

Java EE 5 Enterprise JavaBeans™ 3.0

Alexis Moussine-Pouchkine
Sun Microsystems France



Plate-forme Java EE 5

- EJB 3.0 (JSR-220)
- JavaServer Faces (JSR-252)
- New persistence API (JSR-220)
- StAX (JSR-173)
- JAXB (JSR-222)
- JAX-WS (JSR-224)
- Web Services Metadata (JSR-181)
- Common Annotations (JSR-250)

Project GlassFish



- Sun Java Application Server 9 en Open Source
 - > Parfait pour se former à Java EE 5 (Blueprints)
 - > Parfait pour une mise en production (Performance)
- Innovation et collaborations
 - > Framework HTTPd Grizzly, management de Web Services, Oracle Toplink, outils d'admin, call-flow analysis, load on-demand, etc...
 - > Spring, Hibernate, Struts, iBatis, AppFuse, Wicket, JSPWiki, WebWork, Jive, OpenSymphony workflow, Roller, etc...
 - > Version Java EE 5 dès sortie officielle (printemps 2006)
 - > <http://glassfish.dev.java.net>
- Outils de développement
 - > NetBeans IDE 5.5 synchronisé avec Java EE 5 et GlassFish
 - > Plugin pour Eclipse WTP

Objectifs EJB™ 3.0

Simplification des
Développements

avec

Compatibilité et
Migration

Un peu de contexte

- Java 5 (JDK 1.5.0) sorti en septembre 2004
 - > Nouveautés du langage (generics, enum, annotations, ...)
 - > Evolution importante de presque toutes les API
 - > Performance, management et monitoring
- Java EE 5 (J2EE 1.5) en cours de développement
 - > Thèmes :
 - > Ease of Development
 - > Ease of Development
 - > Ease of Development
 - > Travail quasi-finalisé (JSR 244)
 - > Intégration JSF 1.2 (JSR 252)
 - > EJB 3.0 (JSR 220)

Approche EJB 3.0

- Simplification des API manipulées
 - > POJO et POJI
 - > Descripteurs de déploiements rendus facultatifs
- Utilisation de meta-données sous forme d'annotations Java
 - > Nombreuses options par défaut pour les cas les plus fréquents (configuration par exception)
 - > Expression de meta-données si nécessaire (ex: mapping O/R)
- Le conteneur décharge le développeur de certaines de ses responsabilités

Annotations

- Notion introduite dans Java 5 (JSR 175)
- Décoration d'éléments du code source pour exprimer le comportement d'une application
- Alternative à XDoclet:
 - > JSR intégré au langage et à la plate-forme Java
 - > Génération de code et manipulation de bytecode (*.class)
- Meta informations dans le fichier source
 - > Gestionnaire de source, gestion des diff, éditeurs et outils simplifiés
- Descripteurs de déploiement toujours disponibles
 - > Redéfinition des annotations

Exemple EJB 2.1

```
public class ShopCartBean implements javax.ejb.SessionBean {
    private SessionContext ctx;
    DataSource productDB;

    public void setSessionContext(SessionContext ctx) {
        this.ctx = ctx;
    }

    public void ejbCreate() {
        Context initialContext = new InitialContext();
        productDB = (DataSource)initialContext.lookup(
            "java:comp/env/jdbc/myDB");
    }

    public void ejbActivate() {}
    public void ejbPassivate() {}
    public void ejbRemove() {}

    public Collection startToShop (String productName) {
        Connection conn = productDB.getConnection();
        ...
    }
}
```

Exemple EJB 2.1 (suite)

```
public interface ShopCart extends EJBObject {  
    public Collection startToShop (String  
        productName)  
        throws RemoteException;  
    ...  
}
```

```
public interface ShopCartHome extends EJBHome {  
    ShopCart create()  
        throws CreateException, RemoteException;  
}
```

Exemple client EJB 2.1

...

```
Context initialContext = new InitialContext();
```

```
ShopCartHome myCartHome = (ShopCartHome)
```

```
    initialContext.lookup("java:comp/env/ejb/cart");
```

```
ShopCart myCart= myCartHome.create();
```

```
//Utilisation du composant
```

```
Collection widgets = myCart.startToShop("widgets")
```

...

```
// Code pour gérer javax.ejb.CreateException
```

...

