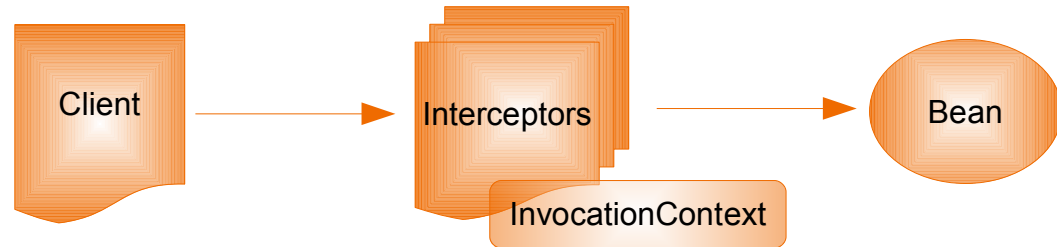
```
}
```

Primary  
Key

Entity

# Intercepteurs

- Intercepte les appels des méthodes métiers
  - Session Beans + Message Driven Beans
  - Peut traiter les exceptions remontées par les méthodes
  - Peut changer les valeurs des paramètres et de retour d'une méthode



- Intercepteurs

- Définition en utilisant l'annotation `@AroundInvoke`
- Référencés par l'annotation `@Interceptors`
  - Référence depuis une classe ou une méthode
- Intercepteurs par défaut (définis dans les descripteurs de déploiement)
- Possibilité d'exclure certains intercepteurs
  - Intercepteurs par défaut
  - Intercepteurs définis au niveau de la classe

# InvocationContext

```
public interface InvocationContext {
    Object getTarget();
    Method getMethod();
    Object[] getParameters();
    setParameters(Object[]);
    Map getContextData(); // partagé par tous les intercepteurs
    Object proceed() throws Exception; // appel du prochain intercepteur
}
```

## InvocationContext interface

### @AroundInvoke

```
public Object trace(InvocationContext invocationContext) throws Exception {
    long tStart = System.currentTimeMillis();
    try { return invocationContext.proceed();
    } finally {
        long elapsed = System.currentTimeMillis() - tStart;
        System.out.println(inv.getMethod() + " took " + elapsed + " ms.");
    }
}
```

## Interceptor : tracing



# Intercepteurs utilisateurs

- Exemple d'un intercepteur pour tracer les appels.
- Fonctionnement de l'interception de la méthode helloWorld()

1/ **StatelessBean.helloWorld()**

2/ StatelessBean.EasyBeansInvoca...helloWorldAROUNDINVOKE.proceed()

3/ **StatelessBean.trace()**

4/ StatelessBean.EasyBeansInvoca...helloWorldAROUNDINVOKE.proceed()

5/ **StatelessBean.original\$EasyBeans\$helloWorld()**

- (1) Méthode d'origine a été renommée
- (2) Appel de la méthode proceed() qui réalise l'appel sur le code de l'intercepteur (3)
- (4) Appel de la méthode proceed() qui réalise l'appel sur le code de la méthode d'origine (5)

# Intercepteurs / Notification du cycle de vie

- Reception de notification lors d'événements du cycle de vie
- Définition dans la classe du bean

```
@PreDestroy  
public void myPreDestroy() { ...}
```

- Ou dans une classe séparée

```
@PostConstruct  
public void myPostConstruct(InvocationContext ctx) { ...}
```

- Même conception que les intercepteurs de méthodes métiers

# Notifications pour la partie persistance

Annotation	SLSB	SFSB	MDB
@PostConstruct ( <i>ejbCreate</i> )	X	X	X
@PreDestroy ( <i>ejbRemove</i> )	X	X	X
@PostActivate ( <i>ejbActivate</i> )		X	
@PrePassivate ( <i>ejbPassivate</i> )		X	

Session & Message Driven Beans

@PostPersist  
@PreRemove  
@PostRemove  
@PreUpdate  
@PostUpdate  
@PostLoad

Entities

# Injection de dépendance [1/3]

## ■ Récupération d'une connexion vers une BDD

- Avec les EJB 2.X/J2EE 1.4 :

```
Context ictx = new InitialContext();
DataSource myDS = null;
try {
    myDS = (DataSource) ictx.lookup("java:comp/env/jdbc/myDS");
} catch (NamingException e) { ... }
Connection connection = myDS.getConnection();
```

## ■ Avec l'injection de dépendance / EJB 3.0 :

```
@Resource(name = "jdbc/myDS",.....)
private DataSource myDS;
```

Request Dependency Injection

```
private void method() {
    Connection connection = myDS.getConnection();
}
```

# Injection de dépendance [2/3]

## ■ Récupération d'un EJB

- Avec les EJB 2.X/J2EE 1.4 :

```
Context ictx = new InitialContext();
EJBHome ejbHome = null;
try {
    Object o = ictx.lookup("java:comp/env/ejb/myEJB");
    ejbHome = (EJBHome) PortableRemoteObject.narrow(o, EJBHome.class);
} catch (NamingException e) { ... }
InterfaceEJB ejb = ejbHome.create();
ejb.helloWorld();
```

## ■ Avec l'injection de dépendance / EJB 3.0 :

```
@EJB
private InterfaceEJB ejb;
```

Request Dependency Injection

```
private void method() {
    ejb.helloWorld();
}
```

No more Home, invoking directly  
the business interface



# Injection de dépendance [3/3]

```
@Resource(name="myDS")  
public DataSource myDS;
```

## Instance Variables

```
@Resource(name="myDS")  
public void setDataSource(DataSource myDS) {  
    this.ds = myDS;  
}
```

## Setter Injection

# Modèle de persistance [1/2]

- Entities : Création avec « new() »

```
Employee employee = new Employee();
```

- Entities : persistance avec le gestionnaire d'Entity

```
entityManager.persist(employee);
```

- EntityManager peut être injecté

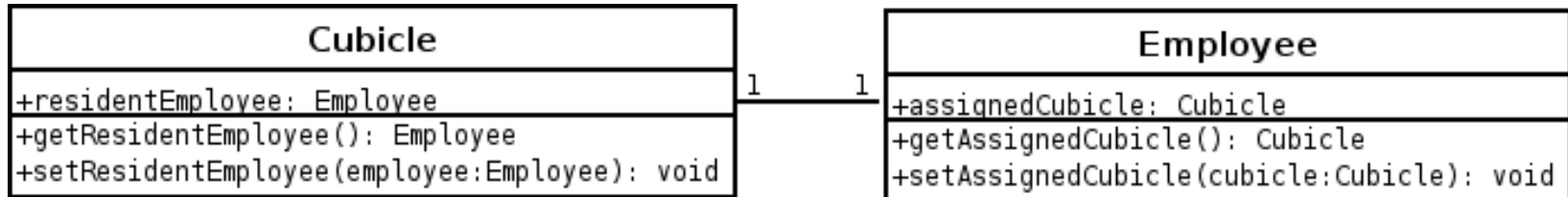
```
@PersistenceContext (unitName="unit1")  
private EntityManager entityManager;
```

- EJB-QL queries / Entity Manager

```
entityManager.createQuery("Select .....");
```



# Relations / Entités



- Employee & Cubicle
- Un Employé possède un bureau, un bureau possède un seul employé.
- Relation OneToOne dans les deux cas
- => Deux classes POJOs
  - Cubicle class
  - Employee class

