Démarrer avec JPA/Hibernate

Architecture Hibernate / JPA

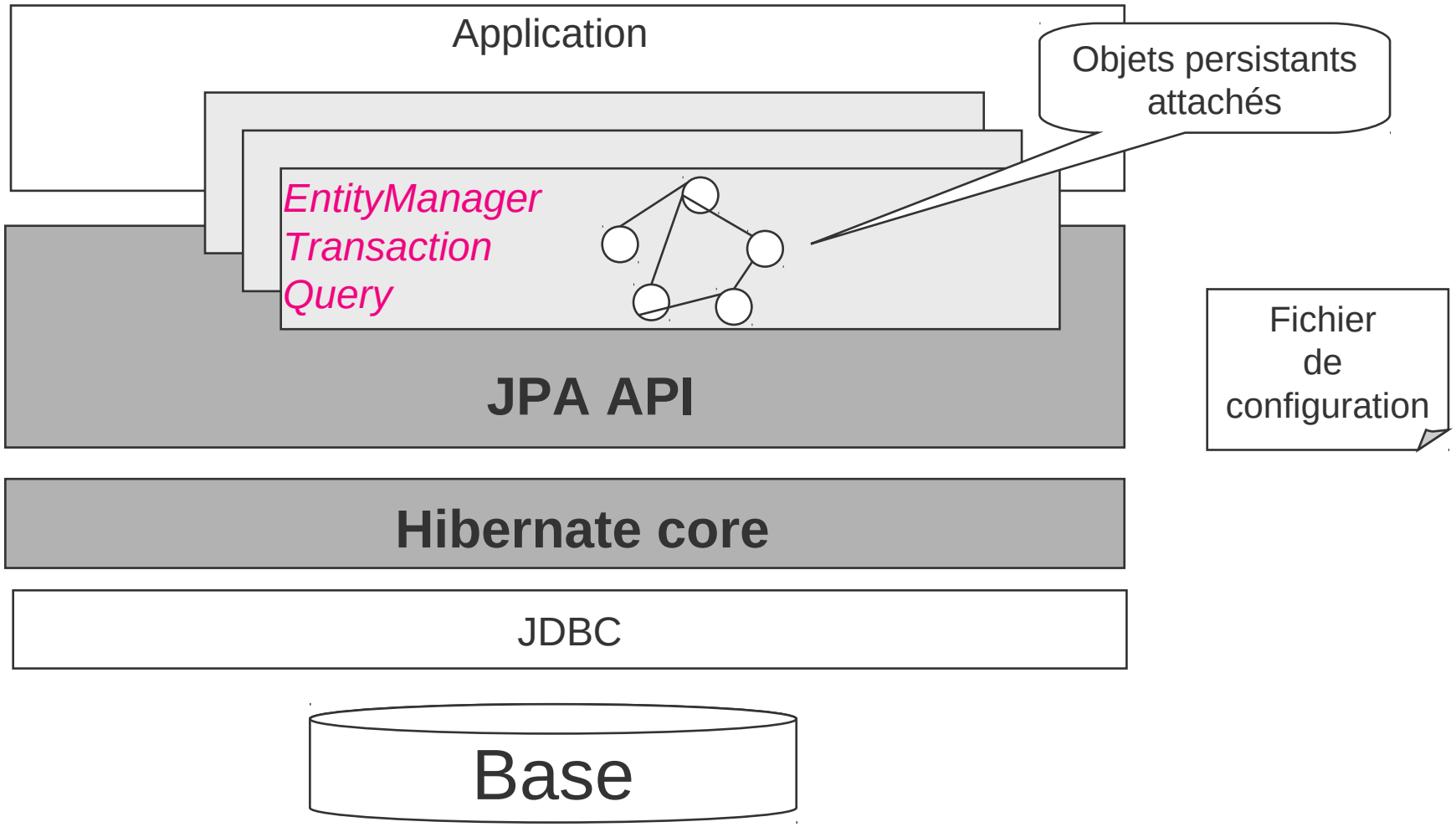
Le fichier *persistence.xml*

Annotations indispensables pour les entités

API EntityManager

Exemple

Architecture Hibernate
Présentation



Correspondance Hibernate / JPA

Nom	JPA	Hibernate	Description
Fabrique de gestionnaire de persistance	***EntityManagerFactory***	*SessionFactory*	Encapsule la configuration à un support de persistance (BD) et permet d'instancier des *EntityManager*
Gestionnaire de persistance	***EntityManager***	*Session*	API Permettant d'effectuer des opérations CRUD sur les classes persistantes
Contexte de persistance			Un ensemble de classe persistante. A l'intérieur d'un contexte de persistance, un seul gestionnaire de persistance gère les classes entités.
Unité de persistance	***persistence.xml***	*hibernate.cfg.xml*	L'ensemble des types d'entités pouvant être gérées par un gestionnaire de persistance et donc mappé vers la même base de données.

Démarrer avec JPA/Hibernate

Architecture Hibernate / JPA

Le fichier *persistence.xml*

Annotations indispensables pour les entités

API *EntityManager*

Exemple

Unité de persistance

- Représente un ensemble d'entités mappés vers une base de données
- Défini à l'intérieur de ***META-INF/persistence.xml***
- Il spécifie :
  - Les moyens de se connecter à la base : *JDBC* ou *DataSource JNDI*