Descripteur Déploiement EJB 2.1

```
<session>
  <ejb-name>ShopCartBean</ejb-name>
  <local-home>ShopCartHome</local-home>
  <local>ShopCart</local>
  <ejb-class>com.example.ShopCartBean</ejb-class>
  <session-type>Stateful</session-type>
  <transaction-type>Container</transaction-type>
  <resource-ref>
    <res-ref-name>jdbc/myDB</res-ref-name>
    <res-ref-type>javax.sql.DataSource</res-ref-type>
    <res-auth>Container</res-auth>
  </resource-ref>
</session>
...
<assembly-descriptor>
...
</assembly-descriptor>
```

Exemple EJB 3.0

```
@Remote // s'applique à l'ensemble des méthodes
public interface ShopCart {
    public Collection startToShop(String productName);
    public void checkout();
}

@Stateful
public class ShopCartBean implements ShopCart {
    @Resource DataSource productDB;
    @Init /* Première méthode métier */
    public Collection startToShop (String productName) {
        Connection conn = productDB.getConnection();
        ...
    }
    @Remove public void checkout(){ ... }
}
```

Plus de descripteur de déploiement

Simplifications EJB 3.0

- Interface métier = interface Java
- Un composant peut implémenter plusieurs interfaces
- Plus de référence à **EJBHome** ou **EJBObject**
- Génération automatique d'interface possible
- Plus de méthode imposée de cycle de vie
- Plus de JNDI pour le développeur ou le client
- Accès distant transparent (plus de **RemoteException**)

Annotations EJB 3.0

- Nature de l'EJB
 - > `@Stateless`, `@Stateful`, `@MessageDriven`, `@Entity`
- Accès distant ou local
 - > `@Remote`, `@Local`
 - > Plus de `RemoteException`
- Cycle de vie
 - > `@PostConstruct`, `@PostActivate`, `@PreDestroy`, `@PrePassivate`, `@Remove`, ...
 - > Selon les natures d'EJB
 - > Ex: `@Remove` indique une suppression de l'entité après exécution de la méthode annotée

Injection

- Le composant déclare ce dont il a besoin au travers de meta-données (annotations)
- Le conteneur injecte les ressources demandées à la création du composant
 - > Référence EJB
 - > Contexte de transaction
 - > Contexte de sécurité
 - > Notifications de timer
 - > Contexte de persistance
 - > ...
- L'injection ne fonctionne que sur des objets *managed* (créé par le conteneur)

Annotations d'injection

- **@EJB**
 - > Annote les interfaces métier des EJB (déclaration ou *setter*)
 - > Annote les interfaces Home (accès composants EJB 2.1)
- **@Resource**
 - > Permet l'injection de tout ce qui se trouve dans JNDI
 - > Permet d'injecter **SessionContext**
UserTransaction**EntityManager** ...
 - > Injection de variable d'instance ou de *setter*
- Injection possible dans le descripteur de déploiement

Configuration par défaut

```
package exemple.pkg;  
@Stateless  
public class EchoBean implements Echo {  
    @EJB private Hello helloBean;  
}
```

est équivalent à :

```
package exemple.pkg;  
@Stateless  
public class EchoBean implements Echo {  
    @EJB (name="exemple.pkg.EchoBean/helloBean",  
        beanName="HelloBean",  
        beanInterface=Hello.class)  
    private Hello helloBean;  
}
```

- Un seul EJB implémente l'interface Hello
- l'interface distante et l'attribut ont le même type
- le nom relatif JNDI est `exemple.pkg.EchoBean/helloBean`

Autres Services EJB 3.0

- Transactions
 - > Container-Managed Transaction (CMT) par défaut
 - > Attribut de transaction REQUIRED par défaut
 - > Bean-Managed Transaction (BMT) disponible via annotation
 - > **@TransactionAttribute (MANDATORY)**
 - > S'applique aux appels de méthodes imbriqués
- Sécurité
 - > Effectué en entreprise au déploiement
 - > Le développeur peut donner des indications
 - > Annotations de permissions
 - > **@Unspecified** => géré au déploiement
 - > **@Unchecked**, **@SecurityDomain**, **@RolesAllowed**, **@RunAs**
 - > Applicables à une classe ou à une méthode

