Criteria API

adriencaubel.fr

Table des matières

Introduction

Rappels

- Nous avons étudié les concepts suivants
  - La notion `@Entity`
  - Les associations `@OneToMany` ... et `mappedBy` (bidirectionnelle)
  - L'héritage `@Inheritance`
  - L'optimisation des lectures (`Fetch.LAZY` / `Fetch.EAGER`)

Nous allons maintenant nous intéresser à **l'API Criteria**

Objectifs ?

> The basic semantics of a Criteria query consists of a SELECT clause, a FROM clause, and an optional WHERE clause, similar to a JPQL query. Criteria queries set these clauses by using Java programming language objects, so the query can be created in a typesafe manner.

- Une API pour écrire des requête SQL directement avec des méthodes Java

Exemple

```
EntityManagerFactory emf = Persistence.createEntityManagerFactory("ma-pu");
EntityManager em = emf.createEntityManager();

CriteriaBuilder cb = em.getCriteriaBuilder();
CriteriaQuery<Pet> cq = cb.createQuery(Pet.class);
Root<Pet> pet = cq.from(Pet.class);                    // from
cq.select(pet);                                        // select
TypedQuery<Pet> q = em.createQuery(cq);
List<Pet> allPets = q.getResultList();
```

<=>

```
List<Pet> = em.createQuery("SELECT p FROM Pet p", Pet.class).getResultList();
```

Pourquoi l'utiliser

Jusqu'à présent, les opérations de base sur les entités (persist, merge, remove) ne permettaient pas de contrôler les conditions WHERE lors de leur exécution. Plusieurs solutions sont possibles :

- faire un filtre directement en Java
- utiliser du JPQL + `createQuery()`
- utiliser l'API Criteria

Les différents filtres

Filtre en Java

```java
// Récupération d'un propriétaire par son ID
Owner owner = em.find(Owner.class, ownerId);

// Récupération des animaux de compagnie du propriétaire
List<Pet> pets = new ArrayList<>();
for (Pet pet : owner.getPets()) {
    if (pet.getType().equals("dog") && pet.getAge() < 5) {
        pets.add(pet);
    }
}
// Trier les animaux par ordre alphabétique
pets.sort(Comparator.comparing(Pet::getName));
```

Combien de requêtes exécutées ?

Filtre en Java

Résultat : Il y aura deux appels SQL (FETCH.LAZY)

- un premier pour récupérer le owner
- et le second lorsqu'on exécute la méthode owner.getPets()

```
[Hibernate]
    select
        o1_0.id,
        o1_0.name
    from
        owners o1_0
    where
        o1_0.id=?
[Hibernate]
    select
        p1_0.owner_id,
        p1_0.id,
        p1_0.age,
        p1_0.name,
        p1_0.type
    from
        pets p1_0
    where
        p1_0.owner_id=?
```

Filtre en JPQL

```
Long ownerId = 1L;  // Replace with the actual owner ID
String jpql = "SELECT p FROM Pet p WHERE p.type = :type
                  AND p.age < :age AND p.owner.id = :ownerId  ORDER BY p.name ASC";

TypedQuery<Pet> query = em.createQuery(jpql, Pet.class);
query.setParameter("type", "dog");
query.setParameter("age", 5);
query.setParameter("ownerId", ownerId);

List<Pet> filteredPets = query.getResultList();
for (Pet pet : filteredPets) {
    System.out.println(pet);
}
```

Combien de requêtes exécutées ?