# Relations entre Entités

```
@Entity
public class Employee {
    private Cubicle assignedCubicle;
    @OneToOne
    public Cubicle getAssignedCubicle() { return assignedCubicle; }
    public void setAssignedCubicle(Cubicle cubicle) { this.assignedCubicle = cubicle; }
    ...
}
```

```
@Entity
public class Cubicle {
    private Employee residentEmployee;
    @OneToOne(mappedBy="assignedCubicle")
    public Employee getResidentEmployee() { return residentEmployee; }
    public void setResidentEmployee(Employee employee) {
        this.residentEmployee = employee;
    }
    ...
}
```

# Annotations EJB 3.0

- <http://www.easybeans.net/xwiki/bin/download/Main/Documentation/ejb3-annotations.pdf>

## EasyBeans

### EJB3 Annotations (Core)

#### javax.annotation

[M] @PostConstruct  
[M] @PreDestroy  
[F, M, T] @Resource(authenticationType={CONTAINER, APPLICATION}, description=String, mappedName=String, name=String, shareable=boolean, type=Class)  
[T] @Resources(Resource[])

#### javax.annotation.security

[T] @DeclareRoles(String[])  
[M] @DenyAll  
[M, T] @PermitAll  
[M, T] @RolesAllowed(String[])  
[T] @RunAs(String)

#### javax.ejb

[] @ActivationConfigProperty(propertyName=String, propertyValue=String)  
[T] @ApplicationException(rollback=boolean)  
[F, M, T] @EJB(beanInterface=Class, beanName=String, description=String, mappedName=String, name=String)  
[T] @EJBs(EJB[])  
[M] @Init(String)  
[T] @Local(Class[])  
[T] @LocalHome(Class)  
[T] @MessageDriven(activationConfig=ActivationConfigProperty[], description=String, mappedName=String, messageListenerInterface=Class, name=String)  
[M] @PostActivate  
[M] @PostPassivate  
[T] @Remote(Class[])  
[T] @RemoteHome(Class)  
[M] @Remove(retainIfException=boolean)  
[T] @Stateful(description=String, mappedName=String, name=String)  
[T] @Stateless(description=String, mappedName=String, name=String)  
[M] @Timeout  
[M, T] @TransactionAttribute({MANDATORY, REQUIRED, REQUIRES\_NEW, SUPPORTS, NOT\_SUPPORTED, NEVER})  
[T] @TransactionManagement({CONTAINER, BEAN})

#### javax.interceptor

[M] @AroundInvoke  
[M] @ExcludeClassInterceptors  
[M, T] @ExcludeDefaultInterceptors  
[M, T] @Interceptors(Class[])

### EJB3 Annotations (Persistence)

#### javax.persistence

[F, M, T] @AssociationOverrides(joinColumns=JoinColumn[], name=String)  
[F, M, T] @AssociationOverrides(AssociationOverride[])  
[F, M, T] @AttributeOverride(column=Column, name=String)  
[F, M, T] @AttributeOverrides(AttributeOverride[])  
[F, M] @Basic(fetch={LAZY, EAGER}, optional=boolean)  
[F, M] @Column(columnDefinition=String, insertable=boolean, length=int, name=String, nullable=boolean, precision=int, scale=int, table=String, unique=boolean, updatable=boolean)  
[] @ColumnResult(name=String)  
[T] @DiscriminatorColumn(columnDefinition=String, discriminatorType={STRING, CHAR, INTEGER}, length=int, name=String)  
[T] @DiscriminatorValue(String)  
[T] @Embeddable  
[F, M] @Embedded  
[F, M] @EmbeddedId  
[T] @Entity(name=String)  
[T] @EntityListeners(Class[])  
[] @EntityResult(discriminatorColumn=String, entityClass=Class, fields=FieldResult[])  
[F, M] @Enumerated({ORDINAL, STRING})  
[T] @ExcludeDefaultListeners  
[T] @ExcludeSuperclassListeners  
[] @FieldResult(column=String, name=String)  
[F, M] @GeneratedValue(generator=String, strategy={TABLE, SEQUENCE, IDENTITY, AUTO})  
[F, M] @Id  
[T] @IdClass(Class)  
[T] @Inheritance(strategy={SINGLE\_TABLE, JOINED, TABLE\_PER\_CLASS})  
[F, M] @JoinColumn(columnDefinition=String, insertable=boolean, name=String, nullable=boolean, referencedColumnName=String, table=String, unique=boolean, updatable=boolean)  
[F, M] @JoinColumns(JoinColumn[])  
[F, M] @JoinTable(catalog=String, inverseJoinColumns=JoinColumn[], joinColumns=JoinColumn[], name=String, schema=String, uniqueConstraints=UniqueConstraint[])  
[F, M] @Lob  
[F, M] @ManyToMany(cascade={ALL, PERSIST, MERGE, REMOVE, REFRESH}, fetch={LAZY, EAGER}, mappedBy=String, targetEntity=Class)  
[F, M] @ManyToOne(cascade={ALL, PERSIST, MERGE, REMOVE, REFRESH}, fetch={LAZY, EAGER}, optional=boolean, targetEntity=Class)  
[F, M] @MapKey(name=String)  
[T] @MappedSuperclass  
[T] @NamedNativeQueries(NamedNativeQuery[])  
[T] @NamedNativeQuery(hints=QueryHint[], name=String, query=String, resultClass=Class, resultSetMapping=String)  
[T] @NamedQueries(NamedQuery[])  
[T] @NamedQuery(hints=QueryHint[], name=String, query=String)  
[F, M] @OneToMany(cascade={ALL, PERSIST, MERGE, REMOVE, REFRESH}, fetch={LAZY, EAGER}, mappedBy=String, targetEntity=Class)  
[F, M] @OneToOne(cascade={ALL, PERSIST, MERGE, REMOVE, REFRESH}, fetch={LAZY, EAGER}, mappedBy=String, optional=boolean, targetEntity=Class)  
[F, M, T] @PersistenceContext(name=String, properties=PersistenceProperty[], type={TRANSACTION, EXTENDED}, unitName=String)  
[F, M, T] @PersistenceContext(name=String, properties=PersistenceProperty[], type={TRANSACTION, EXTENDED}, unitName=String)  
[T] @PersistenceContexts(PersistenceContext[])  
[] @PersistenceProperty(name=String, String)  
[F, M, T] @PersistenceUnit(name=String, unitName=String)  
[T] @PersistenceUnits(PersistenceUnit[])  
[M] @PostLoad  
[M] @PostPersist  
[M] @PostRemove  
[M] @PostUpdate  
[M] @PrePersist  
[M] @PreRemove  
[M] @PreUpdate  
[F, M, T] @PrimaryKeyJoinColumn(columnDefinition=String, name=String, referencedColumnName=String)  
[F, M, T] @PrimaryKeyJoinColumns(PrimaryKeyJoinColumn[])  
[] @QueryHint(name=String, String)  
[T] @SecondaryTable(catalog=String, name=String, pkJoinColumns=PrimaryKeyJoinColumn[], schema=String, uniqueConstraints=UniqueConstraint[])  
[T] @SecondaryTables(SecondaryTable[])  
[F, M, T] @SequenceGenerator(allocationSize=int, initialValue=int, name=String, sequenceName=String)  
[T] @SqlResultSetMapping(columns=ColumnResult[], entities=EntityResult[], name=String)  
[T] @SqlResultSetMappings(SqlResultSetMapping[])  
[T] @Table(catalog=String, name=String, schema=String, uniqueConstraints=UniqueConstraint[])  
[F, M, T] @TableGenerator(allocationSize=int, catalog=String, initialValue=int, name=String, pkColumnName=String, pkColumnValue=String, schema=String, table=String, uniqueConstraints=UniqueConstraint[], valueColumnName=String)  
[F, M] @Temporal({DATE, TIME, TIMESTAMP})  
[F, M] @Transient  
[] @UniqueConstraint(columnNames=String[])  
[F, M] @Version

F: Field  
M: Method  
T: Type (class)  
Pkg: Package  
Par: Parameter

Provided by EasyBeans (<http://www.easybeans.net>)  
Produced automatically by analyzing classes.