Intercepteurs

- Fonctionnalité avancée pour intercepter des invocations de méthodes
- Manipulation des arguments et du résultat
- Annotations **@Interceptors** et **@AroundInvoke**
- Peut être spécifié dans un descripteur de déploiement
- Exemples
 - > Sécurité, audit, log, etc...
- Similaire par certains aspects à l'AOP

```
@Interceptors( { Interceptor.class } )  
public class HelloWorldBean {  
    public String sayHi() {  
        return "Hi";  
    }  
}  
  
public class Interceptor {  
    @AroundInvoke  
    public Object log(InvocationContext ic) {  
        System.err.println(  
            ic.getMethod().getName() + " called." );  
        return ic.proceed();  
    }  
}
```

Persistence EJB 3.0

- Objectifs initiaux
 - > Simplification du modèle de programmation EJB entité
 - > Meilleure modélisation de domaines métier
 - > Relations, héritage, polymorphisme
 - > Meilleures capacités de requêtage
 - > Spécification mapping O/R
 - > EJB entité utilisables en dehors du conteneur EJB (vs. Data Transfert Object et autres anti-patterns)
- Objectif supplémentaire
 - > Rendre la persistance utilisable en dehors sur conteneur J2EE
- A terme, spécification séparée
 - > Java Persistence API (JPA)

Périmètre de la Persistance

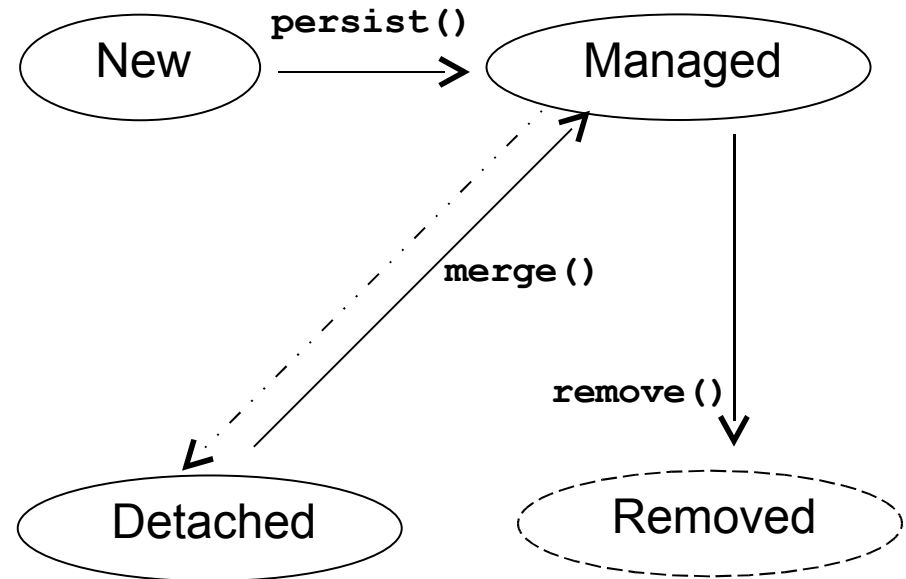
- Modèle de persistance unique et mapping O/R
 - > API de persistance dans un document de spécification séparé
 - > API complémentaires pour une utilisation en dehors de J2EE
- Elargissement du groupe d'experts JSR-220
 - > Editeurs JDO et autres experts venus du JSR 243 (JDO 2)
- Synthèse des expertises Hibernate, JDO, TopLink et d'éditeurs J2EE/EJB
 - > L'API de persistance tire son inspiration de toutes ces sources
 - > Combinaison des idées et des expériences, pas des API
- Flexibilité
 - > Utilisable avec ou sans conteneur J2EE
 - > A terme, providers (implémentations) inter-changeable

