Développement de composant métier

ENTERPRISE JAVA* BEANS

* Just Another Vague Acronym

Plan du cours

I. Introduction
- Selon vous…
- EJB
- Applications distribuées – Applications centralisées
- Architectures 1,2,3,n Tiers
- EJB Historique
- EJB Container

II. Les différents type d'EJB
- Session Beans
- Entity Beans
- MDB – Message Driven Beans

III. L'accès aux données
- JPQL – utiliser le langage de requête de JPA

IV. Les Transaction
- JTA comme context Transactionnel
- Container Managed Transaction
- Bean Managed Transaction

V. Les Intercepteurs

VI. La Sécurité

INTRODUCTION

INTRODUCTION

Jusqu'a présent vous avez probablement abordé Java SE – les applications consoles, AWT, SWING, Java(fx)

Java EE Fourni un ensemble d'outils utilisés pour le développement d'applications d'entreprise

Cette version fourni également pléthore de Framework pouvant être interconnecté avec des applications JEE - JSE

- Java DataBase Connector – JDBC
- Java Naming and Directory Interface – JNDI
- Java Persistance API – JPA
- Java Transaction API – JTA
- Hibernate (dont jpa, jta, jdbc,…)
- …

Cette version de JAVA va nous permettre de mettre en place des architectures qui vont faciliter l'utilisation de business logic par – n – applications distantes ou non.

La technologie que nous utiliserons pour ce faire est celle des Enterprises Java Beans

INTRODUCTION - Selon vous…

Essayez de définir les éléments suivants:

- EJB
- "Composants Métiers"»
- Architecture N Tiers
- Application Distribuée – Application Centralisée

A votre avis:

- Dans quel context les EJB peuvent être utilisés?
- De quoi avons-nous besoin pour mettre en place cette technologie?
- Quels sont les différents types d'EJB qui existent, quelles sont leurs particularités et leur cas d'utilisation?

INTRODUCTION - Les EJB

Quelques caractéristiques des EJB:

- Un EJB contient de la logique métier concernant les données.
- Les instances des EJB sont gérées par un container.
- Un EJB peut être customisé au déploiement en modifiant ses variables d'environnement.
- L'EJB peut être configuré par des annotations, ou séparément, dans un fichier XML (deployment descriptor) ➔ On privilégie plus que largement les annotations depuis la spécification 3.0 des EJB

INTRODUCTION - Les EJB

Les Enterprise Java Beans ou EJB sont des composants logiciels coté serveur, non-visuels, qui se doivent de respecter les spécification d'un modèle défini par sun

Le modèle initial défini une architecture à respecter, un environnement d'exécution et un ensemble d'API à exploiter

Cette normalisation permet l'utilisation de ces composants quel que soit l'environnement d'exécution dans lequel ils se situent, pour peut qu'ils n'incluent pas de composants propre à un environnement d'exécution particulier.

La finalité des EJB étant la facilitation de création d'application distribuées.

INTRODUCTION – Application Distribuée

Application Distribuée : Applications dont tous les composants ne se situent pas au même endroit et/ou sur la même machine – Respectent le principe d'Architecture Distribuée – De séparation des ressources

Exemple :

INTERNET – La Toile – tiens son surnom de son architecture sans nœud centrale

Concrètement :

Si par malheurs un ou plusieurs serveurs crashent, explosent ou sont tout simplement tournés off, internet existera toujours. On aura peut être pas accès à une quantité infime de ressources spécifiques mais le web en général fonctionnera toujours…

INTRODUCTION – Application Centralisée

En opposition aux Applications Distribuées nous retrouvons les Applications Centralisées: Ce sont des applications dont tous les composants se situent même endroit et sur la même machine – Respectent le principe d'Architecture Centralisée

Exemple: les infrastructures et applications client/serveur – Un site internet

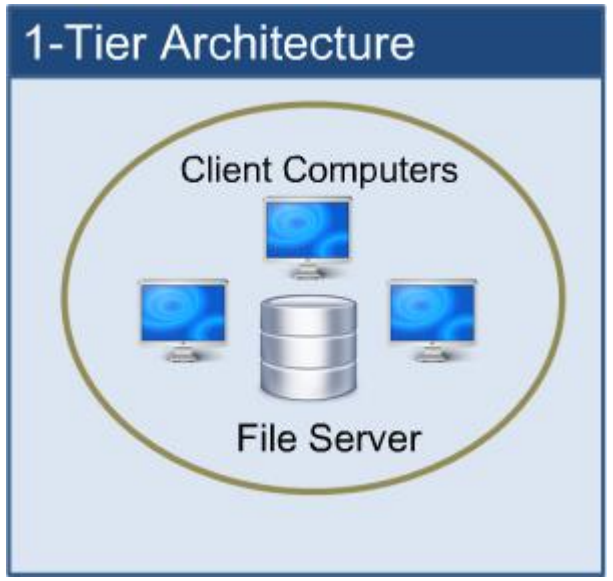
Concrètement: si le serveur est down, nous n'avons plus accès à rien

Ces exemples sont poussées à l'extrême et ne tiennent évidemment pas compte du principe de répartition de charge et donc de la démultiplication des serveurs pour une même et unique application – Virtualisation et Virtualisation de Stockage (RAID)

INTRODUCTION – Architectures

Quelques-unes des différentes architectures d'applications sont les suivantes:

- Architecture 1 Tiers



Exemple: Excel, Word,…

Dans ce type d'architecture, les 3 Tiers principaux que sont:

1° La Business Logic (BL)
2° La Base de Donnée (DB)
3° L'interface utilisateur (UI)

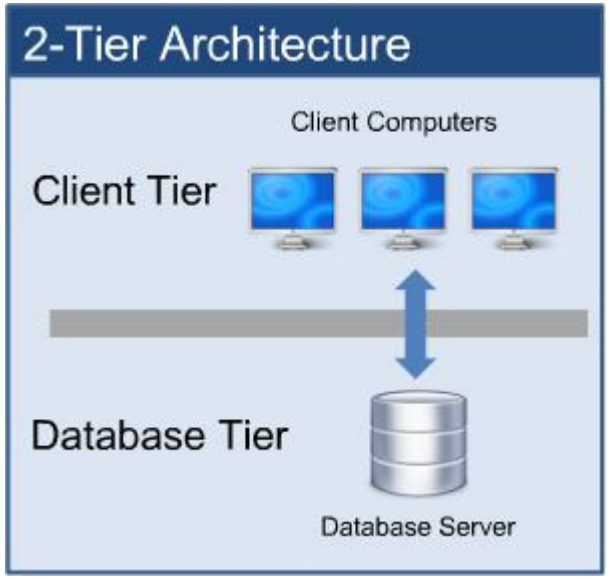
Sont rassemblés au même endroit en un seul package software
Les données sont éventuellement stockées dans un fichier sur la machine du client.
Ce type d'architecture est:

- Simple et pas cher.
- Très vite peu performant en cas de surcharge de données.
- Non évolutif.