# EJB 3.1 : Le futur

## ■ JSR 318

- <http://jcp.org/en/jsr/detail?id=318>
- Java EE 6
- Q4 2008 (final release)

## ■ Interface locale optionnelle

- Utilisation directe de la classe du bean
- Méthodes publiques = méthodes métiers

## ■ Bean singleton

- Start/stop. Initialisation (lazy/eager)

## ■ Évolution du service timer

- Syntaxe proche de l'outil cron (unix)

## ■ Profiles (Serveurs Java EE lite)

- Approche conteneurs légers mais standards

## ■ ...



Développer des applications EJB 3.0

# Pré-Requis

- JOnAS:
  - JOnAS 4.8 + EasyBeans
    - JOnAS 4.8.6 = dernière version stable
  - JOnAS 5.0M1+
- JDK 5.0
- Outils :
  - Ant : <http://ant.apache.org>
  - Maven 2 : <http://maven.apache.org>

# JOnAS 4.8.6 + EasyBeans

- EasyBeans est déjà intégré dans JOnAS 5.0
- Pour JOnAS 4.8.6 :
  - Définir la variable JONAS\_ROOT si ce n'est pas déjà effectué
  - Télécharger EasyBeans sur le site [www.easybeans.net](http://www.easybeans.net)
  - Décompresser l'archive, par exemple la version avec Hibernate:
    - `unzip ow2-easybeans-jonas-1.0.0.RC1-hibernate.zip`
  - Copier le fichier .rar de l'archive sous JONAS\_ROOT/rars/autoload
    - `cp ow2-easybeans-jonas-1.0.0.RC1/easybeans-rar_for_jonas-hibernate-1.0.0.RC1.rar JONAS_4_8_6/rars/autoload/`

# Applications EJB 3.0

- Bean stateless
  - Intercepteurs
  - Debugging
- Bean stateful
- Bean avec persistance
- MDB
- Migration Hibernate -> JPA



Bean sans état : Stateless

# Exemple d'un bean stateless : une calculatrice

- Comment procéder ?
- Définir une interface
  - Une méthode pour additionner deux entiers
  - Une méthode pour diviser deux entiers

```
package org.easybeans.exemple.calculatrice;

/**
 * Définition de l'interface de la calculatrice
 * @author Florent BENOIT
 */
public interface CalculatriceInterface {

    /**
     * Réalise la somme des entiers a et b et retourne la somme obtenue.
     * @param a le premier entier
     * @param b le second entier
     * @return la somme des entiers
     */
    int addition(final int a, final int b);

    /**
     * Réalise la division des entiers a et b et retourne la division obtenue.
     * @param a le premier entier
     * @param b le second entier
     * @return la division des 2 entiers
     */
    int division(final int a, final int b);

}
```

# Calculatrice: Ajout du code métier

- Le code métier associé aux méthodes définies par l'interface doit être ajouté

```
package org.easybeans.exemple.calculatrice;

/**
 * Code métier de la calculatrice.
 * @author Florent BENOIT
 */
public class Calculatrice implements CalculatriceInterface {

    /**
     * Réalise la somme des entiers a et b et retourne la somme obtenue.
     * @param a le premier entier
     * @param b le second entier
     * @return la somme des entiers
     */
    public int addition(final int a, final int b) {
        return a + b;
    }

    /**
     * Réalise la division des entiers a et b et retourne la division obtenue.
     * @param a le premier entier
     * @param b le second entier
     * @return la division des 2 entiers
     */
    public int division(final int a, final int b) {
        return a / b;
    }
}
```

# Transformer Interface + Classe en EJB 3.0 [1/2]

- On souhaite que ce bean soit accessible à distance
- => Ajout de l'annotation @Remote sur l'interface

```
import javax.ejb.Remote;
```

```
/**  
 * Définition de l'interface de la calculatrice  
 * @author Florent BENOIT  
 */
```

```
@Remote
```

```
public interface CalculatriceInterface {
```

# Transformer Interface + Classe en EJB 3.0 [2/2]

- Ce Bean est un bean sans état : aucune conservation d'attributs => Stateless Session Bean
- Ajout de l'annotation `@Stateless` sur la classe du bean

```
import javax.ejb.Stateless;
```

```
/**  
 * Code métier de la calculatrice.  
 * @author Florent BENOIT  
 */
```

```
@Stateless
```

```
public class Calculatrice implements CalculatriceInterface {
```

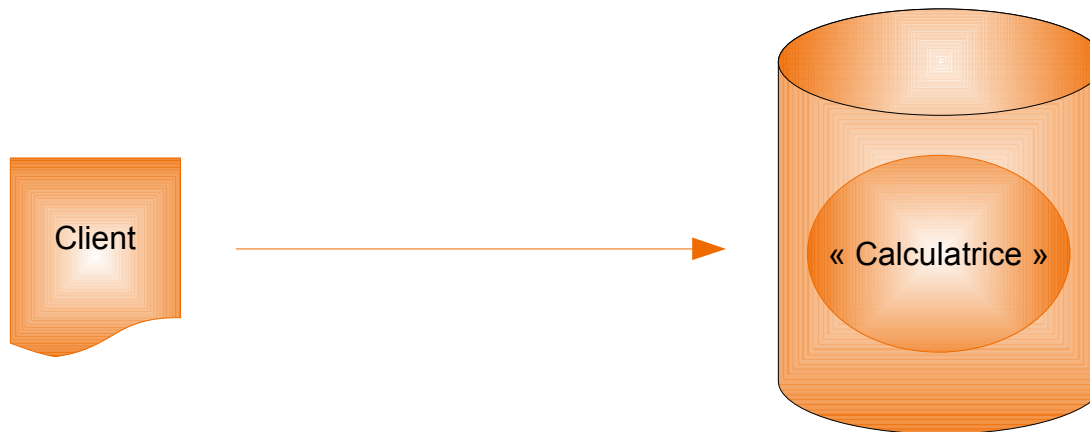
# Client de la calculatrice

## ■ Principe :

- Accéder au bean
- Appel d'une méthode

## ■ Pour simplifier, nous nommons le bean « Calculatrice » dans le registre RMI (Son nom JNDI)

```
@Stateless(mappedName="Calculatrice")  
public class Calculatrice implements CalculatriceInterface {
```



# Code du client

## ■ Client léger :

- Création d'une servlet
- Appel de la servlet depuis un navigateur WEB

```
/**
 * Client qui va faire appel au bean calculatrice
 * @author Florent Benoit
 */
public class Client extends HttpServlet {

    /**
     * Obtient une instance du bean calculatrice.
     * @return une instance du bean
     * @throws Exception si le bean n'est pas trouvé.
     */
    private CalculatriceInterface getCalculatrice() throws Exception {
        Context initialContext = new InitialContext();
        Object o = initialContext.lookup("Calculatrice");

        if (o instanceof CalculatriceInterface) {
            CalculatriceInterface calculatrice = (CalculatriceInterface) o;
            return calculatrice;
        }
        throw new Exception("Object calculatrice invalide");
    }
}
```