Entité EJB

- Un EJB entité est une simple classe Java
 - > Classe concrète – constructeur sans argument (test avec **new**)
 - > Héritage et polymorphisme
 - > Méthodes **getXXX/setXXX** ou attributs public (hors **@Transient**) pour les propriétés persistantes
 - > Pas d'interface à implémenter
 - > Pas de méthode de callback requise
- Utilisable comme objet « détaché » dans d'autres parties de l'application
 - > Plus besoin de Data Transfer Object (DTO)
 - > Doit implémenter **Serializable**

Cycle de vie EJB entité

- **New**
 - > Pas d'identité de persistance
 - > Pas associé au conteneur
- **Managed**
 - > Identité de persistance
 - > Géré par le conteneur
 - > Synchronisation SGBDR
 - > Gestion des relations
- **Detached**
 - > Identité de persistance
 - > Non géré par le conteneur
 - > Synchronisation nécessite un **merge**
 - > Pas de gestion des relations



EntityManager

- **EntityManager** est un point de contrôle des opérations sur les EJB entité
 - > Méthodes de cycle de vie
 - > `persist`, `remove`, `merge`, `flush`, `find`, `refresh`, etc
 - > Méthodes de requêtage pré-définies (find)
 - > `EntityManager.find(MyClass.class, id);`
 - > Factory pour des objets **Query**
 - > Requêtes (nommées) statiques et dynamiques
 - > Requêtes EJB-QL et SQL
 - > Gestion du contexte de persistance
 - > Réunion des identités persistantes
 - > Contexte étendu de persistance (transactions multiples)
 - > Similaire à Hibernate Session, JDO PersistenceManager, etc.

Injection EntityManager

```
@Stateless()  
public class CreateUsersFactory implements  
        CreateUsersFactoryLocal {  
    @PersistenceContext  
    EntityManager em;  
  
    public CreateUsersFactory() {  
        UserCredentials user =  
            new UserCredentials("scott","tiger");  
        em.persist(user);  
    }  
}
```

- L'object est écrit en base à la sortie de la méthode (CMT)
- Injection également possible dans le conteneur web (servlet, ...) ou dans un ACC (Application Client Container)

@PersistenceContext

- Contexte de persistance:
 - > Propose un accès à des unités de persistance (1 à N) pour un ensemble de composants
- Unité de persistance:
 - > Classes mises en correspondance avec une source de données unique

persistence.xml (dans EJB-JAR, WAR, ou JAR) :

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="1.0">
  <persistence-unit name="DEFAULT"
                    transaction-type="JTA">
    <jta-data-source>jdbc/__default</jta-data-source>
    <properties>
      <property name="toplink.ddl-generation"
                value="drop-and-create-tables"/>
    </properties>
  </persistence-unit>
</persistence>
```

Version application-managed possible

```
@Entity
public class Customer {           // table = CUSTOMER
    private Long id;
    private String name;
    private Address address;
    private HashSet<Order> orders = new HashSet<Order>();

    @Id                           // clé primaire
    @GeneratedValue(strategy = AUTO) // géré par la base
    public Long getID() {
        return id;
    }

    private void setID(Long id) { this.id = id; }
    public String getName() { return name; }
    public void setName(String name) { this.name = name; }

    @OneToMany(cascade=ALL, fetch=EAGER)
    public Set<Order> getOrders() { // cardinalité
        return orders;
    }

    public void setOrders(Set<Order> orders) {
        this.orders = orders;
    }
}
```

```
@Entity
@Table(name="ORDER_TBL")          // nom table explicite
public class Order {

    ...

    @Id                          // clé primaire
    @GeneratedValue(strategy = AUTO)
    public Long getID() {
        return id;
    }

    @ManyToOne(cascade=ALL)        // create, merge, delete
    @JoinColumn(name="cust_id")    // clé étrangère
    public Customer getCustomer() {
        return cust;
    }

    public setCustomer ( Customer c ) {
        this.customer = c;
    }

    ...