  - Le type de transaction à utiliser pour les opérations de persistance : *RESOURCE_LOCAL* ou *JTA*
  - L'implémentation de JPA
  - Des propriétés spécifiques à l'implémentation

le fichier *persistence.xml*

- Le fichier *persistence.xml* définit une ou plusieurs unité de persistance

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence>
<persistence-unit name="entreprise">
.
.
.
</persistence-unit>
<persistence-unit name="personnel">
.
.
.
</persistence-unit>
</persistence>
```

le fichier *persistence.xml* Java SE

```
<?xml version="1.0" encoding="UTF-8"?>


<persistence version="2.0" >
<!-- nom donné à l'unité de persistance -->
<persistence-unit name="entreprise" transaction-type="RESOURCE_LOCAL">
<!-- Implémentation -->
<provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>
<!-- fichier XML optionnel de mapping (par défaut orm.xml) -->
<mapping-file>orment.xml</mapping-file>
<!-- propriétés spécifiques à l'implémentation JPA -->
<properties>
<property name="javax.persistence.jdbc.driver" value="org.gjt.mm.mysql.Driver"/>
<property name="javax.persistence.jdbc.url" value="jdbc:mysql://localhost/ssii"/>
<property name="javax.persistence.jdbc.user" value="dbUser"/>
<property name="javax.persistence.jdbc.password" value="secret"/>
<property name="hibernate.hbm2ddl.auto" value="create-drop"/>
</properties>
</persistence-unit>
</persistence>
```

le fichier *persistence.xml* Java EE