# Code du client (suite)

## ■ Réalisation d'une addition :

```
/**
 * Affiche somme de val1 et val2.
 * @param out pour écrire le code HTML
 * @param val1 premier entier
 * @param val2 second entier
 */
private void afficheAddition(final PrintWriter out, final int val1, final int val2) {
    out.println("<br> La somme de '" + val1 + "' et '" + val2 + "' = ");
    try {
        int sum = getCalculatrice().addition(val1, val2);
        out.println(sum);
    } catch (Exception e) {
        afficheException(out, e);
    }
}
```

## ■ Réalisation d'une division :

```
/**
 * Affiche division de val1 et val2.
 * @param out pour écrire le code HTML
 * @param val1 premier entier
 * @param val2 second entier
 */
private void afficheDivision(final PrintWriter out, final int val1, final int val2) {
    out.println("<br> La division de '" + val1 + "' par '" + val2 + "' = ");
    try {
        int div = getCalculatrice().division(val1, val2);
        out.println(div);
    } catch (Exception e) {
        afficheException(out, e);
    }
}
```

# Déploiement des applications

- Comment réaliser le déploiement de l'EJB et du client ?
- Réalisation de deux archives
  - Une archive pour l'EJB : un fichier .jar
  - Une archive pour le client : un fichier .war (Web Archive)
- EJB contient l'interface et le code métier du bean
- Client contient l'interface du bean et le code du client

# Archive EJB

- Descripteurs de déploiement optionnels
  - Fichier jar contient uniquement les classes/interfaces du bean + le fichier Manifest par défaut

```
0 Sun Nov 01 00:00:00 CET 2007 META-INF/  
106 Sun Nov 01 00:00:00 CET 2007 META-INF/MANIFEST.MF  
0 Sun Nov 01 00:00:00 CET 2007 org/  
0 Sun Nov 01 00:00:00 CET 2007 org/easybeans/  
0 Sun Nov 01 00:00:00 CET 2007 org/easybeans/exemple/  
0 Sun Nov 01 00:00:00 CET 2007 org/easybeans/exemple/calculatrice/  
694 Sun Nov 01 00:00:00 CET 2007 org/easybeans/exemple/calculatrice/Calculatrice.class  
265 Sun Nov 01 00:00:00 CET 2007 org/easybeans/exemple/calculatrice/CalculatriceInterface.class
```

- Archive à déposer dans le répertoire JONAS\_BASE/easybeans-deploy/
- EasyBeans détecte automatiquement les nouvelles archives et se charge de réaliser le déploiement

# Archive client léger : WAR

- Contient les classes du client (servlet) + interface du bean + un fichier XML décrivant la liste des servlets
- Classes sous WEB-INF/classes
- Descripteur de déploiement dans le rép WEB-INF/

```
0 Sun Nov 01 00:00:00 CET 2007 META-INF/
106 Sun Nov 01 00:00:00 CET 2007 META-INF/MANIFEST.MF
0 Sun Nov 01 00:00:00 CET 2007 WEB-INF/
601 Sun Nov 01 00:00:00 CET 2007 WEB-INF/web.xml
0 Sun Nov 01 00:00:00 CET 2007 WEB-INF/classes/
0 Sun Nov 01 00:00:00 CET 2007 WEB-INF/classes/org/
0 Sun Nov 01 00:00:00 CET 2007 WEB-INF/classes/org/easybeans/
0 Sun Nov 01 00:00:00 CET 2007 WEB-INF/classes/org/easybeans/exemple/
0 Sun Nov 01 00:00:00 CET 2007 WEB-INF/classes/org/easybeans/exemple/calculatrice/
265 Sun Nov 01 00:00:00 CET 2007 WEB-INF/classes/org/easybeans/exemple/calculatrice/CalculatriceInterface.class
4401 Sun Nov 01 00:00:00 CET 2007 WEB-INF/classes/org/easybeans/exemple/calculatrice/ClientLeger.class
```

# Descripteur de déploiement

## ■ WEB-INF/web.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

```
<web-app  
  xmlns="http://java.sun.com/xml/ns/j2ee"  
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"  
  version="2.4">
```

```
  <!-- Class of the client -->
```

```
  <servlet>  
    <servlet-name>Calculatrice</servlet-name>  
    <servlet-class>  
      org.easybeans.exemple.calculatrice.ClientLeger  
    </servlet-class>  
  </servlet>
```

```
  <servlet-mapping>  
    <servlet-name>Calculatrice</servlet-name>  
    <url-pattern>/</url-pattern>  
  </servlet-mapping>
```

```
</web-app>
```

■ <http://localhost:9000/<nom du fichier war>/>



Démo



Ajout d'intercepteurs

# Intercepteur

```
public interface InvocationContext {  
    Object getTarget();  
    Method getMethod();  
    Object[] getParameters();  
    setParameters(Object[]);  
    Map getContextData(); // partagé par tous les intercepteurs  
    Object proceed() throws Exception; // appel du prochain intercepteur  
}
```

## ■ Exemple : En

## ■ Algorithme :

- Récupérer les paramètres de la méthode « division »
- Si le deuxième paramètre est « 0 » alors exception

@AroundInvoke

```
private Object verifieDivisionParZero(final InvocationContext invocationContext) throws Exception {  
    // Méthode  
    Method m = invocationContext.getMethod();  
    System.out.println("Vérification des paramètres de la méthode " + m);  
  
    // Paramètres  
    Object[] parametres = invocationContext.getParameters();  
    // Verifie qu'il y a bien deux paramètres  
    if (parametres == null || parametres.length < 2) {  
        throw new IllegalStateException("Intercepteur invalide");  
    }  
  
    // Recupere valeur deuxieme argument  
    Integer diviseur = (Integer) parametres[1];  
  
    // Si 0, alors erreur !  
    if (diviseur.equals(0)) {  
        throw new IllegalArgumentException("On ne peut pas diviser par zéro");  
    }  
  
    // Sinon, on continue l'appel  
    return invocationContext.proceed();  
}
```

# Intercepteur méthode métier / Division par zéro

```
package org.easybeans.exemple.calculatrice;

import java.lang.reflect.Method;
import javax.interceptor.AroundInvoke;
import javax.interceptor.InvocationContext;

/**
 * Classe contenant un intercepteur
 * @author Florent BENOIT
 */
public class CalculatriceIntercepteur {
    /**
     * Intercepteur permettant d'empêcher la division par 0.
     * @param invocationContext contient des infos comme les paramètres de la méthode
     * @return résultat de la méthode
     * @throws Exception s'il y a échec
     */
    @AroundInvoke
    public Object verifieDivisionParZero(final InvocationContext invocationContext) throws Exception {
        // Méthode
        Method m = invocationContext.getMethod();
        System.out.println("Vérification des paramètres de la méthode " + m);
        // Paramètres
        Object[] parametres = invocationContext.getParameters();
        // Verifie qu'il y a bien deux paramètres
        if (parametres == null || parametres.length < 2) {
            throw new IllegalStateException("Intercepteur invalide");
        }
        // Recupere valeur deuxieme argument
        Integer diviseur = (Integer) parametres[1];
        // Si 0, alors erreur !
        if (diviseur.equals(0)) {
            throw new IllegalArgumentException("On ne peut pas diviser par zéro");
        }
        // Sinon, on continue l'appel
        return invocationContext.proceed();
    }
}
```

# Appliquer l'intercepteur sur la méthode

- Comment dire que l'intercepteur ne s'applique sur la méthode division ?

```
/**
 * Réalise la division des entiers a et b et retourne la division obtenue.
 * @param a le premier entier
 * @param b le second entier
 * @return la division des 2 entiers
 */
@Interceptors(CalculatriceIntercepteur.class)
public int division(final int a, final int b) {
    return a / b;
}
```