Pose généralement problème en cas d'utilisateurs multiple – accès concurrents aux fichiers

INTRODUCTION – Architectures

- Architecture 2 Tiers



L'architecture 2 Tiers est le type d'architecture que vous devez en principe tous connaitre : **architecture client/serveur**

Le client contient l'application et le serveur contient et gère le back-end DB

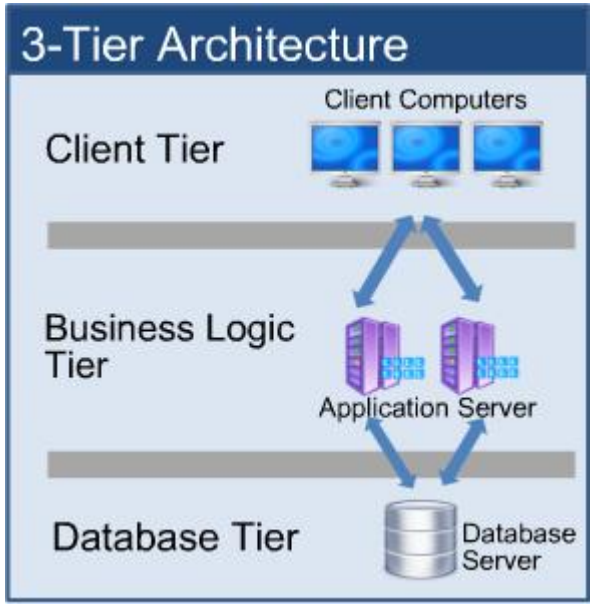
Le client s'occupe donc de l'UI et de la BL tandis que le serveur s'occupe de la DB

Lorsque le client lance l'application, il établi une connexion avec le serveur qui va lui permettre de communiquer avec.

Cette solution permet de fonctionner avec de multiple users qui vont interagir avec la DB. Plus le serveur est puissant, plus il supportera de charge, plus il pourra y avoir des clients qui l'interrogent

INTRODUCTION – Architectures

- Architecture 3 Tiers



L'architecture 3 Tiers ou architecture –n – Tiers est le modèle d'architecture qui apporte une séparation stricte entre les 3 couches UI, BL, DB

Le Tiers Client se chargera de l'affichage
Le Tiers Business (Serveur d'application) s'occupera de la logique Business
Le Tiers Data (Serveur de Bases de Données) s'occupera de la persistance et la restitution de celles-ci

Cette architecture apporte:

- Un allègement du client – Notion de Client léger (Web Techs)
- Une meilleure scalabilité de l'application
- L'hétérogénéité des plateformes utilisées
- Une amélioration de la sécurité des données, le client n'est plus en charge des vérifications et validation d'usage en ce qui concerne l'insertion en DB

INTRODUCTION – Historique

EJB 1.0

- Introduction des Session Beans (Stateless & Stateful) et des Entity Beans.
- Accessibles uniquement via une architecture "remote".

EJB 1.1

- Possibilité d’écrire un "deployment descriptor" en XML plutôt que dans un fichier de classe spécial.

EJB 2.0

- Apparition des interfaces locales qui rend plus efficace les communications avec les EJB.
- Arrivée des Message-Driven Beans.
- Introduction du EJB QL (// SQL)

INTRODUCTION - Historique

EJB 2.1

- Ajout du support des Web Services via un Session Bean

EJB 3.0

- Les EJB deviennent des POJO
- Les interfaces spécifiques aux EJB ne sont plus nécessaires
- Possibilité de définir les EJB via des annotations, et / ou un XML Deployment Descriptor.
- Les Entity Beans sont transférés vers la specification JPA et peuvent dès lors être utilisés hors d'un EJB container. C'est-à-dire qu'ils peuvent aussi être utilisés dans un projet Java SE !

INTRODUCTION - Historique

EJB 3.1

- Apparition des Singleton Session Beans.
- Introduction d'EJBLite.

EJB 3.2

- Amélioration d'EJBLite.
- Simplification des règles concernant les comportements Local et Remote.

INTRODUCTION – EJB Container

S’est dans ce contexte d’architecture –n– Tiers que nous allons développer nos EJB.

Parmi les exigences de la technologie il nous faut encapsuler nos EJB dans un EJB Container au sein du serveur d’application. Cet EJB Container est un cadre au sein du quel il nous est possible de mettre en place des composants distribués.

Tous les serveur n’offrent pas d’EJB Container. Tomcat, que vous avez probablement déjà utilisé, n’en offre pas.

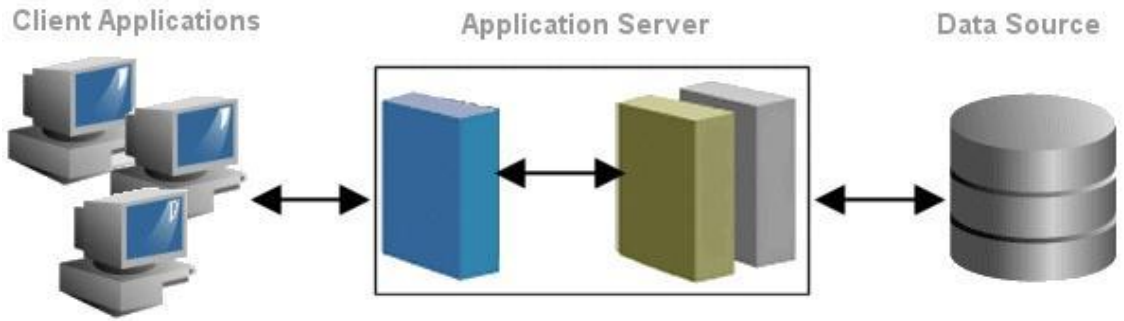
Parmis les serveurs open-source nous retrouvons:

- GlassFish
- Jboss
- Jonas
- WildFly => Jboss specifications

En ce qui nous concerne nous utiliseront une version WildFly 8.X

INTRODUCTION – EJB Container

Dans le cas d'une application Web couplée à l'utilisation des EJB et d'une base de donnée le schéma de communication serait le suivant, en se basant sur un pattern désormais bien connu qu'est le pattern MVC



- Le client se charge de l'affichage
- Le serveur d'application va se charger d'un part, de générer des pages à envoyer au client (web container), ces pages peuvent être alimentés en données dynamique via la business logic fournie par les EJB (ejb container)
- Les données dynamiques sont bien souvent issues de la DB, stockée sur le serveur de base de données

LES DIFFERENTS TYPE D'EJB

LES SESSION BEAN

DIFFERENTS TYPES D'EJB – Session Bean

Les Session Bean sont des objets qui encapsulent la logique métier de l'application.

Ils peuvent être invoqués de manière programmatique par le clients

- Soit par appel de leur nom au moyen du Java Naming and Directory Interface (JNDI)
- Soit au moyen d'injection de dépendance => annotation @EJB

Le client appel les méthodes de l'objet déployé dans l'EJB Container du serveur d'application

Le Session Bean va donc se charger du travail au lieu de laisser cette tâche au client.

Bien qu'un Session Bean n'est pas persistant en tant que tel, il peut par contre faire utilisation de méthodes permettant la persistance de donnée (partie importante de la Business Logic

DIFFERENTS TYPES D'EJB – Session Bean

Le client qu'il soit local ou remote n'accède jamais directement à un Session Bean, il va communiquer avec lui par le biais d'un ou plusieurs interfaces

On retrouve différents types de clients:

- Les clients locaux : le client et le Session Bean se trouvent dans la même JVM, les arguments sont passée par références, les performances sont donc optimales

- Les clients remote (distants): le client et le Session Bean sont dans des JVM différentes, les arguments sont passés par valeur, sérialisées et les communications passent par le réseau ➔ les performances sont affectées

DIFFERENTS TYPES D’EJB – Session Bean

Le client Local:

- Il doit utiliser l’interface @Local du bean
- Il est de ceux qui peuvent retrouver les beans dont il a besoin grâce à l’injection de dépendance:

```
@EJB
private MonBean monBean;
```

DIFFERENTS TYPES D'EJB – Session Bean

Le client Distant:

- Le client distant doit faire un appel JNDI pour retrouver les beans dont il a besoin :

  Context ctx = new InitialContext(jndiProperties);

  MonBeanRemote monBean = (MonBeanRemote) ctx.lookup("…");

- Il utilise l'interface annotée @Remote du bean.