```
<?xml version="1.0" encoding="UTF-8"?>


<persistence version="2.0">
<!-- nom donné à l'unité de persistance -->
<persistence-unit name="entreprise" transaction-type="JTA">
<!-- Implémentation -->
<provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>
<!-- nom JNDI de la DataSource -->
<jta-data-source>java:/DefaultDS</jta-data-source>
<!-- fichier XML optionnel de mapping (par défaut orm.xml) -->
<mapping-file>orment.xml</mapping-file>
<!-- propriétés spécifiques à l'implémentation JPA -->
<properties>
   <property name="hibernate.hbm2ddl.auto" value="create-drop"/>
</properties>
</persistence-unit>
</persistence>
```

Attributs et éléments standards

- ***name*** Toute unité de persistance doit avoir un nom.
- ***transaction-type*** : *JTA* (défaut dans environnement Java EE) ou RESOURCE_*LOCAL* (défaut dans un environnement Java SE) . => En général pas besoin de le préciser
- ***provider*** : Implémentation de la persistance
  => Pas besoin de le préciser sauf si il y a plusieurs implémentations dans le classpath
- ***jta-data-source***, ***non-jta-data-source*** : Le nom JNDI de la source de données.
- ***mapping-file*** : Le fichier de mapping XML
- ***jar-file*** : Le jar à analyser pour trouver les entités
- ***exclude-unlisted-classes*** : Seules les classes explicitement listées seront cherchées
- ***class*** : Permet de lister des classes en dehors de l'archive
- ***shared-cache-mode*** : Permet de configurer de façon transverse le cache de second niveau
- ***validation-mode*** : Permet d'effectuer une validation Java de l'entité avant un opération CRUD, activé par défaut

Propriétés standards

Si il n'y a pas d'annuaire JNDI dans l'environnement, il faut fournir les propriétés JDBC :

- ***javax.persistence.jdbc.driver*** : Driver
- ***javax.persistence.jdbc.url*** : URL de la base
- ***javax.persistence.jdbc.user*** : Utilisateur
- ***javax.persistence.jdbc.password*** : Mot de passe

Propriétés spécifiques Hibernate

Configuration :

hibernate.dialect : Nécessaire pour la génération DDL
hibernate.show_sql, hibernate.format_sql : Traces SQL
hibernate.generate_statistics : Tuning performance
hibernate.max_fetch_depth,
hibernate.default_batch_fetch_size

Connexion et JDBC :

hibernate.jdbc.fetch_size, hibernate.jdbc.batch_size
hibernate.connection.isolation, …

Cache de second-niveau

hibernate.cache.use_second_level_cache,
hibernate.cache.provider_class

Transaction

hibernate.transaction.factory_class, jta.UserTransaction,
hibernate.transaction.manager_lookup_class

La propriété *hibernate.hbm2ddl.auto*

La propriété ***hibernate.hbm2ddl.auto*** permet d'exporter ou de valider un schéma de base de données lors de a création de la *SessionFactory*

4 valeurs sont possibles pour cette propriété :

- ***validate*** : Le schéma de bases de données est validé vis à vis des entités. Si incohérence => Exception.
- ***update*** : Si incohérence, le schéma est modifié pour s'adapter aux entités trouvées => ALTER TABLE.
- ***create*** : La base est créée en fonction des entités trouvées (ordres CREATE TABLE)
- ***create-drop*** : La base est supprimée à l'arrêt de la session factory

Avec les modes *create/create-drop*, il est possible de fournir un script permettant d'initialiser la base avec des données (Par défaut, *import.sql*)

Lister explicitement les entités

- Par défaut, toutes les entités déployées dans la même archive que le fichier *persistence.xml* font partie de l'unité de persistance

- L'élément ***<exclude-unlisted-classes>*** positionné à *true* permet d'exclure toutes les entités non déclarés explicitement au moyen des éléments <*mapping-file*>, <*jar-file*> ou <*class*>

le fichier *persistence.xml*

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="1.0" ...>
<persistence-unit name="entreprise">
<jta-data-source>java:/DefaultDS</jta-data-source>
<exclude-unlisted-classes>true</exclude-unlisted-classes>
<class>vin.Vin</class>
<class>vin.Cepage</class>
<properties>
<property name="hibernate.hbm2ddl.auto" value="create-drop"/>
</properties>
</persistence-unit>
</persistence>
```

Usine à gestionnaires de persistance

- Au runtime, les informations de configuration d'une unité de persistance ne sont lues qu'une seule fois et stockées dans l'*EntityManagerFactory*
  - Dans un environnement Java SE :

```
EntityManagerFactory emf =
   Persistence.createEntityManagerFactory("entreprise") ;
```

  - Dans un environnement JavaEE, l'usine est injectée par le serveur d'application:

```
@PersistenceUnit
private EntityManagerFactory emf
```

Démarrer avec JPA/Hibernate

Architecture Hibernate / JPA

Le fichier *persistence.xml*

Annotations indispensables pour les entités

API EntityManager

Exemple

Classes entités

- Les entités sont de simple Java Beans (constructeur sans argument et getter/setter) qui en plus contiennent
  - Les méta-données de mapping exprimées via des annotations
    - Obligatoires
    - Nécessaire pour des types de champs spécifiques (Enumeration, Date, etc...)
    - Adaptation au modèle physique
    - Associations entres entités