# Intercepteurs / Cycle de vie

- Être notifié lors de la création de l'instance du bean
- Intercepteur de cycle de vie
  - Annotation `@PostConstruct`

```
package org.easybeans.exemple.calculatrice;
```

```
import javax.annotation.PostConstruct;  
import javax.interceptor.Interceptor;
```

```
/**  
 * Intercepteur de cycle de vie  
 * @author Florent BENOIT  
 */
```

```
public class CalculatriceInter
```

```
/**  
 * Appel lors de la création  
 * @param invocationContext  
 */
```

```
@PostConstruct
```

```
public void postConstruction(final InvocationContext invocationContext) {  
    System.out.println("Appel de la méthode PostConstruct sur le bean " + invocationContext.getTarget());  
    try {  
        invocationContext.proceed();  
    } catch (Exception e) {  
        throw new RuntimeException("Cannot proceed invocationContext", e);  
    }  
}
```

```
}
```

```
public interface InvocationContext {  
    Object getTarget();  
    Method getMethod();  
    Object[] getParameters();  
    setParameters(Object[]);  
    Map getContextData(); // partagé par tous les intercepteurs  
    Object proceed() throws Exception; // appel du prochain intercepteur  
}
```



Démo



Et le debug ?

# Activer le mode debug

- Lancer JOnAS en mode debug :
- `jonas start -fg -debug -p 4142 -s y`
  - `-fg` = foreground. Par défaut = background
  - `-debug` = active le mode debug
  - `-p <xx>` le N° de port sur lequel écouter pour le debug
  - `-s [y/n]`
    - si « y », en attente d'un client de DEBUG comme Eclipse avant de démarrer la machine virtuelle Java.
    - Si « n », on démarre la machine virtuelle java et les clients se connectent quand ils veulent.
- Lancement d'Eclipse
  - Run/Open Debug dialog...
    - Remote Java Application. Localhost / 4142



Démo



Bean Stateful (avec état)

# Création d'un bean stateful : Panier d'achat

## ■ Définition d'une interface

```
package org.easybeans.exemple.panier;

import javax.ejb.Remote;

/**
 * Interface d'un panier
 * @author Florent Benoit
 */
@Remote
public interface PanierInterface {

    /**
     * Achete pour le montant donné
     * @param montant le montant donné
     */
    void acheter(int montant);

    /**
     * Récupère le montant du panier
     * @return le montant du panier
     */
    int lectureMontant();
}
```

# Bean Panier

```
@Stateful(mappedName="Panier")
public class Panier implements PanierInterface {

    /**
     * Montant total du panier.
     */
    private int total = 0;

    /**
     * Initialisation des montants.
     */
    public Panier() {
        total = 0;
    }

    /**
     * Achète pour le montant donné
     * @param montant le montant donné
     */
    public void acheter(final int montant) {
        this.total += montant;
    }

    /**
     * Récupère le montant du panier
     * @return le montant du panier
     */
    public int lectureMontant() {
        return total;
    }
}
```

# Création du client

- Obtenir une instance du bean :

```
/**
 * Obtient une instance du bean calculatrice.
 * @return une instance du bean
 * @throws Exception si le bean n'est pas trouvé.
 */
private PanierInterface getPanier() throws Exception {
    Context initialContext = new InitialContext();
    Object o = initialContext.lookup("Panier");

    if (o instanceof PanierInterface) {
        PanierInterface panier = (PanierInterface) o;
        return panier;
    }
    throw new Exception("Object panier invalide");
}
```

- A chaque lookup, on a un bean différent
- Un bean stateful est lié à un seul client

# Effectuer un achat avec le bean panier

```
// Creation d'un panier
PanierInterface panier = null;
try {
    panier = getPanier();
} catch (Exception e) {
    afficheException(out, e);
}

// Achat
try {
    out.println("Achat pour 30 euros<br/>");
    panier.acheter(30);
} catch (Exception e) {
    afficheException(out, e);
}

// Lecture
try {
    out.println("Lecture du montant du panier<br/>");
    out.println(panier.lectureMontant() + " euros<br/>");
} catch (Exception e) {
    afficheException(out, e);
}

// Creation d'un nouveau panier
PanierInterface nouveauPanier = null;
try {
    nouveauPanier = getPanier();
} catch (Exception e) {
    afficheException(out, e);
}
```

```
Achat pour 30 euros
Lecture du montant du panier
30 euros
Lecture du montant du nouveau panier
0 euros
```

# Et les transactions ?

- Un bean stateful peut être notifié tout au long d'une transaction : Du début de la transaction au commit ou rollback
- Interface spéciale : `javax.ejb.SessionSynchronization`

```
public void afterBegin() throws RemoteException {  
}
```

```
public void beforeCompletion() throws RemoteException {  
}
```

```
public void afterCompletion(final boolean committed) throws RemoteException {  
}
```

# Ajout réel du montant que si commit de la tx

```
@Stateful(mappedName="Panier")
```

```
public class Panier implements SessionSynchronization, PanierInterface {
```

```
    /**
     * Montant total du panier.
     */
    private int total = 0;

    /**
     * Valeur dans la transaction, pas encore committé.
     */
    private int newtotal = 0;

    /**
     * Initialisation des montants.
     */
    public Panier() {
        total = 0;
        newtotal = total;
    }

    /**
     * Achète pour le montant donné
     * @param montant le montant donné
     */
    public void acheter(final int montant) {
        newtotal = newtotal + montant;
        return;
    }

    /**
     * Récupère le montant du panier
     * @return le montant du panier
     */
    public int lectureMontant() {
        return total;
    }
}
```

```
    public void afterBegin() throws RemoteException {
        newtotal = total;
    }

    public void beforeCompletion() throws RemoteException {
    }

    public void afterCompletion(final boolean committed)
        throws RemoteException {
        if (committed) {
            total = newtotal;
        } else {
            newtotal = total;
        }
    }
}
```

# Client avec gestion de la transaction

```
// Obtient une transaction
UserTransaction transaction = null;
try {
    transaction = getTransaction();
} catch (Exception e) {
    afficheException(out, e);
}

// Démarrer la transaction
try {
    out.println("Démarre transaction");
    transaction.begin();
} catch (Exception e) {
    afficheException(out, e);
}

// Creation d'un panier
PanierInterface panier = null;
try {
    panier = getPanier();
} catch (Exception e) {
    afficheException(out, e);
}

// Achat
try {
    out.println("Achat pour 30 euros");
    panier.acheter(30);
} catch (Exception e) {
    afficheException(out, e);
}
```

```
/**
 * Obtient une instance du gestionnaire de transaction
 * @return une instance du gestionnaire de transaction
 * @throws Exception si l'objet transaction n'est pas trouvé.
 */
private UserTransaction getTransaction() throws Exception {
    Context initialContext = new InitialContext();
    Object o = initialContext.lookup("java:comp/UserTransaction");

    if (o instanceof UserTransaction) {
        UserTransaction userTransaction = (UserTransaction) o;
        return userTransaction;
    }
    throw new Exception("Object UserTransaction invalide");
}
```

```
// Lecture
try {
    out.println("Lecture du montant du panier");
    out.println(panier.lectureMontant() + " euros");
} catch (Exception e) {
    afficheException(out, e);
}
```