- Tous les objets qui sont passés en argument au bean, ou que ce-dernier retourne doivent implémenter l'interface Serializable.

DIFFERENTS TYPES D'EJB – Session Bean

On retrouve au sein des Session Bean au-delà du caractère Local ou Remote de celui-ci deux types particuliers:

- Les stateless Session Bean, définis comme tel au moyen de l'annotation @Stateless
- Les stateful Session Bean, définis comme tel au moyen de l'annotation @Stateful

- **Stateless** : le container donne la même identité à tous les objets qui référencent l'interface d'un session bean stateless. Cela signifie que l'appel à la méthode equals() renverra true dans tous les cas.

  N'importe quelle instance pourra être utilisée pour n'importe quel client.

- **Stateful** : le container assigne une identité au session object stateful à sa création. Cela signifie que le résultat d'un appel à la méthode equals renverra false dans le cas où les instances sont différentes.

  L'état du bean sera donc maintenu pour un client entre les différents appels.

DIFFERENTS TYPES D'EJB – Session Bean

Le cycle de vie d'un Bean est assez particulier, peut traverser 3 états distincts:

- Existe
- N'existe Pas
- Est en sursis sur le disque

Le cycle de vie va varier en fonction que le bean soit Stateless ou Stateful

A l'instar d'un Objet qui existe tant que le garbage collector n'en a pas fait son repas les EJB on un cycle de vie légèrement différent.

1° Une fois instancié, notre objet existe

2° Nous l'utilisons puis continuons notre chemin

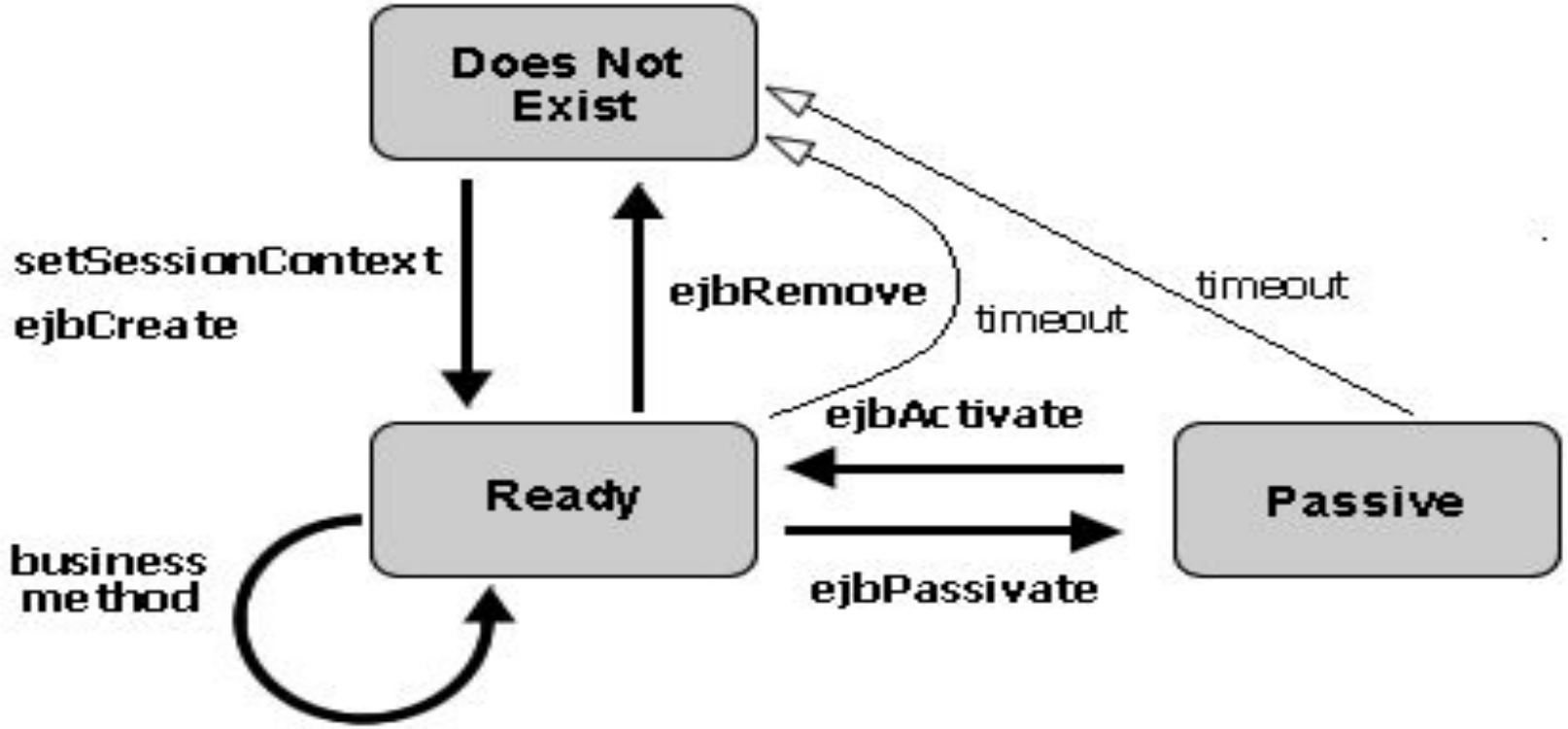
3° Après un certain temps, le Bean va être sauvegarder temporairement sur le disque pour économiser les ressource de l'application, à l'initiative du Server (Passivation)

4° En cas de rappel du bean, cette sauvegarde temporaire est utilisée pour continuer à travailler avec le bean (Activation)

5° Sinon, après un certain temps ou à la demande, celui-ci va être tout simplement détruit: il n'existe plus (Remove)