Filtre en JPQL

Résultat : Un seul appel SQL

```
[Hibernate]
    select
        p1_0.id,
        p1_0.age,
        p1_0.name,
        p1_0.owner_id,
        p1_0.type
    from
        pets p1_0
    where
        p1_0.type=?
        and p1_0.age<?
        and p1_0.owner_id=?
    order by
        p1_0.name
```

Filtre avec Criteria API

```
CriteriaBuilder cb = em.getCriteriaBuilder();
CriteriaQuery<Pet> cq = cb.createQuery(Pet.class);
Root<Pet> pet = cq.from(Pet.class);
Join<Pet, Owner> owner = pet.join("owner");

cq.select(pet)
  .where(
    cb.equal(pet.get("type"), "dog"),
    cb.lessThan(pet.get("age"), 5),
    cb.equal(owner.get("id"), ownerId)
  )
  .orderBy(cb.asc(pet.get("name")));

TypedQuery<Pet> q = em.createQuery(cq);
List<Pet> filteredPets = q.getResultList();
```

Combien de requêtes exécutées ?

Filtre avec Criteria API

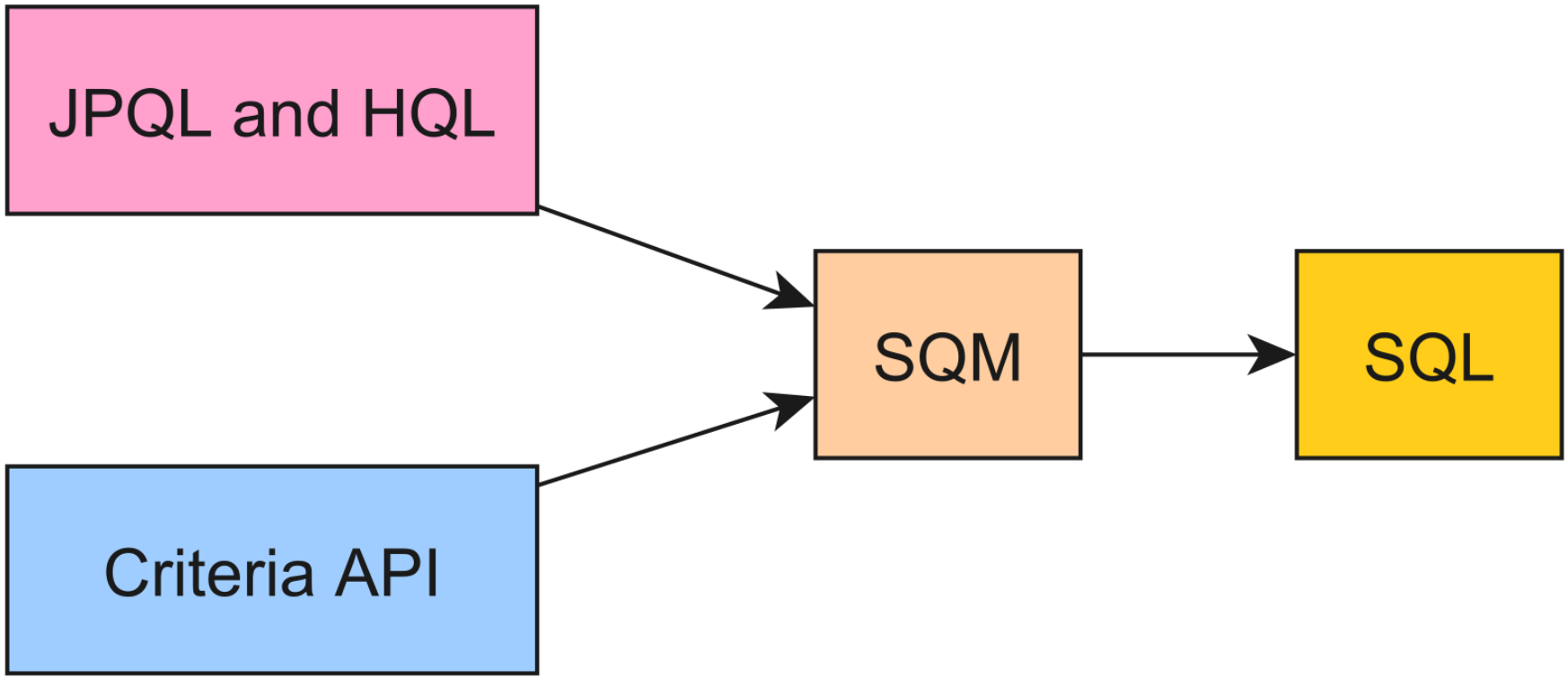
- La construction de la requête est modulaire et peut être facilement étendue ou modifiée.
- Les performances sont généralement meilleures car la requête est optimisée par le fournisseur JPA.

```
[Hibernate]
    select
        p1_0.id,
        p1_0.age,
        p1_0.name,
        p1_0.owner_id,
        p1_0.type
    from
        pets p1_0
    where
        p1_0.type=?
        and p1_0.age<?
        and p1_0.owner_id=?
    order by
        p1_0.name
```

