Gestion des entités

Java EE 6 - Sommaire

- Introduction
- Exemple simple
- Entity Manager
- Manipulation d'entités
- JPQL

Introduction

- JPA gère 2 aspects :
  - Le mapping d'objets dans des bases relationnelles
  - La recherche et la gestion de ces entités
- L'Entity Manager est le service central pour gérer les entités : il fournit une API pour créer, rechercher, supprimer et synchroniser des entités avec la base. Il permet aussi l'exécution de requêtes JPQL. Des mécanismes de verrouillage sont aussi possibles

Exemple simple

- Sauvegarde et recherche d'un Book par son id

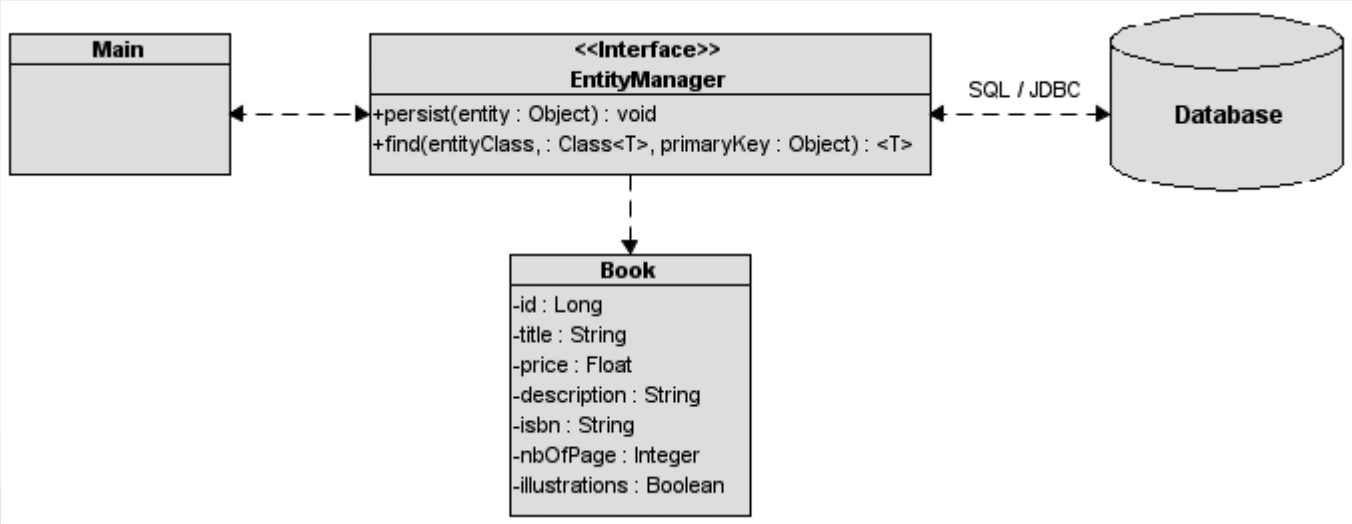
```
public class Main {
  public static void main(String[] args) {
    // 1-Create an instance of the Book entity
    Book book = new Book();
    book.setId(1234L);
    book.setTitle("The Hitchhiker's Guide to the Galaxy");
    book.setPrice(12.5F);
    book.setDescription("Science fiction created by Douglas Adams.");
    book.setIsbn("1-84023-742-2");
    book.setNbOfPage(354);
    book.setIllustrations(false);

    // 2-Get an entity manager and a transaction
    EntityManagerFactory emf = ➥
        Persistence.createEntityManagerFactory("chapter04PU");
    EntityManager em = emf.createEntityManager();
    EntityTransaction tx = em.getTransaction();

    // 3-Persist the book to the database
    tx.begin();
    em.persist(book);
    tx.commit();

    // 4-Retrieve the book by its identifier
    book = em.find(Book.class, 1234L);
    System.out.println(book);
    em.close();
    emf.close();
  }
}
```

```
@Entity
public class Book {
    @Id
    private Long id;
    private String title;
    private Float price;
    private String description;
    private String isbn;
    private Integer nbOfPage;
    private Boolean illustrations;
    // Constructors, getters, setters
}
```