Conception

- Bien que rigoureusement non obligatoire, il est souhaitable que la classe implémente l'interface ***java.io.Serializable***
  - cela permet aux instances de transiter sur le réseau entre le conteneur et un client distant
- Par défaut, la classe entité est **associée à une table** du nom de sa classe
- Une entité est toujours associé à une **clé primaire** (*primary key*) qui permet de l'identifier de façon unique dans la base de données

Annotations obligatoires

L'annotation de classe ***@Entity*** indique que les instances de cette classe pourront être prises en charge par un service de persistance

Au moins, une annotation ***@Id*** doit indiquer la clé primaire

- L'annotation est placée soit sur la déclaration de l'attribut soit sur le getter
- Son emplacement conditionne l'emplacement des autres annotations

Par défaut, tous les attributs ayant des méthodes get/set sont **persistants**

Exemple

```
@Entity
public class Personne implements Serializable{
// l'attribut id désigne la clé primaire, le mapping est défini sur les attributs du bean
// les attributs id, nom, prenom sont mappés respectivement
// sur les colonnes id, nom, prenom
@Id private int id;
private String nom, prenom;
  public Personne(){}
  public int getId(){return id;}
  public void setId(int pk){id=pk;}
  public String getNom(){return nom;}
  public void setNom(String n){nom=n;}
  public String getPrenom(){return prenom;}
  public void setPrenom(String p){prenom=p;}
}
```

Démarrer avec JPA/Hibernate

Architecture Hibernate / JPA
Le fichier *persistence.xml*
Annotations indispensables pour les entités
API EntityManager
Exemple

Gestionnaire de persistance

- Le gestionnaire de persistance, i.e. ***EntityManager*** est l'API permettant de faire les opérations de lecture et d'écriture sur la BD
- C'est un cache qui détecte les changements dans les classes entités attachés afin de les propager dans la base.

Obtention d’une référence

Dans un environnement *JavaSE*, on utilise l'objet *EntityManagerFactory* :

```
EntityManager em = emf.createEntityManager();
```

Dans un environnement JavaEE, le manager est généralement injecté par le serveur applicatif:

```
@PersistenceContext
private EntityManager em
```

Fonctions du gestionnaire de persistance

- Les principales fonctions de l'entity manager sont :
  - Chargement d'entités via leur id
  - insertions d'entités
  - suppression d'entités
  - synchronisation entre entités et base de données
  - Requêtes et recherche d'entité

Chargement d'entité

Charger une entité à partir de sa clé primaire :

- ***<T> T find(Class<T> entityClass, Object primaryKey) :***

  retourne l'entité trouvé ou *null* si l'entité n'est pas trouvée
  Ex : **`em.find(Theme.class, 1l) ;`**

- ***<T> T getReference(Class<T> entityClass,Object primaryKey)***

  retourne le bean entité trouvé ou lance *EntityNotFoundException* si l'entité n'est pas trouvée

Création/Suppression

- Persister un objet :
  void persist(Object entity);
  - le bean transmis en paramètre est inséré en base de données (INSERT SQL)
- Supprimer une entité est :
  void remove(Object entity);
  - le bean transmis en paramètre est supprimé de la base de données (DELETE SQL)

Mise à jour

- La mise à jour s'effectue généralement via : ***<T> T merge(T entity);***
  - le bean entité fourni en paramètre est recopié dans un bean entité attaché à un EntityManager. Il est retourné par la méthode *merge*
  - le bean entité fourni en paramètre reste détaché
  - la représentation en base de données est mise-à-jour avec l'entité attaché (UPDATE SQL)

Gestion du cache

- Rafraîchissement du cache :
  void refresh(Object entity);
  - le bean transmis en paramètre est synchronisé par les données lues en base de données (SELECT SQL)
- Savoir si un bean entité est attaché :
  boolean contains(Object entity);
- Détacher tous les beans entités :
  void clear();
  - Il est prudent d'exécuter la méthode *flush* avant *clear*, de façon à mettre à jour la base de données

FlushMode Hibernate

- La méthode permettant de déclencher les ordres SQL est : ***void flush() ;***
- En général, il n'est pas nécessaire de l'appeler explicitement.
  Par défaut, le flush est automatique et les ordres SQL ne sont exécutés qu'en fin de transaction.