JPQL == Criteria API ?

Hibernate 3, 4 et 5

Hibernate 6



Coder avec l'API Criteria

La classe CriteriaBuilder

La classe CriteriaBuilder est le principal point d'entrée de l'API Criteria.

- `createQuery()` : Crée une instance de **CriteriaQuery** pour définir la requête.
- `equal()`, `greaterThan()`, `like()`, etc. : Créent des prédicats pour filtrer les données.
- `sum()`, `avg()`, `max()`, etc. : Créent des expressions d'agrégation.

```
CriteriaBuilder cb = em.getCriteriaBuilder();  // Créer un CriteriaBuilder depuis EntityManager
CriteriaQuery<Pet> cq = cb.createQuery(Pet.class); // Utiliser le CB pour créer une requête
```

3 Types d’opérations

- `CriteriaQuery` pour le SELECT.
- `CriteriaUpdate` pour l’UPDATE.
- `CriteriaDelete` pour le DELETE.

La classe CriteriaQuery

- **Définition de la source** (`.from()`) : elle renvoie un objet `Root<T>`, qui représente cette entité dans la requête et permet d'accéder aux attributs de celle-ci.
- **Les sélections** (`.select()`) : pour définir les champs à récupérer.
- **Les filtres** (`.where()`) : pour appliquer des conditions de filtrage.
- **Les jointures** (`.join()`) : pour lier plusieurs entités.
- **Les tris** (`.orderBy()`) : pour ordonner les résultats.
- **Les agrégations** (`.groupBy()` , `.having()`) : pour grouper et appliquer des conditions sur les groupes.

Exemple

1. Construire la requête

```
Root<Pet> pet = cq.from(Pet.class);
cq.select(pet);
```

2. Une fois la requête construite, elle peut être exécutée via **l'EntityManager**.

```
TypedQuery<Pet> q = em.createQuery(cq);
List<Pet> allPets = q.getResultList(); // permet d'exécuter la requête
```

Les expressions de prédicats

Définir les conditions de filtrage dans une requête

- `cb.equal(root.get("name"), "John")` : Filtre les entités où le champ `name` est égal à "John".
- `cb.greaterThan(root.get("age"), 18)` : Filtre les entités où l'`age` est supérieur à 18.
- `cb.like(root.get("email"), "%@example.com")` : Filtre les entités dont l'email se termine par "@example.com".
- `cb.isNull(root.get("address"))` : Filtre les entités où l'`address` est null.

Exemple complet

```
// Créer un objet de type CriteriaQuery
CriteriaBuilder cb = em.getCriteriaBuilder();
CriteriaQuery<Pet> cq = cb.createQuery(Pet.class);

// Construire la requête
Root<Pet> pet = cq.from(Pet.class);
Join<Pet, Owner> owner = pet.join("owner"); // Jointure

cq.select(pet)
  .where(
    cb.equal(pet.get("type"), "dog"),
    cb.lessThan(pet.get("age"), 5),
    cb.equal(owner.get("id"), ownerId)
  )
  .orderBy(cb.asc(pet.get("name")));

// Executer la requête
TypedQuery<Pet> q = em.createQuery(cq);
List<Pet> filteredPets = q.getResultList();
```