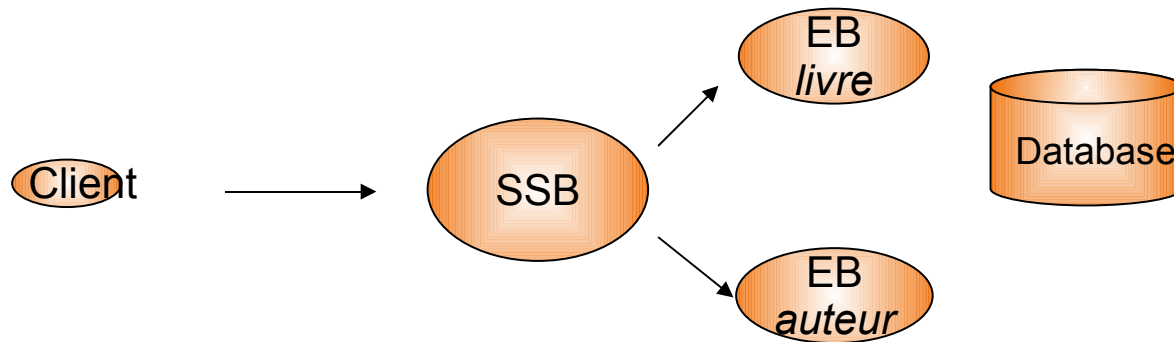
```
Démarre transaction
Achat pour 30 euros
Lecture du montant du panier avant commit
0 euros
Commit transaction
Lecture du montant du panier
30 euros
```



Stocker des informations en BDD avec  
les « Entities »

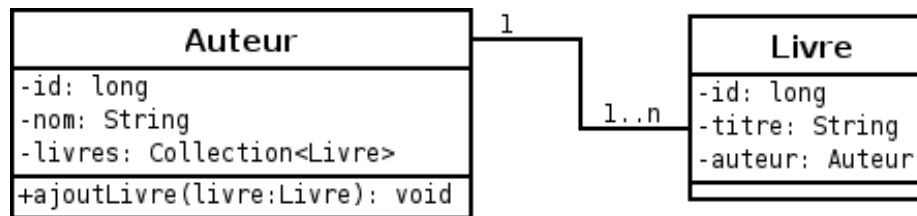
# Exemple de persistance : Bibliothèque

- Stocker/Lister les auteurs
- Stocker/Lister les livres
- Relation simpliste :
  - Un livre est écrit par un seul auteur
  - Un auteur peut écrire plusieurs livres



# Relation entre Livre et Auteur

- Auteur : identifiant, nom et une collection de livres
- Livre : identifiant, nom et lien vers un auteur
- Relations :
  - Un auteur peut écrire plusieurs livres
  - Un livre ne possède qu'un seul auteur



- => Définition de 2 POJOs : Livre et Auteur

# Entity Auteur

```
/**
 * Definition d'une classe representant l'auteur d'un livre.
 * @author Florent Benoit
 */
@Entity
@NamedQuery(name = "tousLesAuteurs", query = "select o FROM
Auteur o")
public class Auteur implements Serializable {
    private long id;
    private String nom = null;
    private Collection<Livre> livres;

    public Auteur() {
        livres = new ArrayList<Livre>();
    }

    /**
     * Non utilisation du mode Lazy.
     * @return livres écrits par cet auteur.
     */
    @OneToMany(mappedBy = "auteur", fetch = FetchType.EAGER,
cascade = CascadeType.ALL)
    public Collection<Livre> getLivres() {
        return livres;
    }

    /**
     * @return un identifiant (auto-incremental)
     */
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    public long getId() {
        return this.id;
    }
}
```

```
public void ajoutLivre(final String titre) {
    Livre livre = new Livre();
    livre.setTitre(titre);
    livre.setAuteur(this);
    livres.add(livre);
}

public void setLivres(final Collection<Livre> livres) {
    this.livres = livres;
}

public String getNom() {
    return nom;
}

public void setNom(final String nom) {
    this.nom = nom;
}

public void setId(final long id) {
    this.id = id;
}
```

# Entity Livre

```
@Entity  
@NamedQuery(name = "tousLesLivres", query = "select o FROM Livre o")  
public class Livre implements Serializable {
```

```
    private long id;  
    private Auteur auteur;  
    private String titre;
```

```
    @Id  
    @GeneratedValue(strategy = GenerationType.AUTO)  
    public long getId() {  
        return this.id;  
    }
```

```
    @ManyToOne  
    @JoinColumn(name = "Auteur_id")  
    public Auteur getAuteur() {  
        return auteur;  
    }  
  
    public void setAuteur(final Auteur auteur) {  
        this.auteur = auteur;  
    }  
    public String getTitre() {  
        return titre;  
    }  
    public void setTitre(final String titre) {  
        this.titre = titre;  
    }  
    public void setId(final long id) {  
        this.id = id;  
    }  
}
```

# Informations de persistance

- On souhaite indiquer le nom du datasource à utiliser
  - JTA ou non JTA
- Fournisseur de persistance
  - Hibernate Entity Manager, Apache OpenJPA, TopLink Essentials
- Dialecte avec la base de données
- Fichier persistence.xml :

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence" version="1.0">
<persistence-unit name="entity" transaction-type="JTA">
  <provider></provider>
  <jta-data-source jdbc_1</jta-data-source>
  <properties>
    <property name="hibernate.dialect" value="org.hibernate.dialect.HSQLDialect" />
    <property name="hibernate.hbm2ddl.auto" value="create-drop"/>
    <property name="toplink.target-database" value="HSQL"/>
    <property name="toplink.ddl-generation" value="drop-and-create-tables"/>
    <property name="toplink.ddl-generation.output-mode" value="database"/>
    <property name="openjpa.jdbc.DBDictionary" value="hsql"/>
    <property name="openjpa.jdbc.SynchronizeMappings" value="buildSchema(ForeignKeys=true)"/>
  </properties>
</persistence-unit>
</persistence>
```

# Interface pour accéder à ces Entities

- Définition d'une interface pour un bean stateless :
  - Initialisation des auteurs & livres
  - Obtenir la liste des auteurs
  - Obtenir la liste des livres

```
/**
 * Interface remote du bean jouant le role du bean "facade".
 * @author Florent Benoit
 */
@Remote
public interface SessionFacadeRemote {

    /**
     * Creation d'auteurs et de livres.
     */
    void init();

    /**
     * @return la liste des auteurs.
     */
    List<Auteur> listeDesAuteurs();

    /**
     * @return la liste des livres
     */
    List<Livre> listeDesLivres();
}
```

# Session Facade Bean réalisant l'accès aux entities

```
@Stateless
```

```
public class SessionFacade implements SessionFacadeRemote {
```

```
/**  
 * Entity manager utilisé par ce bean.  
 */
```

```
@PersistenceContext
```

```
private EntityManager entityManager = null;
```

```
public void init() {
```

```
    // livres de balzac
```

```
    Auteur balzac = new Auteur("Honore de Balzac");
```

```
    Livre pereGloriot = new Livre("Le Pere Goriot", balzac);
```

```
    balzac.getLivres().add(pereGloriot);
```

```
    Livre lesChouans = new Livre("Les Chouans", balzac);
```

```
    balzac.getLivres().add(lesChouans);
```

```
    // Enregistrement (seulement l'auteur car il y a l'attribut Cascade positionne a ALL).
```

```
    entityManager.persist(balzac);
```

```
    ...  
}
```

```
public List<Auteur> listeDesAuteurs() {
```

```
    Query auteurs = entityManager.createNamedQuery("tousLesAuteurs");
```

```
    return auteurs.getResultList();  
}
```

```
public List<Livre> listeDesLivres() {
```

```
    return entityManager.createNamedQuery("tousLesLivres").getResultList();  
}
```

```
}
```

Injection de  
l'objet

```
@OneToMany(mappedBy = "auteur", fetch =  
FetchType.EAGER, cascade =  
CascadeType.ALL)  
public Collection<Livre> getLivres() {}
```