}
```

Vue cliente

```
@Stateless
public class OrderEntryBean {
    @PersistenceContext
    private EntityManager em;          // singleton

    public void addUser (Customer cust) {
        em.persist (cust);           //
    }  // Ecriture en base (CMT par défaut)

    public void enterOrder (int custId, Order newOrder) {
        Customer cust = em.find(Customer.class, custId);
        cust.getOrders().add(newOrder);    // Pointeur de retour
        newOrder.setCustomer(cust);
    }  // Ecriture en base (CMT par défaut)

    public Order createOrder (String id, ...) {
        return new Order (String id, ...);
    }
}
```

Persistence Java SE

- Nécessite des classes de « bootstrap »
 - > `javax.persistence.Persistence`
 - > `javax.persistence.EntityManagerFactory`
- Exemple

```
public class EmployeeLoader {  
    public static void main(String[] args) {  
        EntityManagerFactory emf = Persistence  
            .createEntityManagerFactory("HRSysSystem");  
        EntityManager em = emf.createEntityManager();  
        em.getTransaction().begin();  
        em.persist(new Order(args[0], args[1]));  
        em.getTransaction().commit();  
        em.close();  
        emf.close();  
    }  
}
```

Relations

- Uni ou bi-directionnelles
- Cardinalités
 - > **@OneToOne**
 - > **@OneToMany**
 - > **@ManyToOne**
 - > **@ManyToMany**
- Cascade
 - > Propagation d'une opération sur les relations (attribut de l'annotation)
 - > **PERSIST, REMOVE, MERGE, REFRESH, ALL**
 - > Aucune propagation par défaut

Héritage

- Rendu possible par le modèle POJO
 - > Héritage de classes abstraites ou concrètes, entité ou non
 - > Annotation de la classe
 - > `@Inheritance(strategy=InheritanceType.SINGLE_TABLE)`
- Stratégies de correspondance vers une base
 - > **SINGLE_TABLE**(par défaut) – une seule table, valeur discriminatoire relative à la classe
 - > **TABLE_PER_CLASS**(optionnel) – une table pour chaque classe concrète dans la hiérarchie
 - > **JOINED** – jointure vers d'autres tables spécifiques aux attributs définis dans les sous-classes

- SINGLE_TABLE

ANIMAL				
ID	DISC	NAME	LEG_CNT	WING_SPAN

- TABLE_PER_CLASS

LAND_ANML			AIR_ANML		
ID	NAME	LEG_COUNT	ID	NAME	WING_SPAN

- JOINED

ANIMAL	
ID	NAME

LAND_ANML		AIR_ANML	
ID	LEG_COUNT	ID	WING_SPAN

Requêtes EJB-QL

Dynamiques :

```
public List findWithName(String name) {  
    return em.createQuery(  
        "SELECT c from Customer c WHERE c.name LIKE :custName" )  
        .setParameter("custName", name)  
        .setMaxResults(10)  
        .getResultList();  
}
```

Polymorphiques :

```
select avg(e.salary) from Employee e where e.salary > 80000
```

Nommées :

```
@NamedQuery(name="findAllCustomersWithName",  
    query="SELECT c FROM Customer c WHERE c.name LIKE ?1")  
  
