Java Persistence API (JPA)

AMT
4. JDBC and JPA

Résumé du document
Definition

Table des matières

1. Object Relational Mapping (ORM) pour Java

L'Object Relational Mapping (ORM) est une technique permettant de faire correspondre des objets à des bases de données relationnelles. L'API Java Persistence (JPA) est l'API Java utilisée pour cela et permet aux applications Java d'interagir avec une base de données. Des runtimes JPA comme Hibernate sont des implémentations de l'API JPA, souvent basées sur JDBC. JPA est une API de haut niveau qui réduit le code répétitif.

2. POJO et JavaBeans

Un POJO (Plain Old Java Object) est un objet Java classique sans conventions spécifiques. Les JavaBeans, quant à eux, respectent certaines conventions :

- Un constructeur par défaut public.
- Des champs privés.
- Des getters et setters publics.
- L'implémentation de l'interface Serializable.

Les JavaBeans sont souvent utilisés dans les frameworks pour leur facilité de manipulation via la réflexion.

3. Entités JPA

Une entité JPA est une classe Java représentant une entité en base de données. Elle ressemble à une classe JavaBean et est annotée avec `@Entity`. L'implémentation de `Serializable` n'est requise que dans certains cas spécifiques. Exemple :

```
@Entity
public class Person {
    @Id
    @Column(name = "id")
    private Long id;

    @Column(name = "first_name")
    private String firstName;

    @Column(name = "last_name")
    private String lastName;
}
```

4. Entity Manager

L'Entity Manager gère les entités JPA : création, persistance et suppression. Il est souvent injecté avec l'annotation `@Inject`. Exemple d'utilisation :

```
@ApplicationScoped
public class PersonService {

    @Inject
    private EntityManager em;

    @Transactional
    public void create(Person person) {
        em.persist(person);
    }
}
```

5. Annotations d’entités

Les annotations sont utilisées pour mapper les classes et champs aux tables et colonnes de la base de données :

- `@Entity` pour marquer une classe comme entité.
- `@Id` pour marquer la clé primaire.
- `@Column` pour mapper les champs aux colonnes.
- `@OneToOne`, `@OneToMany`, `@ManyToOne`, `@ManyToMany` pour les relations entre entités.
- `@JoinColumn` pour les clés étrangères.
- `@GeneratedValue` pour générer des clés primaires.
- `@Enumerated` pour mapper un enum.
- `@Embeddable` et `@Embedded` pour mapper des objets embarqués.

Les possibilités de JPA vont bien au-delà de ce cours. Il est conseillé de consulter la documentation de JPA et des runtimes comme Hibernate.

6. Résumé des annotations JPA

6.1. `@OneToOne`

L'annotation `@OneToOne` définit une relation un-à-un entre deux entités, où une instance d'une entité est associée à une seule instance d'une autre entité. Cette relation est souvent représentée par une clé étrangère dans l'une des tables, permettant une liaison directe entre les deux. Elle peut être unidirectionnelle (une seule entité connaît l'autre) ou bidirectionnelle (les deux entités se connaissent mutuellement).

6.2. `@OneToMany`

L'annotation `@OneToMany` décrit une relation un-à-plusieurs, où une entité est associée à plusieurs instances d'une autre entité. La table correspondant à l'entité "enfant" possède généralement une clé étrangère pointant vers l'entité "parent". Ce type de relation est souvent utilisé pour modéliser des scénarios de type parent-enfants, comme un département avec plusieurs employés.

6.3. `@ManyToOne`

L'annotation `@ManyToOne` est l'inverse de `@OneToMany`. Elle représente une relation plusieurs-à-un, où plusieurs instances d'une entité sont liées à une seule instance d'une autre entité. Une clé étrangère dans la table de l'entité "enfant" pointe vers l'entité "parent". Ce type de relation est couramment utilisé lorsque de nombreux objets partagent un élément commun, comme plusieurs employés dans un même département.

6.4. `@ManyToMany`

L'annotation `@ManyToMany` établit une relation plusieurs-à-plusieurs entre deux entités. Chaque entité peut être liée à plusieurs instances de l'autre, ce qui crée une relation réciproque. Cette relation est souvent implémentée à l'aide d'une table de jointure qui stocke les associations entre les deux tables. Par exemple, un étudiant peut être inscrit à plusieurs cours, et chaque cours peut avoir plusieurs étudiants.

6.5. `@JoinColumn`

L'annotation `@JoinColumn` est utilisée pour spécifier la colonne de jointure dans une relation entre deux entités. Elle indique la colonne de la base de données qui sert de clé étrangère pour lier une entité à une autre. Elle est souvent associée à des relations telles que `@ManyToOne` ou `@OneToOne`, afin de définir explicitement la colonne de référence dans la base de données.

6.6. `@Embeddable` et `@Embedded`

L'annotation `@Embeddable` marque une classe comme étant intégrable dans une autre entité. `@Embedded` est utilisée pour indiquer que cette classe intégrée fait partie d'une autre entité. Ces annotations sont utiles pour encapsuler des groupes de champs qui sont réutilisés dans plusieurs entités, sans créer une table distincte. Elles permettent une modularisation des entités tout en évitant des relations complexes de type clé étrangère.