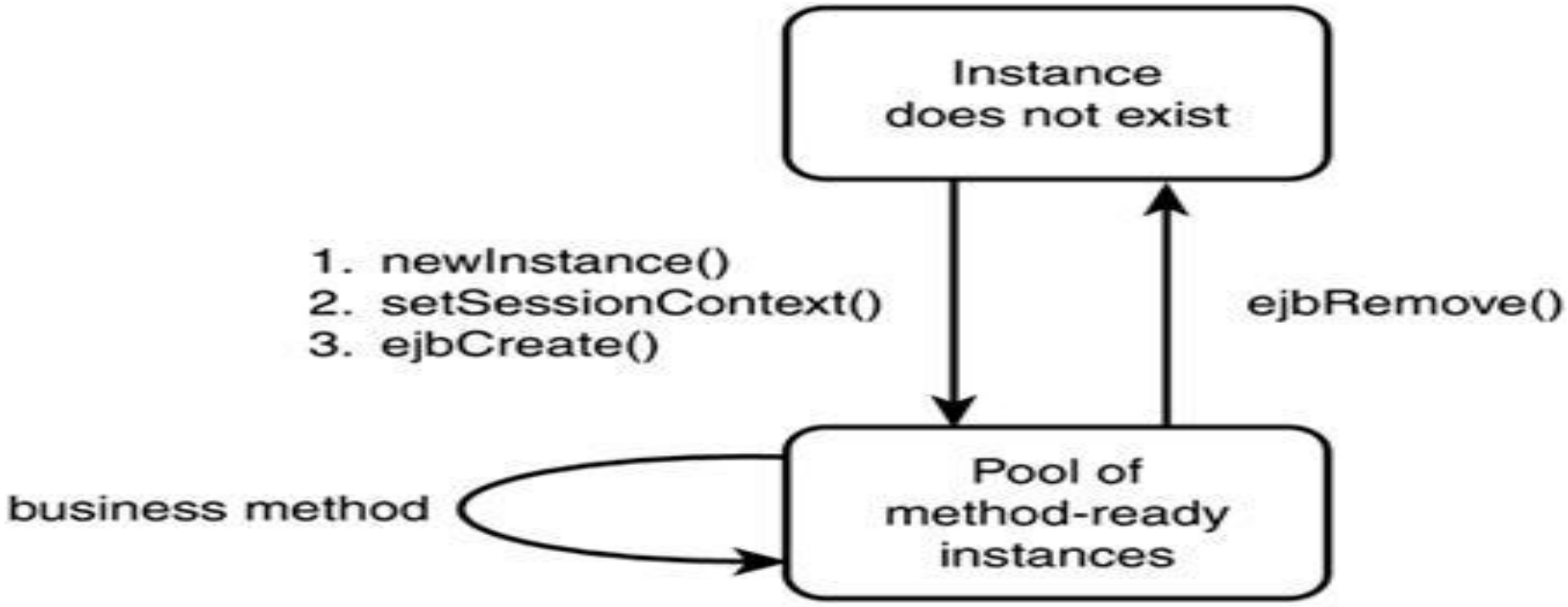
DIFFERENTS TYPES D'EJB – Session Bean

De manière schématique, le cycle de vie d'un Stateful Session Bean ressemble à ceci:



DIFFERENTS TYPES D'EJB – Session Bean

Le cycle de vie d'un Stateless Session Bean ressemble à ceci:



Life Cycle of a Stateless Bean instance

DIFFERENTS TYPES D'EJB – Session Bean

Les Session Bean peuvent gérer des évènements particulier en fonction qu'ils soient Stateless ou Stateful

Ces évènements portent le nom d'Interceptor.

Nous retrouvons parmis ceux-ci, pour les Stateless Session Bean:

- @PostConstruct : qui va être exécuté si défini après toute injection de dépendances et avant tout appel de methode business
- @PreDestroy : qui va être exécuté si défini avant destruction du bean/appel de sa methode Remove() ()

DIFFERENTS TYPES D'EJB – Session Bean

Pour les Stateful Session Bean, nous retrouvons les interceptors suivants:

- @PostContruct
- @PreDestroy
- @PrePassivate : qui va être exécuté si défini avant toute passivation du bean (mise en veille dans le pool du container)
- @PostActivate : qui va être exécuté si défini après toute réactivation du bean (sortie de veille depuis le pool du container, (ré)appel du bean)

Le @PostContruct pourrait par exemple servir à définir des valeurs par défaut pour le Bean lors de son instanciation initiale (@PostContruct et @PreDestroy n'étant appelé qu'UNE fois là ou un contructeur pourra être rappelé lors de l'activation du Bean)

DIFFERENTS TYPES D'EJB – Session Bean

Petit plus en ce qui concerne les Interceptor

Nous pouvons également utiliser deux annotations supplémentaires:

- @Init - qui va spécifier l'appel d'une méthode lors de l'initialisation du Bean
- @Remove qui va spécifier au serveur que le Bean doit être détruit après appel de cette méthode

DIFFERENTS TYPES D’EJB – Session Bean

Il est possible d’implémenter un Web Service via un Session Bean depuis la version 2.1 des EJB

- L’implementation d’un web service via un session bean se fait grâce à la specification JAX-WS.
- On utilise les annotations @WebService et @WebMethod pour configure son service.
- Automatiquement, un "wsdl" sera créé et reprendra toute la configuration du web service, qui pourra être testé, par exemple grâce à SOAP UI.

DIFFERENTS TYPES D'EJB – Session Bean

En plus des Session Bean Stateless et Statefull, un troisème type de Session Bean à vu le jour avec la version 3.1 est EJB, le Singleton Session Bean

Particularité du Singleton:

- Il ne possède qu'une instance unique pour la durée de vie de l'application
- Le container se charge de l'instancier
  - Nous pouvons demander l'instanciation au démarrage au moyen de l'annotation @Startup
- Nous pouvons utiliser plusieurs Singleton au besoin
- On peut ordonner leur instanciation en cas de dépendance entre ceux-ci
- Annotation @DependsOn pour spécifier 4 DependsOn 3 DependsOn 2 DependsOn 1 Startup➔ Orde d'instanciation au démarrage: 1 – 2 – 3 – 4

DIFFERENTS TYPES D'EJB – Session Bean

Etant donné que l'instance du Singleton est unique et partage nous devons absolument gérer l'accès concurrent éventuel aux données

On peut gérer cette concurrence de deux façon:

- De manière CMC – Container-Managed Concurrency: par défaut – Conseillé – les méthodes du Singleton sont verrouillées lors de l'écriture (fichiers, db,…) ➔ LockType.Write
  - Ce Comportement peut être modifier au moyen de l'annotation @Lock
- De manière BMC – Bean-Managed Concurrency: le développeur doit se charger de gérer la concurrence des accès – Dangereux – par exemple au moyen du modificateur synchronized

En cas de besoin, le choix de la gestion de la concurrence se fait grâce à l'annotation @ConcurrencyManagement()

LES ENTITY BEAN

DIFFERENTS TYPES D'EJB – Entity Bean

Un EJB de type Entity Bean est un objet qui possède les caractéristiques suivantes:

- Il fait partie intégrante du modèle d'application et représente une vue objet des données stockées en base de données
- Il peut avoir une longue durée de vie (généralement au moins aussi longue que les données en DB)
- Il est, depuis EJB 3.0 devenut un POJO's (Plein Old Java Object ⇔ Bon vieux objet java) et peut donc facilement être utilisé dans le cas d'application Java SE (Hors JVM serveur d'application ➔ Remote)
- Il est, depuis EJB 3.0 , managé par le Java Persistence API (JPA) et réponds donc à ses normes

DIFFERENTS TYPES D'EJB – Entity Bean

Les Entity Beans sont configurés grâce aux annotations en principe désormais connues:

- @Entity sur la classe de l'EJB
- Eventuellement l'annotation @Table pour nommer la table au moyen de l'attribut – *name* – afin de spécifier le nom de la table en DB, ou d'éventuelles contraintes
- @Id sur le champ qui servira d'identifiant (
- En complément d'@Id vous pouvez également utiliser l'annotation @GeneratedValue avec l'attribut – *strategy* – afin de définir le type de génération d'Id (manuel, auto-incrément,…)
- Eventuellement @Column sur les champs avec l'attribut – *name* – afin de spécifier le nom de la colonne en DB

DIFFERENTS TYPES D'EJB – Entity Bean

Autres annotations utiles :