customers = em.createNamedQuery("findAllCustomersWithName")  
    .setParameter(1, "Smith")  
    .getResultList();
```

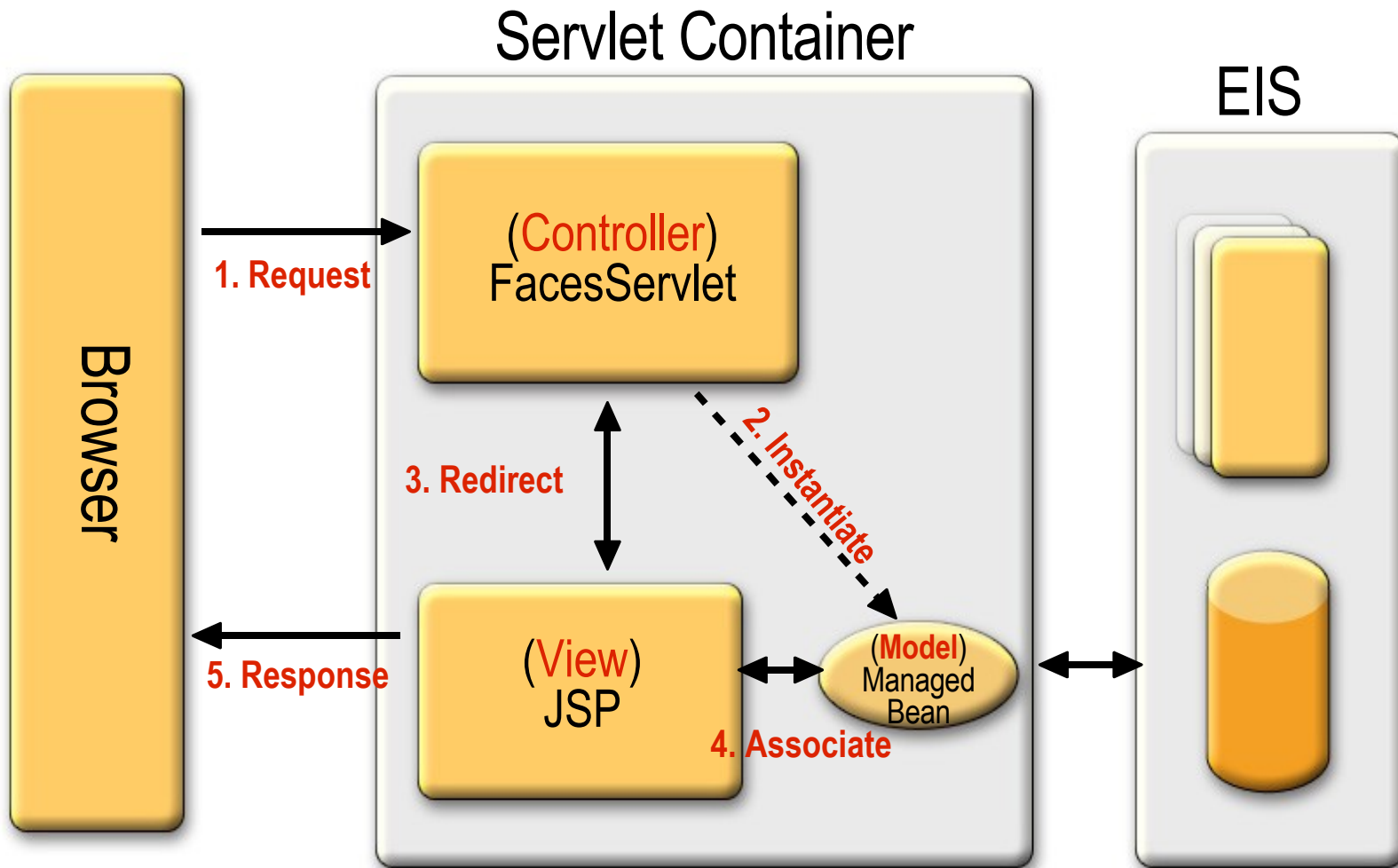
Evolutions EJB-QL

- Requêtes Natives SQL – résultats sous forme d'objets
- Bulk updates – plus besoin de tout monter en cache
- Requêtes dynamiques
- Pagination – **Query.setMaxResults()**
- Projections – Objets de types différents ou partiels
- Jointures (JOIN)
- Clauses GROUP BY, HAVING
- Fonctions supplémentaires: **concat()**, **trim()**, **substring()**, **lower()**, **upper()**, **length()**, **locate()**, **abs()**, **sqrt()**, **size()**, **mod()**

Java Server Faces (JSF 1.2)

- Framework MVC2
- Modèle de composant extensible
- Modèle de rendu extensible
- Modèle de gestion d'événements
- Framework de validation
- Gestion des règles de navigation
- Internationalisation
- Accessibilité
- Tool friendly

JSF Extending Servlet/JSP



AJAX et JavaServer Faces

- Ajax « BluePrints Solutions catalog » :
<https://bpcatalog.dev.java.net/nonav/solutions.html>
 - > AJAX utilisant l'architecture JavaServer Faces pour traiter les requêtes AJAX
 - > Les composants JSF effectuent le rendu en encapsulant et en exécutant les scripts AJAX
 - > Le Servlet Controller gère les requêtes AJAX
- AJAX est intégré au cycle de vie JSF
 - > Un écouteur de phase est utilisé pour récupérer le code JavaScript initial
 - > Un écouteur de phase est utilisé pour exécuter les requêtes AJAX
 - > L'état de la page est accessible depuis le code JavaScript

Web Service Java EE 5