Exemple simple

persistence.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence" version="1.0">
  <persistence-unit name="chapter04PU" transaction-type="RESOURCE_LOCAL">
    <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>
    <class>com.apress.javaee6.chapter04.Book</class>
    <properties>
      <property name="eclipselink.target-database" value="DERBY"/>
      <property name="eclipselink.jdbc.driver" ➥
              value="org.apache.derby.jdbc.ClientDriver"/>
      <property name="eclipselink.jdbc.url" ➥
              value="jdbc:derby://localhost:1527/chapter04DB"/>
      <property name="eclipselink.jdbc.user" value="APP"/>
      <property name="eclipselink.jdbc.password" value="APP"/>
    </properties>
  </persistence-unit>
</persistence>
```



Entity Manager

- C'est le service central de JPA :
  - Il maintient l'état et le cycle de vie des entités
  - Il permet de faire des requêtes dans un contexte de persistence
- Des objets peuvent être managés par l'entity manager ou être détachés
- EntityManager est une interface implémentée par le ”persistence provider”

EntityManager

```
public interface EntityManager {
    public EntityTransaction getTransaction();
    public EntityManagerFactory getEntityManagerFactory();
    public void close();
    public boolean isOpen();

    public void persist(Object entity);
    public <T> T merge(T entity);
    public void remove(Object entity);
    public <T> T find(Class<T> entityClass, Object primaryKey);
    public <T> T find(Class<T> entityClass, Object primaryKey,➥
                LockModeType lockMode);
    public <T> T find(Class<T> entityClass, Object primaryKey,➥
              LockModeType lockMode, Map<String, Object> props);
    public <T> T getReference(Class<T> entityClass, Object pKey);

    public void flush();
    public void setFlushMode(FlushModeType flushMode);
    public FlushModeType getFlushMode();

    public void lock(Object entity, LockModeType lockMode);
    public void lock(Object entity, LockModeType lockMode,➥
                Map<String, Object> properties);

    public void refresh(Object entity);
    public void refresh(Object entity, LockModeType mode);
    public void refresh(Object entity, LockModeType mode,➥
                    Map<String, Object> properties);

    public void clear();
    public void detach(Object entity);
    public boolean contains(Object entity);

    public Map<String, Object> getProperties();
    public Set<String> getSupportedProperties();
    public   Query createQuery(String qlString);
    public   Query createQuery(QueryDefinition qdef);
    public   Query createNamedQuery(String name);
    public   Query createNativeQuery(String sqlString);
    public   Query createNativeQuery(String sqlString,➥
                                    Class resultClass);
    public   Query createNativeQuery(String sqlString,➥
                                      String resultSetMapping);
    public void joinTransaction();

    public <T> T unwrap(Class<T> cls);
    public Object getDelegate();
    public QueryBuilder getQueryBuilder();
}
```

EntityManager

- Récupérer une instance sur l'EntityManager dépend de l'environnement d'exécution :
  - Application-managed environment : le code est responsable de la création de l'entity manager et des transactions
  - Container-managed environment : les transactions sont gérées par le conteneur et l'entity manager peut être injecté par annotation. Cela concerne les servlets, EJB, web services, … : tous les composants qui s'exécutent dans un contexte JEE

EntityManager

- Ex : stateless EJB

```
@Stateless
public class BookBean {
    @PersistenceContext(unitName = "chapter04PU")
    private EntityManager em;

    public void createBook() {
      // Create an instance of book
      Book book = new Book();
      book.setId(1234L);
      book.setTitle("The Hitchhiker's Guide to the Galaxy");
      book.setPrice(12.5F);
      book.setDescription("Science fiction created by Douglas Adams.");
      book.setIsbn("1-84023-742-2");
      book.setNbOfPage(354);
      book.setIllustrations(false);
      // Persist the book to the database
      em.persist(book);
      // Retrieve the book by its identifier
      book = em.find(Book.class, 1234L);
      System.out.println(book);
    }
}
```

Manipulation d'entités

- En plus de permettre des requêtes JPQL complexes, l'EntityManager offre aussi les méthodes d'un DAO générique

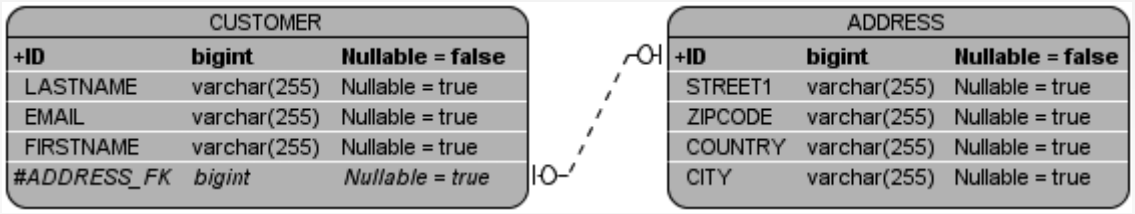
Method	Description
`void persist(Object entity)`	Makes an instance managed and persistent
`<T> T find(Class<T> entityClass, Object primaryKey)`	Searches for an entity of the specified class and primary key
`<T> T getReference(Class<T> entityClass, Object primaryKey)`	Gets an instance, whose state may be lazily fetched
`void remove(Object entity)`	Removes the entity instance from the persistence context and from the underlying database
`<T> T merge(T entity)`	Merges the state of the given entity into the current persistence context
`void refresh(Object entity)`	Refreshes the state of the instance from the database, overwriting changes made to the entity, if any
`void flush()`	Synchronizes the persistence context to the underlying database
`void clear()`	Clears the persistence context, causing all managed entities to become detached
`void clear(Object entity)`	Removes the given entity from the persistence context
`boolean contains(Object entity)`	Checks whether the instance is a managed entity instance belonging to the current persistence context

Manipulation d'entités

- Définissons ainsi 2 entités :