- @Transient : spécifie qu'un champs ne doit pas être persistant
- @Basic : grâce à l'attribut fetch, permet de spécifier le comportement du Fetch – L'attribut fetch peut être utilisé dans d'autres annotations
- @Temporal : spécifie le type de champs temporel à sauvegarder en db
- @Enumerated : permet d'utiliser une énumération comme contraintes de valeurs pour un champ
- @Lob : spécifie qu'un champs doit être sauvegardé dans un Blob / Clob en db

DIFFERENTS TYPES D'EJB – Entity Bean

Annotations des relations

- @OneToOne, nécéssite la présence d'un objet Y dans X et X dans Y
- @OneToMany, nécessite la présence d'objet Y dans X et d'une List<X> dans Y
- @ManyToOne, nécessite la présence d'une List<Y> dans X et d'un objet X dans Y
- @ManyToMany, nécessite la présence d'une List<Y> dans X et d'une List<X> dans Y

En spécifiant une relation, on peut spécifier un attribut *cascade* :

- Persist, Merge, Remove, Refresh, All

LES MESSAGES-DRIVEN BEAN

DIFFERENTS TYPES D'EJB – MDB

Les MDB ou Message-Driven Bean sont des objets qui possèdent les caractéristiques suivantes:

- Il s'exécute à la réception d'un message.
- Il peut être appelé de façon asynchrone.
- Il peut être conscient des transactions.
- Il peut mettre à jour des données dans une base de données.
- Il ne représente pas directement ces données, même s'il peut y avoir accès.
- Il a une courte durée de vie.

DIFFERENTS TYPES D'EJB – MDB

Il existe deux modèles de communication :

- Le modèle Point-to-Point (P2P)
- Le modèle publish-subscribe (pub-sub)

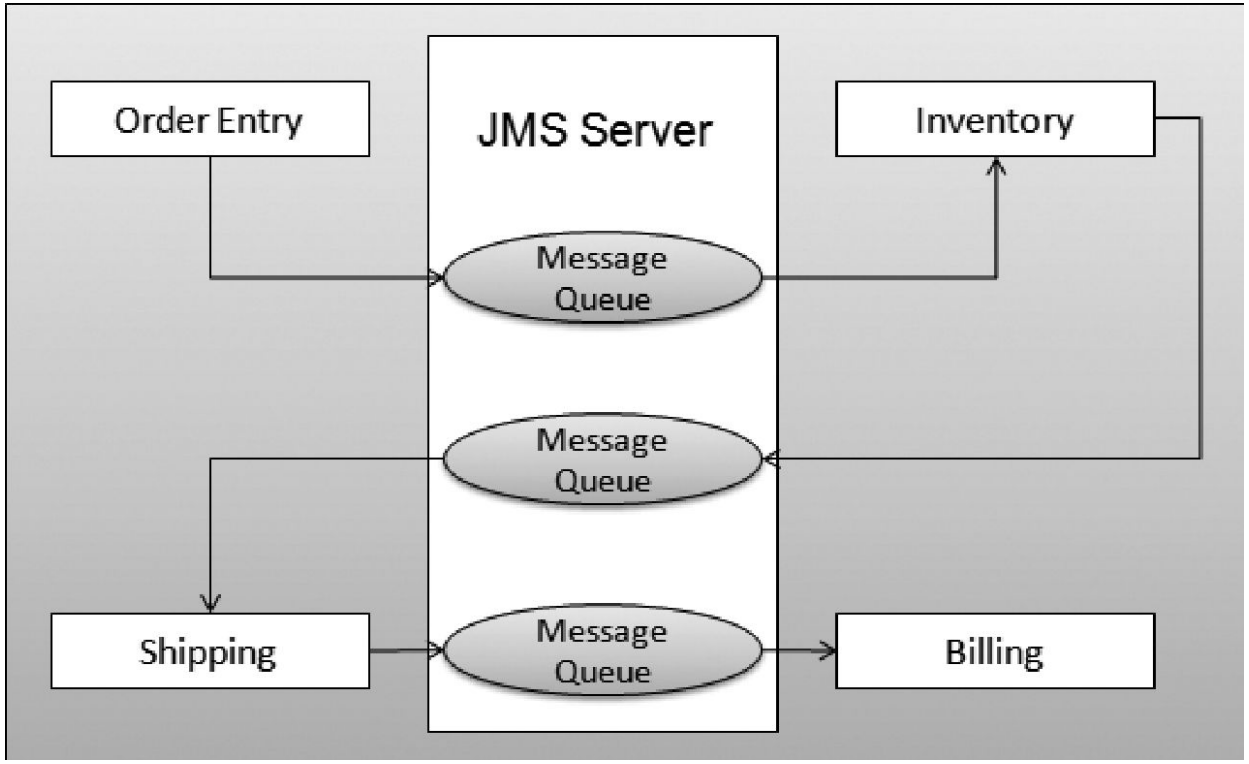
Les Message-Driven Beans utilisent JMS (Java Message Service).

L'API JMS est une API qui permet d'échanger des messages asynchrones. Il en existe plusieurs implémentations.

Les types de messages qui peuvent être échangés sont les suivants : ByteMessage, MapMessage, ObjectMessage, StreamMessage et TextMessage.