```
@Entity  
@NamedQuery(name = "tousLesAuteurs", query = "select o FROM Auteur o")  
public class Auteur implements Serializable {
```

```
@Entity  
@NamedQuery(name = "tousLesLivres", query = "select o FROM Livre o")  
public class Livre implements Serializable {
```



# Le client

## ■ Commandes exécutées via le session bean facade

```
// Initialisation de la base
facadeBean.init();

// Recuperation de la liste des auteurs
List<Auteur> auteurs = facadeBean.listeDesAuteurs();

// Liste par auteur des livres ecrits par ceux-ci.
if (auteurs != null) {
    for (Auteur auteur : auteurs) {
        System.out.println("Liste de livres pour l'auteur dont le nom est '" + auteur.getNom() + "' :");
        Collection<Livres> livres = auteur.getLivres();
        if (livres == null) {
            System.out.println("- Aucun livre.");
        } else {
            for (Livres livre : livres) {
                System.out.println("- Livre ayant pour titre '" + livre.getTitre() + "'.");
            }
        }
    }
} else {
    System.out.println("Aucun auteur.");
}

// Recuperation de la liste des livres
System.out.println("");
System.out.println("Liste des livres :");
List<Livres> livres = facadeBean.listeDesLivres();
if (livres != null) {
    for (Livres livre : livres) {
        System.out.println(" - Livre avec titre '" + livre.getTitre() + "' ayant pour auteur '"
            + livre.getAuteur().getNom() + "'.");
    }
}
}
```



# Client léger

## ■ Initialisation uniquement si besoin

```
out.println("<br> Initialisation des livres...<br />");

List<Auteur> auteurs = null;
try {
    auteurs = getFacade().listeDesAuteurs();
} catch (Exception e) {
    afficheException(out, e);
    return;
}

if (auteurs != null && auteurs.size() > 0) {
    out.println("Initialisation déjà effectuée !<br/>");
} else {
    try {
        getFacade().init();
    } catch (Exception e) {
        afficheException(out, e);
    }
}
```

# Exécution de l'exemple

Initialisation des livres...

Initialisation déjà effectué !

Liste de livres pour l'auteur dont le nom est 'Honore de Balzac' :

- Livre ayant pour titre 'Le Pere Goriot'.
- Livre ayant pour titre 'Les Chouans'.

Liste de livres pour l'auteur dont le nom est 'Victor Hugo' :

- Livre ayant pour titre 'Les Miserables'.
- Livre ayant pour titre 'Notre-Dame de Paris'.

Liste des livres :

- Livre avec titre 'Le Pere Goriot' ayant pour auteur 'Honore de Balzac'.
- Livre avec titre 'Les Chouans' ayant pour auteur 'Honore de Balzac'.
- Livre avec titre 'Les Miserables' ayant pour auteur 'Victor Hugo'.
- Livre avec titre 'Notre-Dame de Paris' ayant pour auteur 'Victor Hugo'.



Démo



Exemple Message Driven Bean  
Les messages sont asynchrones

# MDB: JMS (Java Message Service)

## ■ Très simple

```
/**
 * Exemple de Message JMS
 * @author Florent Benoit
 */
@MessageDriven(activationConfig = {
    @ActivationConfigProperty(propertyName = "destination", propertyValue = "SampleQueue"),
    @ActivationConfigProperty(propertyName = "destinationType", propertyValue = "javax.jms.Queue")
})
public class MyJMSMDB implements MessageListener {

    /**
     * Reception de messages.
     * @param message le message donné
     */
    public void onMessage(final Message message) {
        String txt = "Reception d'un message '" + message + "'.";
        // Vérification
        if (message instanceof TextMessage) {
            try {
                txt += " Avec comme contenu '" + ((TextMessage) message).getText();
            } catch (JMSEException e) {
                System.err.println("Erreur lors de la reception du message");
                e.printStackTrace();
            }
        }
        // Affiche le texte
        System.out.println(txt);
    }
}
```

Destination

Réception  
du message

Plusieurs types  
de messages

# Envoi des messages depuis un client

Factory JMS

```
// Obtenir factory
QueueConnectionFactory queueConnectionFactory;
try {
    queueConnectionFactory = (QueueConnectionFactory) initialContext.lookup(QueueConnectionFactory.class.getName());
} catch (NamingException e) {
    afficheException(out, e);
    return;
}

// Récupérer JMS Queue
Queue queue;
try {
    queue = (Queue) initialContext.lookup(SAMPLE_QUEUE);
} catch (NamingException e) {
    afficheException(out, e);
    return;
}

// Création connexion
QueueConnection queueConnection = queueConnectionFactory.createQueueConnection();

// Création session
QueueSession queueSession = queueConnection.createQueueSession(false, Session.AUTO_ACKNOWLEDGE);

// Création expéditeur
QueueSender queueSender = queueSession.createSender(queue);

// Envoi des messages
TextMessage message = null;
for (int i = 0; i < NUMBER_MESSAGES; i++) {
    message = queueSession.createTextMessage();
    String text = "Message_" + i;
    message.setText(text);
    queueSender.send(message);
    out.println("Envoi du message [" + message.getJMSMessageID() + ", texte:" + text + "]\n");
}
}
```

JMS Queue

Paramètres

Création du message

Envoi du message !



Démo



Hibernate / EJB3

# Class POJO: Cubicle

```
package example;

public class Cubicle {

    private long id;

    private String name;

    private Employee residentEmployee;

    public Employee getResidentEmployee() {
        return residentEmployee;
    }
    public void setResidentEmployee(Employee employee)
    {
        this.residentEmployee = employee;
    }

    public long getId() {
        return id;
    }

    public void setId(long id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

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



# Class POJO: Employee

```
package example;

public class Employee {

    private long id;
    private Cubicle assignedCubicle;
    private String name;
    private String address;

    public String getAddress() {
        return address;
    }
    public void setAddress(String address) {
        this.address = address;
    }

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

    public Cubicle getAssignedCubicle() {
        return assignedCubicle;
    }
    public void setAssignedCubicle(Cubicle cubicle)
    {
        this.assignedCubicle = cubicle;
    }

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



# Hibernate Configuration

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
    "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
    "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
  <session-factory>
    <!-- Database connection settings -->
    <property name="connection.driver_class">org.hsqldb.jdbcDriver</property>
    <property name="connection.url">jdbc:hsqldb:hsql://localhost</property>
    <property name="connection.username">sa</property>
    <property name="connection.password"></property>

    <!-- JDBC connection pool (use the built-in) -->
    <property name="connection.pool_size">1</property>

    <!-- SQL dialect -->
    <property name="dialect">org.hibernate.dialect.HSQLDialect</property>

    <!-- Enable Hibernate's automatic session context management -->
    <property name="current_session_context_class">thread</property>

    <!-- Disable the second-level cache -->
    <property name="cache.provider_class">org.hibernate.cache.NoCacheProvider</property>

    <!-- Echo all executed SQL to stdout -->
    <property name="show_sql">>false</property>

    <!-- Drop and re-create the database schema on startup -->
    <property name="hbm2ddl.auto">create</property>

    <mapping resource="example/Cubicle.hbm.xml"/>
    <mapping resource="example/Employee.hbm.xml"/>

  </session-factory>
</hibernate-configuration>
```