```
@Entity
public class Customer {
    @Id @GeneratedValue
    private Long id;
    private String firstName;
    private String lastName;
    private String email;
    @OneToOne (fetch = FetchType.LAZY)
    @JoinColumn(name = "address_fk")
    private Address address;
    // Constructors, getters, setters
}
```

```
@Entity
public class Address {
        @Id @GeneratedValue
        private Long id;
        private   String   street1;
        private   String   city;
        private   String   zipcode;
        private   String   country;
        // Constructors, getters, setters
    }
```



Manipulation d'entités

- Persistence : sauvegarde de données en base

```
Customer customer = new Customer("Antony", "Balla", "tballa@mail.com");
Address address = new Address("Ritherdon Rd", "London", "8QE", "UK");
customer.setAddress(address);

tx.begin();
em.persist(customer);
em.persist(address);
tx.commit();

assertNotNull(customer.getId());
assertNotNull(address.getId());
```

Manipulation d'entités

- Recherche par ID
  - Récupération de l'entité ou null si pas trouvé

```
Customer customer = em.find(Customer.class, 1234L)
if (customer!= null) {
    // Process the object
}
```

  - Récupération d'une simple référence (lazy)

```
try {
    Customer customer = em.getReference(Customer.class, 1234L)
    // Process the object

    } catch(EntityNotFoundException ex) {
        // Entity not found
    }
```

Manipulation d'entités

- Suppression

```
Customer customer = new Customer("Antony", "Balla", "tballa@mail.com");
Address address = new Address("Ritherdon Rd", "London", "8QE", "UK");
customer.setAddress(address);

tx.begin();
em.persist(customer);
em.persist(address);
tx.commit();

tx.begin();
em.remove(customer);
tx.commit();

// The data is removed from the database
// but the object is still accessible
assertNotNull(customer);
```

Manipulation d'entités

- Synchronisation avec la base
  - Flush : force l'envoi des données vers la base

```
tx.begin();
em.persist(customer);
em.flush();
tx.commit();
```

  - Refresh : recharge l'état à partir de celui de la base

```
Customer customer = em.find(Customer.class, 1234L)
assertEquals(customer.getFirstName(), "Antony");
customer.setFirstName("William");
em.refresh(customer);
assertEquals(customer.getFirstName(), "Antony");
```

Manipulation d'entités

- Contenu du contexte de persistence
  - Le contexte de persistence contient les entités managées. Vous pouvez vérifier auprès de l'entity manager si une entité est managée et retirer toutes les entités du contexte de persistence

Manipulation d'entités

- Contenu du contexte de persistence
  - Contains

```
Customer customer = new Customer("Antony", "Balla", "tballa@mail.com");

tx.begin();
em.persist(customer);
tx.commit();

assertTrue(em.contains(customer));

tx.begin();
em.remove(customer);
tx.commit();

assertFalse(em.contains(customer));
```

Manipulation d'entités

- Contenu du contexte de persistence
  - Clear et detach

```
Customer customer = new Customer("Antony", "Balla", "tballa@mail.com");

tx.begin();
em.persist(customer);
tx.commit();

assertTrue(em.contains(customer));

em.detach(customer);

assertFalse(em.contains(customer));
```

Manipulation d'entités

- Merge : permet de manager de nouveau et de demander la sauvegarde