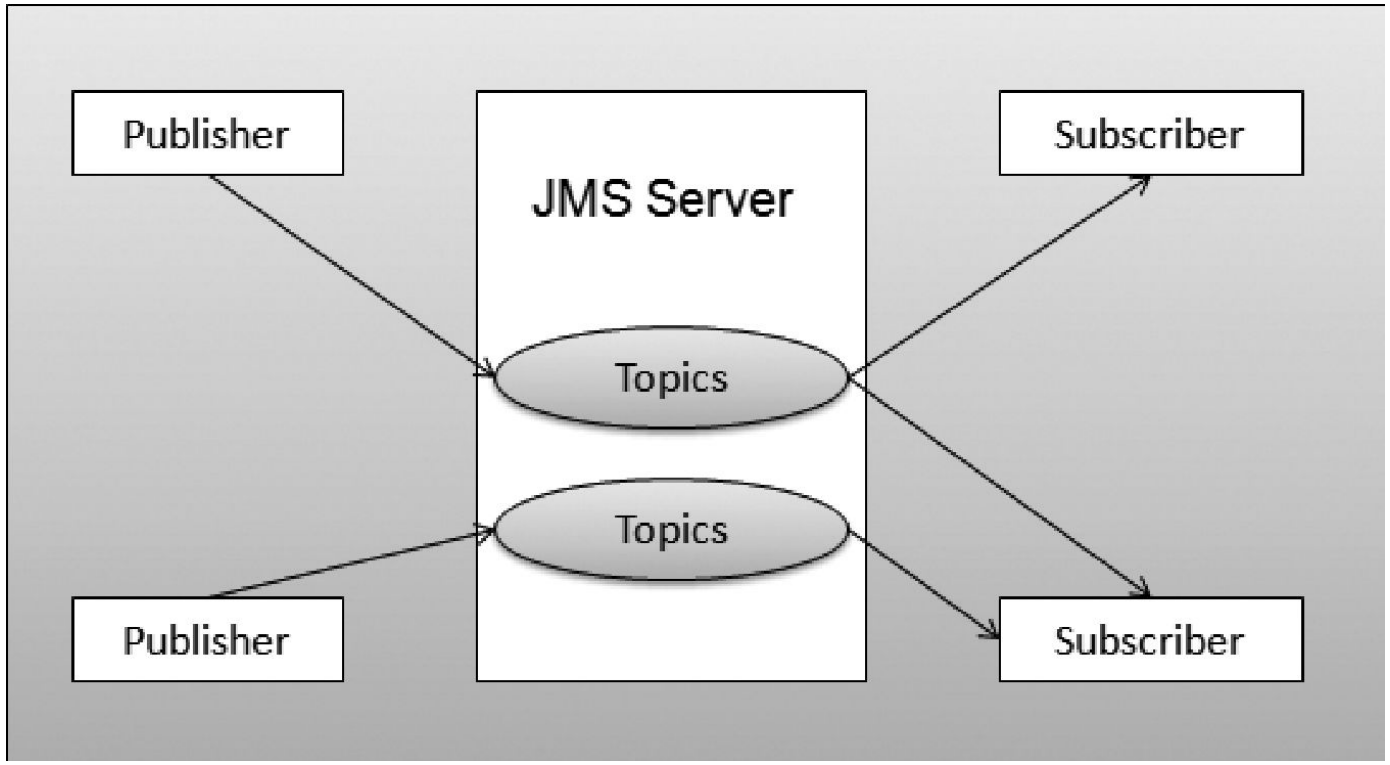
DIFFERENTS TYPES D'EJB – MDB

- Le modèle Point-to-Point définit un émetteur et un destinataire :



DIFFERENTS TYPES D'EJB – MDB

- Le modèle publish-subscribe définit un émetteur, et plusieurs destinataires :



DIFFERENTS TYPES D'EJB – MDB

Pour utiliser les Message-Driven Beans dans Wildfly, il faut soit configurer le standalone.xml en conséquence, ou utiliser le fichier standalone-full.xml

- Les messages aux MDB sont envoyés grâce à une démarche spécifique :

```
@Resource(mappedName = "java:/jms/queue/QueueName")
private Queue demoMessages;

@Resource(mappedName = "java:jboss/DefaultJMSConnectionFactory")
private ConnectionFactory demoMessagesFactory;
```

DIFFERENTS TYPES D'EJB – MDB

- Envoi d'un message:

```
Connection connection = demoMessagesFactory.createConnection();

Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);

MessageProducer producer = session.createProducer(demoMessages);

TextMessage msg = session.createTextMessage("My own message");

msg.setStringProperty("Key", "Value");

producer.send(msg);
```

DIFFERENTS TYPES D'EJB – MDB

Réception des messages:

- Le MDB doit être annoté avec l'annotation *@MessageDriven*, et spécifier les attributs *activationConfig* et *mappedName*.
- L'attribut *activationConfig* doit lui-même posséder les attributs *@ActivationConfigProperty* nécessaires, afin de spécifier la *destination* ainsi que la *destinationType.*
- Dans le MDB, on doit implémenter la méthode *onMessage(Message msg)*

DIFFERENTS TYPES D'EJB – MDB

- Exemple d'annotation du MDB :

```
@MessageDriven(
activationConfig = {
        @ActivationConfigProperty(
                propertyName = "destination",
                propertyValue = "java:/jms/queue/ExpiryQueue"),
        @ActivationConfigProperty(
                propertyName = "destinationType",
                propertyValue = "javax.jms.Queue")

},
mappedName = "ExpiryQueue")
```

DIFFERENTS TYPES D'EJB – MDB

- Exemple de méthode *onMessage(Message message)*

```
public void onMessage(Message message) {

        TextMessage tm = (TextMessage) message;

        try {
                System.out.println(tm.getStringProperty("Key"));
        } catch (JMSException e) {
                e.printStackTrace();
        }
}
```

JPQL – JPA pour exécuter ses requêtes

JPQL – JPA pour exécuter ses requêtes

En premier lieu, fini l'utilisation de notre ManagerFactory

Fini le EntityManagerFactory emf = Persistence.CreateEntityManagerFactory(""PU"»),

Fini le EntityManager em = emf.CreateEntityManager();

JTA (Java Transaction API va s'occuper de gérer les instanciation/libération de l'Entity Manager pour nous)

Tout ce dont il a besoin c'est d'UNE annotation (et oui!)

Il s'agit tout s'mplement d'une injection de dépendance au sein de notre Bean:

```
@PersistenceContext(unitName="""»)
private EntityManager em;
```

JPQL – JPA pour exécuter ses requêtes

Vous avez tous fait du SQL?

Comment récupérer tous les champs de la Table User?

*SELECT * FROM User*

En Java Persistence Query Langage (JPQL) nous allons tout simplement fonctionner non plus avec des Tables et des Colonnes, mais bien avec des Entités et des Attributs

SELECT u FROM User u

JPQL – JPA pour exécuter ses requêtes

Vous avez différentes manières d'effectuer vos requêtes:

- Via les NamedQueries au sein de votre Entité au moyen de l'annotation @NamedQuery

Vous pourrez donc appelez ces NamedQueries au moyen de Em.CreateNamedQuery("nom de votre query", classDeReference).

Vous pouvez faire passez les éventuels paramètres requis au moyen de la méthode setParameter("nomDuParametre",valeur)

Exemple: @NamedQuery(name="findAll", query="SELECT u FROM User u")

Em.CreateNamedQuery("findAll", User.class).getResultList();

JPQL – JPA pour exécuter ses requêtes

Exemple:

```
@NamedQueries({

@NamedQuery(name="getOneByUserName", query="SELECT u FROM User u WHERE u.UserName=:uname")

})
```

```
Em.CreateNamedQuery("getOneByUserName", User.class).setParameter("uname","Georges").setMaxResult(1).getSimpleResult();
```

➔ Va vous retourner un Objet User dont le Username est 'Georges'

JPQL – JPA pour exécuter ses requêtes

- Via le CreateQuery()

Exemple:

```
Em.CreateQuery("SELECT u FROM User u",User.class).getResultList();
```

➔ Va vous retourner une List<User>

JPQL – JPA pour exécuter ses requêtes

Différentes opérations sont effectuées lorsque nous travaillons avec une Base de données:

- Les Insert
- Les Update
- Les Delete
- Les Select

Pour les select, pas de secrets, on passe par les NamedQueries ou le CreateQuery

Pour l'insert, petit mnémotechnique:

- Utilisation d'un API de PERSITANCE,
- Nous cherchons donc à faire PERSISTER les données en DB
- Nous allons utiliser la méthode *persist()* de l'Entity Manager, en lui passant l'objet à faire persister

/!\ Pas n'importe quel objet, of course, une Entité /!\

JPQL – JPA pour exécuter ses requêtes

Pour le delete:

- Il est impératif qu'une suppression se face dans un contexte transactionnel
- ➔ em.getTransaction().Begin() ➔ FINI ☺ merci JTA
- Ensuite: methode *remove()* de l'Entity Manager ➔ Em.remove(monObjet)
- Qui dit début de transaction dit également fin de transaction
  - Fini le Commit en cas de succès em.getTransaction().commit();
  - Fini RollBack en cas d'erreur em.getTransaction().rollBack();
  - ➔ JTA s'en occupe pour nous également!!

JPQL – JPA pour exécuter ses requêtes

Pour l'update:

- Il est impératif qu'une mise à jour se face dans un contexte transactionnel
- ➔ JTA s'en occupe pour nous
- monObjet.setField*(value)* sur autant de champs que désiré ou nécessaire
- Utilisation de méthode *merge()* de l'Entity Manager ➔ em.merge(monObjet)
- Qui dit début de transaction dit également fin de transaction
  - Fini le Commit en cas de succès em.getTransaction().commit();
  - Fini RollBack en cas d'erreur em.getTransaction().rollBack();
  - ➔ JTA s'en occupe pour nous également!!