# Mapping for the classes

## ■ Cubicle:

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
    "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
    "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
  <class name="example.Cubicle" table="CUBICLE">
    <id name="id" column="CUBICLE_ID">
      <generator class="native" />
    </id>
    <one-to-one name="residentEmployee" class="example.Employee" .../>
  </class>
</hibernate-mapping>
```

## ■ Employee:

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
    "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
    "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
  <class name="example.Employee" table="EMPLOYEE">
    <id name="id" column="EMPLOYEE_ID">
      <generator class="native" />
    </id>
    <property name="name" />
    <property name="address" />
    <one-to-one name="assignedCubicle" class="example.Cubicle" outer-join="true"/>

    <query name="listAllEmployees">from Employee</query>
  </class>
</hibernate-mapping>
```

# EJB3

```
@Entity
public class Employee {
    private Cubicle assignedCubicle;
    @OneToOne
    public Cubicle getAssignedCubicle() { return assignedCubicle; }
    public void setAssignedCubicle(Cubicle cubicle) { this.assignedCubicle = cubicle; }
    ...
}
```

```
@Entity
public class Cubicle {
    private Employee residentEmployee;
    @OneToOne(mappedBy="assignedCubicle")
    public Employee getResidentEmployee() { return residentEmployee; }
    public void setResidentEmployee(Employee employee) {
        this.residentEmployee = employee;
    }
    ...
}
```

# Configuration

## ■ Hibernate configuration file:

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
    "-//Hibernate/Hibernate Configuration DTD"
    "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd" [
<hibernate-configuration>
  <session-factory>
    <!-- Database connection settings -->
    <property name="connection.driver_class">org.hibernate.jdbc.JDBCDriver</property>
    <property name="connection.url">jdbc:hsqldb://localhost:9001/</property>
    <property name="connection.username">sa</property>
    <property name="connection.password"></property>

    <!-- JDBC connection pool (use the built-in) -->
    <property name="connection.pool_size">1</property>

    <!-- SQL dialect -->
    <property name="dialect">org.hibernate.dialect.HSQLDialect</property>

    <!-- Enable Hibernate's automatic session context management -->
    <property name="current_session_context_class">thread</property>

    <!-- Disable the second-level cache -->
    <property name="cache.provider_class">org.hibernate.cache.NoCacheProvider</property>

    <!-- Echo all executed SQL to stdout -->
    <property name="show_sql">>false</property>

    <!-- Drop and re-create the database schema on startup -->
    <property name="hbm2ddl.auto">create</property>

    <mapping resource="example/Cubicle.hbm.xml"/>
    <mapping resource="example/Employee.hbm.xml"/>

  </session-factory>
</hibernate-configuration>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence 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_1_0.xsd"
  version="1.0">
  <persistence-unit name="entity" transaction-type="JTA">
    <provider></provider>
    <jta-data-source>jdbc_1</jta-data-source>
    <properties>
      <property name="hibernate.dialect" value="org.hibernate.dialect.HSQLDialect" />
      <property name="hibernate.hbm2ddl.auto" value="create-drop"/>
      <property name="toplink.target-database" value="HSQL"/>
      <property name="toplink.ddl-generation" value="drop-and-create-tables"/>
      <property name="toplink.ddl-generation.output-mode" value="database"/>
      <property name="openjpa.jdbc.DBDictionary" value="hsqldb"/>
    </properties>
  </persistence-unit>
</persistence>
```

## ■ Hibernate :

```
Session session = HibernateUtil.getSessionFactory().getCurrentSession();  
  
session.beginTransaction();  
  
POJO pojo = new POJO();  
pojo.setXXX(xxx);  
pojo.setYYY(yyy);  
  
session.save(pojo);  
  
session.getTransaction().commit();
```

## ■ EJB 3 :

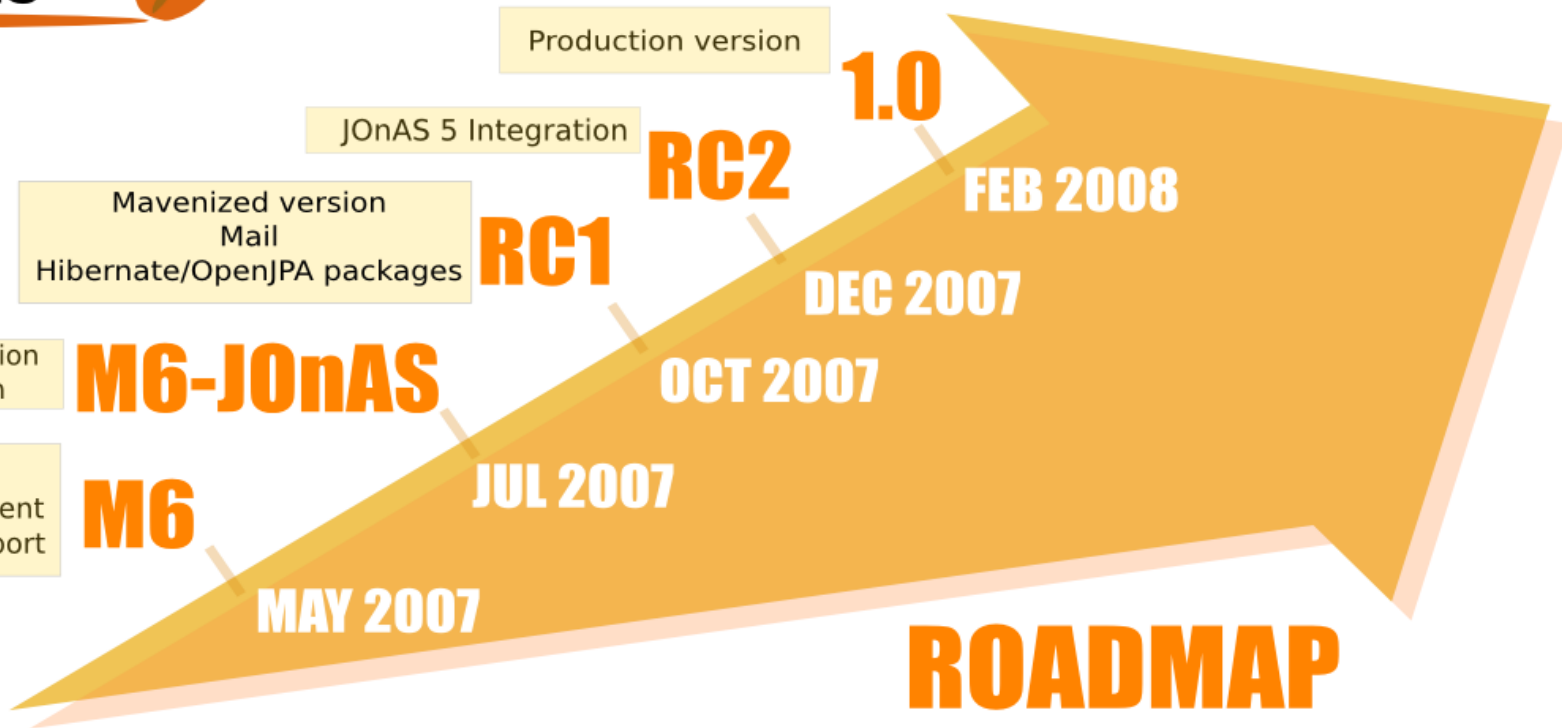
```
/**  
 * Entity manager utilisé par ce bean.  
 */  
@PersistenceContext  
private EntityManager entityManager = null;  
  
public void ....() {  
    POJO pojo = new POJO();  
    pojo.setXXX(xxx);  
    pojo.setYYY(yyy);  
  
    entityManager.persist(pojo);  
}
```

Les transactions sont gérées  
par le conteneur :  
Pas de begin() et commit()



## ROADMAP

# RoadMap



## ROADMAP





Questions ?