```
Customer customer = new Customer("Antony", "Balla", "tballa@mail.com");

tx.begin();
em.persist(customer);
tx.commit();

em.clear(); // Called to detach customer

// Sets a new value to a detached entity
customer.setFirstName("William");

tx.begin();
em.merge(customer);
tx.commit();
```

Manipulation d'entités

- Mise à jour
  - Si l'entité est managée, toute modification de ses attributs est persistée automatiquement

```
Customer customer = new Customer("Antony", "Balla", "tballa@mail.com");

tx.begin();
em.persist(customer); // customer becomes managed
customer.setFirstName("Williman"); // changes are sent to the database
tx.commit();
```

JPQL

- En pratique, on a besoin de faire des recherches autrement que par l'ID
- JPQL est le langage de JPA pour faire des requêtes
- JPQL ressemble à SQL, mais JPQL utilise des entités et des collections d'entités et reste orienté objet en utilisant la notation point (dot)
- Par derrière, les requêtes JPQL sont automatiquement transformées en requêtes SQL

JPQL

- Select

```
SELECT <select expression>
FROM <from clause>
[WHERE <conditional expression>]
[ORDER BY <order by clause>]
[GROUP BY <group by clause>]
[HAVING <having clause>]
```

JPQL

- Select : exemples

```
SELECT c FROM Customer c
SELECT c.firstName FROM Customer c
SELECT c.firstName, c.lastName FROM Customer c
SELECT c.address FROM Customer c
SELECT c.address.country.code FROM Customer c
SELECT NEW com.apress.javaee6.CustomerDTO(c.firstName, c.lastName, ➥
                                    c.address.street1) FROM Customer c
SELECT DISTINCT c FROM Customer c
SELECT DISTINCT c.firstName FROM Customer c
SELECT COUNT(c) FROM Customer c
SELECT c FROM Customer c WHERE c.firstName = 'Vincent'
SELECT c FROM Customer c WHERE c.firstName = 'Vincent' ➥
                                   AND c.address.country = 'France'
SELECT c FROM Customer c WHERE c.age > 18
SELECT c FROM Customer c WHERE c.age NOT BETWEEN 40 AND 50
SELECT c FROM Customer c WHERE c.address.country IN ('USA', 'Portugal')
SELECT c FROM Customer c WHERE c.email LIKE '%mail.com'
```

JPQL

- Select : autres exemples
  - Paramètres de requêtes

```
SELECT c FROM Customer c WHERE c.firstName = ?1 AND c.address.country = ?2
SELECT c FROM Customer c WHERE c.firstName = :fname AND c.address.country = :country
```

  - Sous-requêtes

```
SELECT c FROM Customer c WHERE c.age = (SELECT MIN(c. age) FROM Customer c)
```

  - OrderBy, GroupBy, Having

```
SELECT c FROM Customer c WHERE c.age > 18 ORDER BY c.age DESC
SELECT c FROM Customer c WHERE c.age > 18 ORDER BY c.age DESC,
                                                  c.address.country ASC
SELECT c.address.country, count(c) FROM Customer c GROUP BY c.address.country
SELECT c.address.country, count(c) FROM Customer c GROUP BY c.address.country
                                                  HAVING count(c) > 100
```

JPQL

- Delete multiple (efficace)

DELETE FROM <entity name> [[AS] <identification variable>]
[WHERE <conditional expression>]

DELETE FROM Customer c WHERE c.age < 18

JPQL

- Update multiple (efficace)

UPDATE <entity name> [[AS] <identification variable>]
SET <update statement> {, <update statement>}*
[WHERE <conditional expression>]

UPDATE Customer c SET c.firstName = 'TOO YOUNG' WHERE c.age < 18

JPQL

- JPA définit 4 types de requêtes différentes
  - Requêtes dynamiques : requêtes créées à l'éxecution
  - Requêtes nommées : requêtes statiques non modifiables
  - Requêtes natives : exécution de requête SQL
  - Requête avec critère : nouveau concept de JPA

JPQL

- EntityManager permet de créer des requêtes

Method	Description
`Query createQuery(String jpqlString)`	Creates an instance of Query for executing a JPQL statement for dynamic queries
`Query createQuery(QueryDefinition qdef)`	Creates an instance of Query for executing a criteria query
`Query createNamedQuery(String name)`	Creates an instance of Query for executing a named query (in JPQL or in native SQL)
`Query createNativeQuery(String sqlString)`	Creates an instance of Query for executing a native SQL statement
`Query createNativeQuery(String sqlString, Class resultClass)`	Creates an instance of Query for executing a native SQL statement passing the class of the expected results

JPQL

- EntityManager permet de créer des requêtes