LES TRANSACTIONS

LES TRANSACTIONS

- Qu'est ce qu'une transaction?

Un transaction est un contexte dans lequel un ou une succession d'opérations vont s'effectuer pour faire transiter un application d'un état A complet vers un état B complet.

- En cas d'échec d'une des opérations inclue dans une transaction, le système conserve son état initial.
- En cas de réussite de toutes les opérations inclues dans la transaction, le système valide l'ensemble des opérations effectuées

Les propriétés des transactions sont supposées connues: Atomicité, Consistance, Isolation, Durabilité

LES TRANSACTIONS

En ce qui concerne l'utilisation des transactions dans un context EJB, deux options s'offrent à nous:

- Laisser le Container gérer les Transactions pour nous => Contrainer-Managed Transaction (CMT)
- Laisser le Developpeur s'occuper de la gestion des Transactions => Bean-Managed Transaction (BMT)

Par défaut, les EJB fonctionnent en CMT

LES TRANSACTIONS

En cas de Contrainer-Managed Transaction, nous pouvons utiliser l'annotation @TransactionAttribute sur une, plusieurs ou toutes nos méthodes afin de spécifier le comportement souhaité de la part du contrainer.

Nous retrouvons l'énumeration suivante des attributs:

- Mandatory
- Required
- Requires_New
- Supports
- Not_Supported
- Never

LES TRANSACTIONS

- Mandatory

Ne s'applique que sur les Session Bean

Une méthode avec un TransactionType.Mandatory garantie que cette méthode sera toujours exécuté dans une transaction.

Cependant, c'est le client qui a la charge de démarrer la transaction.

Si le client essaye d'invoquer la méthode sans ayant préalablement démarré une transaction, le container bloquera l'appel et lèvera une exception

LES TRANSACTIONS

- Required

S'applique aux Session Bean et Message-Driven Bean

Une méthode avec un TransactionType.Required garantie que cette dernière sera toujours exécuté au sein d'une transaction.

Si le client essaye d'invoquer la méthode en dehors d'une transaction, le container démarre une nouvelle transaction, exécute l'appel et commit la transaction

/!\ Différence entre un type Mandatory et Required – L'un lève une exception, l'autre se charge de démarrer une nouvelle transaction.

LES TRANSACTIONS

- Requires_New

Ne s'applique que sur les Session Bean

Une méthode avec un TransactionType,Requires_New garanti que cette méthode sera toujours exécuté au sein d'une transaction

Si le client invoque la méthode en dehors ou au sein d'une transaction, cette transaction sera mise en suspend, une nouvelle transaction sera créée par le container, la méthode sera exécutée, la transaction sera commit puis la prochaine transaction sera reprise.

LES TRANSACTIONS

- Supports

Ne s'applique que sur les Session Bean

Une méthode avec un TransactionType.Supports garanti que cette méthode sera invoquée dans le respect de l'état transactionnel du client

Si le client à une transaction en cours la méthode sera invoquée dans le contexte de la transaction, sinon la méthode sera tout simplement invoquée.

Le container ne va rien faire pour changer cet état

LES TRANSACTIONS

- Not_Supported

S'applique aux Session Bean et Message-Driven Bean

Une méthode avec un TransactionType.Not_Supported garanti que la méthode sera toujours exécuté en dehors d'une transaction

Si le client a une transaction en cours, celle-ci sera suspendue, la méthode sera invoquée puis la transaction reprendra son cours.

LES TRANSACTIONS

- Never

Ne s'applique que sur les Session Bean

Une méthode avec un TransactionType.Never garanti que la méthode ne sera exécutée qu'exclusivement en l'absence de toute transaction

Si une transaction est en cours au moment de l'invocation de la méthode, une exception sera levée

TransactionType.Never est l'opposé total de TransactionType.Mandatory

LES TRANSACTIONS

Toujours en cas de CMT

- On peut obtenir un EJBContext via injection de dépendance, ou via lookup JNDI afin de savoir si une transaction attend un callback (getRollbackOnly()) pour lancer soi-même un rollback (setRollbackOnly()), spécialement dans le cas où une exception s'est produite.

- La configuration des transactions appliqué à une méthode ainsi que le contexte transactionnel dans lequel cette méthode est appelée sera appliqué à toutes méthodes que cette méthode appellera

- Lorsqu'une nouvelle transaction est crée pour une méthode, elle subit un commit implicite lorsque la méthode retourne vers le code appelant

LES TRANSACTIONS

En cas de Bean-Managed Transaction:

- Dans le cas où le développeur souhaite plus de flexibilité, et configurer lui-même chaque étape de sa transaction, il peut utiliser les Bean-Managed Transaction.

- Pour cela, on commence par obtenir un SessionContext via injection :

```
@Resource
SessionContext sessionContext;
```

- On acquiert ensuite une UserTransaction

```
UserTransaction utx = sessionContext.getUserTransaction();
```

LES TRANSACTIONS

- Attention ! C’est le rôle du développeur de s’assurer qu’une Bean-Managed Transaction se déroule correctement et est menée à bien.

- C’est donc un outil puissant, mais à utiliser seulement lorsque c’est nécessaire.

LES INTERCEPTEURS

LES INTERCEPTEURS

Comme nous l'avons vu, l'appel de méthode peut s'effectuer après un passage par des méthodes spécifiques en lien direct avec des annotations spécifiques

- @PostContruct
- @PreDestroy
- @PrePassivate
- @PostActivate

Ces méthodes sont en fait des interceptors accessibles à un instant T de la vie de notre EJB

Nous pouvons déclarer nos propres interceptors si nous le souhaitons, afin d'effectuer des vérifications sur le contexte d'une méthode, avant l'appel de celle-ci ou sont éventuel annulation

LES INTERCEPTEURS

Un interceptor est donc:

- Une méthode qui intercepte l'appel d'une méthode métier.
- Appelé automatiquement, juste avant que la méthode du bean soit appelé.
- Défini grâce à des annotations, dans une classe bean ou dans une classe à part.
- Ils peuvent être appliqués sur des session beans ou des message-driven beans.

LES INTERCEPTEURS

Les buts des Interceptors :

- Avoir le contrôle sur le déroulement d'une méthode.
- Analyser et manipuler des paramètres de façon élégante afin, éventuellement, d'autoriser ou d'interdire l'exécution d'une méthode.
- Gérer certains aspects de façon plus claire (logging, sécurité,…).

LES INTERCEPTEURS

Comment mettre ne place un interceptor custom:

- Créer un classe
- Définir une méthode, précédée de l'annotation @AroundInvoke qui correspond au pattern suivant:

[modificateur] Object [nom] (InvocationContext ctx) throwsException { }

- Pour valider le passage par l'interceptor et donc exécuter la méthode, il faut impérativement retourner la méthode *proceed()* de l'InvocationContext
- Sinon, retourner null

/!\ En cas d'héritage entre classes Interceptor, ceux-ci seront appelés dans l'ordre Parent ➔ Enfant

LA SECURITE

LA SECURITE

La sécurité des EJB n'aura certainement rien de compliqué pour les amateurs de framework, que ce soit en JAVA ou dans d'autres langages.