- Avec Hibernate, il est cependant possible de débrayer en mode manuel via la méthode :
  void setFlushMode(FlushModeType flushMode) ;
  - Hibernate propose le choix MANUAL. Les ordres SQL ne sont envoyés que lors d'un appel explicite à *flush()*

Accès API Hibernate via JPA

Depuis la version 2.0, *JPA* offre un accès facile aux API Hibernate sous-jacentes via la méthode ***unwrap*** disponible sur *l'EntityManager* et *l'EntityManagerFactory*

```
Session session = em.unwrap(Session.class);

SessionFactory sessionFactory =
   em.getEntityManagerFactory().unwrap(SessionFactory.class);
```

Démarrer avec JPA/Hibernate

Architecture Hibernate / JPA
Le fichier *persistence.xml*
Annotations indispensables pour les entités
API EntityManager
Exemple

Exemple JPA : Entité Theme

```
@Entity
public class Theme implements Serializable {
    @Id @GeneratedValue( strategy = GenerationType.IDENTITY)
    private Long id;
    private String label;
    @OneToMany(cascade=CascadeType.ALL, fetch=fetchType.LAZY)
    private Set<MotClef> motclefs = new HashSet<MotClef>();

    public Set<MotClef> getMotclefs() {
        return motclefs;
    }
    public void setMotclefs(Set<MotClef> motclefs) {
        this.motclefs = motclefs;
    }
    public Long getId() {
        return id;
    }
    public void setId(Long id) {
        this.id = id;
    }
    public String getLabel() { return label ; }
    public void setLabel(String label) { this.label = label;   }
 … etc
```

Exemple
Requête en lecture des thèmes

```
public List<Theme> getAllThemes(){
    List<Theme> ret = null;
    EntityManager em = emf.createEntityManager();
    EntityTransaction tx = em.getTransaction();
    try {
       tx.begin();
       Query jqlQuery = em.createQuery("from Theme");
       ret = (List<Theme>)jqlQuery.getResultList();
       tx.commit();
    }
                // Gestion des exceptions
        return ret;
    }
        em.close()
```

Exemple
Requête en lecture d'un thème

```
public Theme getThemeComplet(String label){
    EntityManager em = emf.createEntityManager();
    EntityTransaction tx = em.getTransaction();
    Theme ret = null;
    try {
    tx.begin();
    Query jqlQuery =
           em.createQuery("from Theme t where t.label like :labelSearch ");
    jqlQuery.setParameter("labelSearch", "%"+label+"%");
    ret = (Theme)jqlQuery.getSingleResult();
    tx.commit();
    }…… // gestion des exception
        return ret ;
        em.close()
}
```

Exemple
Requête en écriture

```
public void createTheme(Theme t) {
    EntityManager em = emf.createEntityManager();
    EntityTransaction tx = em.getTransaction();
    Theme ret = null;
    try {
        tx.begin();

        em.persist(t);

        tx.commit();
    … // gestion des exceptions
        em.close();
}
```

Exemple
Requête de suppression

```
public void deleteTheme(Theme t) {
    EntityManager em = emf.createEntityManager();
    EntityTransaction tx = em.getTransaction();
    Theme ret = null;
    try {
       tx.begin() ;
       t = em.find(Theme.class,t.getId());
       em.remove(t);
       tx.commit();
    }
    … // gestion des exceptions
    em.close();
}
```

Exemple
Requête de mise à jour

```
public void updateTheme(Theme t) {
    EntityManager em = emf.createEntityManager();
    EntityTransaction tx = em.getTransaction();
    Theme ret = null;
    try {
        tx.begin();
        em.merge(t);
        tx.commit();
    }
        …// gestion des exceptions
        em.close();
}
```