```
public interface Query {
    // Executes a query and returns a result
    public List getResultList();
    public Object getSingleResult();
    public int executeUpdate();

    // Sets parameters to the query
    public Query setParameter(String name, Object value);
    public Query setParameter(String name, Date value,➥
                    TemporalType temporalType);
    public Query setParameter(String name, Calendar value,➥
                    TemporalType temporalType);
    public Query setParameter(int position, Object value);
    public Query setParameter(int position, Date value,➥
                    TemporalType temporalType);
    public Query setParameter(int position, Calendar value,➥
                    TemporalType temporalType);
    public Map<String, Object> getNamedParameters();
    public List getPositionalParameters();

    // Constrains the number of results returned by a query
    public Query setMaxResults(int maxResult);
    public int getMaxResults();
    public Query setFirstResult(int startPosition);
    public int getFirstResult();

    // Sets and gets query hints
    public Query setHint(String hintName, Object value);
    public Map<String, Object> getHints();
    public Set<String> getSupportedHints();

    // Sets the flush mode type to be used for the execution
    public Query setFlushMode(FlushModeType flushMode);
    public FlushModeType getFlushMode();

    // Sets the lock mode type to be used for the execution
    public Query setLockMode(LockModeType lockMode);
    public LockModeType getLockMode();

    // Allows access to the provider-specific API
    public <T> T unwrap(Class<T> cls);
}
```

JPQL

- Requêtes dynamiques

```
Query query = em.createQuery("SELECT c FROM Customer c");
List<Customer> customers = query.getResultList();
```

```
String jpqlQuery = "SELECT c FROM Customer c";
if (someCriteria)
    jpqlQuery += " WHERE c.firstName = 'Vincent'";
query = em.createQuery(jpqlQuery);
List<Customer> customers = query.getResultList();
```

```
jpqlQuery = "SELECT c FROM Customer c";
if (someCriteria)
    jpqlQuery += " where c.firstName = :fname";
query = em.createQuery(jpqlQuery);
query.setParameter("fname", "Vincent");
List<Customer> customers = query.getResultList();
```

```
jpqlQuery = "SELECT c FROM Customer c";
if (someCriteria)
    jpqlQuery += " where c.firstName = ?1";
query = em.createQuery(jpqlQuery);
query.setParameter(1, "Vincent");
List<Customer> customers = query.getResultList();
```

```
Query query = em.createQuery("SELECT c FROM Customer c");
query.setMaxResults(10);
List<Customer> customers = query.getResultList();
```

JPQL

- Requêtes nommées

```
@Entity
@NamedQueries({
    @NamedQuery(name = "findAll", query="select c from Customer c"),
    @NamedQuery(name = "findVincent", ➥
        query="select c from Customer c where c.firstName = 'Vincent'"),
    @NamedQuery(name = "findWithParam", ➥
        query="select c from Customer c where c.firstName = :fname")
})
public class Customer {
    ...
}
```

```
@Entity
@NamedQuery(name = "findAll", query="select c from Customer c")
public class Customer {
    ...
}
```

JPQL

- Requêtes nommées

```
Query query = em.createNamedQuery("findAll");
List<Customer> customers = query.getResultList();

Query query = em.createNamedQuery("findWithParam");
query.setParameter("fname", "Vincent");
query.setMaxResults(3);
List<Customer> customers = query.getResultList();

Query query = em.createNamedQuery("findWithParam").setParameter("fname", "Vincent").setMaxResults(3);
List<Customer> customers = query.getResultList();
```

```
@Entity
@NamedQuery(name = Customer.FIND_ALL, query="select c from Customer c"),
public class Customer {
    public static final String FIND_ALL = "Customer.findAll";
    // Attributes, constructors, getters, setters
}

Query query = em.createNamedQuery(Customer.FIND_ALL);
List<Customer> customers = query.getResultList();
```

JPQL

- Requêtes natives

```
Query query = em.createNativeQuery("SELECT * FROM t_customer", Customer.class);
List<Customer> customers = query.getResultList();
```

```
@Entity
@NamedNativeQuery(name = "findAll", query="select * from t_customer")
@Table(name = "t_customer")
public class Customer {
        ...
        // Attributes, constructors, getters, setters
}
```