Elle se fait, sur les classes, au sein des EJB, par le biais d'annotations

Quant à l'accès aux EJB, celui-ci se fait au moyen d'une identification Username – Password, défini sur le serveur par le System Administrator

LA SECURITE

Les différentes annotations exploitables pour la sécurité sont les suivantes:

- @DeclareRoles
- @RolesAllowed
- @PermitAll
- @DenyAll
- @RunAs

LA SECURITE

- @DeclareRoles

L'annotation est utilisée afin de déclarer les différents rôles qui seront acceptés au sein du bean, l'annotation doit être utiliser en tête de classe

- @RolesAllowed

L'annotation est utilisée en en-tête d'une méthode du bean ou sur le bean lui-même afin de spécifier les rôles qui y auront accès

- @PermitAll

L'annotation est utilisée afin de signifier que tous les rôles ont accès à la méthode ou au bean annoté

LA SECURITE

- @DenyAll

L'annotation est utilisée quant à elle afin de signifier que aucun rôle quel qu'il soit n'a accès à la méthode ou au bean annoté

- @RunAs

L'annotation est utilisée sur la classe du bean afin de spécifier le rôle avec lequel la classe sera traitée, quel que soit le rôle du client

En l'absence de définition d'utilisateur, celui-ci est 'anonymous'

Les Timer

LES TIMER

Principe de Cron, Tâches Planifiées

Les Timer sont une facilités pour les threads d'effectuer des taches uniques, programmées ou automatique, de manière asynchrone et en background du thread principal de l'application

Il peuvent se mettre en place de deux manières différentes:

- De manière automatique
- De manière programmatique

LES TIMER

Le Timer automatique va se faire des la manière suivante:

- Au moyen d'un singleton (exemple)

En annotant la méthode à lancer de l'annotation @Schedule(), en y précisant la récurrence de l'opération s'il y lieu.

L'avantage d'un singleton est qu'il peut être instancié on startup et qu'a ce moment la et pour tout la durée de vie de l'application, des tâches peuvent être exécutée en arrière plan

Nous pouvons sans problème fonctionner avec des injections de dépendance pour travailler avec d'autres EJB au sein de la méthode planifiée, sachez néanmoins que cette dernière ne peut que renvoyer *void*

LES TIMER

Exemple:

```
@Singleton
@Startup

class MonTimer {
 public MonTimer() {   }

 @Schedule(second="/10", minute="*", hour="*")
 private void SayHi() {
            System.out.println("Hello From the Timer");
 }

}
```

LES TIMER

Le Timer programmatique demande quoi qu'il arrive de créer un objet Timer qui exécutera une méthode annotée @TimeOut une fois le décompte effectué.

Pour créer le timer il nous faut:

- Injecter le TimerService

  @Resource

  private TimerService TS;

- Mettre en place la méthode qui va lancer le timer

Cette méthode pourra-être appelée une fois toutes les injections de dépendances effectuées (dans @PostContruct) ou pourquoi pas directement par le client

LES TIMER

Il y a différents possibilités de configurer les Timer:

- createCalendarTimer
- createIntervalTimer
- createTimer
- createSingleActionTimer

Certaines de ces méthodes prennent un double en paramètre (nb de millisecondes avant /entre les appels), d'autre pendront une ScheduleExpression permettant de paramètrer de manière Jour/Mois/Année Heure:Minutes:Secondes – récurrence, Jours de la semaine,…

Pour plus d'info sur les ScheduleExpression :

http://docs.oracle.com/javaee/6/api/javax/ejb/ScheduleExpression.html

LES TIMER

Exemple de Timer Programmatique

- Au moyen d'un Session Bean (exemple)

```
class MonBean(){

@Resource
private TimerService TS;


public void CreateTimer(){
TS.createTimer(5000, 5000, "Le Timer sur mon Session Bean");
}
```

LES TIMER

```
@Timeout
   private void TimedOut(Timer timer){
   System.out.println("-------- TIK TOK ON THE CLOCK -------- ");
   System.out.println("Debut d'exécution => " + timer.getInfo());
   System.out.println("Prochaine exécution => " + timer.getNextTimeout());
   System.out.println("Compteur avant exécution => " + timer.getTimeRemaining());
   System.out.println("---------------------------------- \n");
}
```

EXERCICES

Exercice 1.1: Sans vous aider de la théorie, définissez:

- Ejb
- EjbContainer
- Application distribuée
- Application 1 Tiers
- Application 2 Tiers
- Application n Tiers

Temps – 15 minutes

EXERCICES

Exercice 1.2: Quels sont les principaux avantages de l'utilisation des EJB?

Temps – 5 minutes

Exercice 1.3: Pourquoi Tomcat ne peut pas être utilisé pour développer des EJB?

Temps – 5 minutes

Exercice 1.4: Quels sont les différences fondamentales entre les architectures 1 Tiers, 2 Tiers et N Tiers?

Temps – 10 minutes

EXERCICES

Exercice 2.1: Créez un EJB Project avec Projet EAR ainsi qu'un Dynamic Web Project et un Projet Java que vous ajouterez à cet EAR

Créez un Session Bean Stateless que vous appelez Calculatrice et son interface qui défini les méthodes suivantes:

Calculatrice + CalculatriceRemote

public double operation(int num1, int num2, String operator);

Calculatrice

public double sum(int num1, int num2)

public double min(int num1, int num2)

public double multi(int num1, int num2)

public double div(int num1, int num2)

public double mod(int num1, int num2)

Temps – 15 minutes

EXERCICES

Exercice 2.2: Créez un Singleton instancié au démarrage du serveur d'application qui contiendra les méthodes: direBonjour() et direAuRevoir()

Dans votre WebApp:

- créez un page avec un formulaire et trois champs
  - Operande1
  - Operande2
  - Operation

Récupérez le résultat de l'opération grâce au Bean Calculatrice

Temps – 15 minutes

Exercice 2.3: Faites de même avec le Projet Java en utilisant le Scanner

Astuces:

- une partie de l'opération peut se faire par injection de dépendance, l'autre par référence JNDI
- Les références JNDI vous sont fournies au démarrage du serveur. Elles répondent généralement au pattern [EAR-Serveur Application]/[EJB Project]/[EJB name]![Full Path to EJB Interface]

Temps – 15 minutes

EXERCICES

Exercice 3.1: Créer deux entités

- Une entité User {Username (ID), FirstName, LastName}
- Une entité Profile(IdProfile (ID), Street, Number, City, Country, ZipCode, NickName)

- La relation entre les entités est de type One to One, tenez en compte lors de la création ;)
- L'IdProfile est autoGénéré

Temps – 10 minutes

EXERCICES

Exercice 3.2:

Créer les tables correspondant aux entités créées en base de donnée

Temps – 5 minutes

EXERCICES

Exercice 3.3:

Créer un session bean qui va:

- Permettre d'ajouter certaines valeurs dans les deux tables créées
- Vider la db
- Sélectionner toutes les données
- Sélectionner une donnée sur base d'un Username
- Sélectionner des données sur base d'une ville et d'un code postal

Au sein d'une servlet, tester la BL de notre nouveau bean et faites un affichage de chacun de vos résultSet

Temps – 30 minutes

EXERCICES

Exercice 4.1 : Créer un intercepteur qui va s'assurer que les paramètres passés à la méthode invoquées contiennent une valeur.

Dans le cas ou au moins une valeur est vide, rejetez l'appel de la méthode, retournez null à l'application

/!\ Pensez à gérer les exceptions

Temps – 15 minutes