```
package endpoint;
```

```
import javax.jws.WebService;
```

```
@WebService
```

```
public class MyHelloService {
```

```
    @WebMethod
```

```
    public String sayHello(String param) {  
        return "Hello " + param;  
    }
```

```
}
```

Client Web Service Java EE 5

```
public class MyComponent {  
    @WebServiceRef(MyHelloService.class)  
    HelloIF helloPort;  
  
    public void myMethod() {  
        // Invoke an operation  
        helloPort.sayHello();  
    }  
}
```

Web Services dans Java EE 5

- JAX-WS 2.0 (Java API for XML Web Services)
 - > Nouvelle spécification, successeur de JAX-RPC
 - > JAX-RPC 1.1 reste supporté
- Implémentation de la nouvelle pile WS
 - > JAX-WS 2.0 et JAXB 2.0
 - > StAX et JAXP 1.3
 - > Conçu pour être évolutif (spécifications WS-* en cours)
- Utilisation importantes des annotations
 - > Simplification des développements
- Support Fast Infoset (GlassFish)
 - > Meilleure Performance

Conclusion

- EJB 3.0 – Simplification réelle pour le développeur
 - > Suppression de classes et d'interfaces (POJO)
 - > Modèle d'injection de services et de ressources
 - > Descripteurs de déploiement rendus facultatifs
 - > Capacités de requêtage améliorées
 - > Spécification du mapping O/R (relations)
- EJB 3.0 et JSF 1.2 sont les fondations de Java EE 5
 - > Ease of Development (EoD)
- Disponible en beta (version finale en mai)

<http://java.sun.com/javaee/5/docs/tutorial/doc/index.html>

Rôle des outils pour Java EE 5

- L'éditeur devient le point central
 - > Besoin d'assistants, d'aide en ligne, de hints/tips
 - > Rapport d'erreur en ligne (syntaxe, sémantique, blueprints)
- Analyse des annotations
 - > Doit intervenir avant la compilation ou le déploiement
 - > Complétion de code pour les annotations (y compris pour la persistance)
- L'outil doit connaître la sémantique de chaque annotation
 - > Quelles sont les conséquences et les différences entre `@Stateless` et `@Stateful`

Demo

Project GlassFish



- Sun Java Application Server 9 en Open Source
 - > Parfait pour se former à Java EE 5 (Blueprints)
 - > Parfait pour une mise en production (Performance)
- Innovation et collaborations
 - > Framework HTTPd Grizzly, management de Web Services, Oracle Toplink, outils d'admin, call-flow analysis, load on-demand, etc...
 - > Spring, Hibernate, Struts, iBatis, AppFuse, Wicket, JSPWiki, WebWork, Jive, OpenSymphony workflow, Roller, etc...
 - > Version Java EE 5 dès sortie officielle (printemps 2006)
 - > <http://glassfish.dev.java.net>
- Outils de développement
 - > NetBeans IDE 5.5 synchronisé avec Java EE 5 et GlassFish
 - > Plugin pour Eclipse WTP

Sun Developer Network

- Free resources for all developers
 - > Tools (Creator, Java Studio Enterprise Edition)
 - > Forums
 - > FAQs
 - > Early Access
 - > Newsletters
- Join Today!
 - > <http://developers.sun.com>



Get Connected

Sun Developer Network connects you to what you need, when you need it.

[Join Now >>](#)

Sun Partner Advantage Community :

- For professional developers (for companies only)
 - > Information and training
 - > Technical resources for porting/migration/integration
 - > Marketing tools
 - > Sales engagement
- To know much more, contact :
- jean-michel.pelmont@sun.com or,
partneradvantage.sun.com



Get Connected

Sun Developer Network connects you to what you need, when you need it.

[Join Now >>](#)

Alexis Moussine-Pouchkine
alexis.mp@sun.com
blogs.sun.com/alexismp

