Spring Boot

Olivier Capuozzo, Frédéric Varni

Version 1.2, 2021-01-12

Table des matières

Introduction à JPA

Présentation

JPA est une interface de programmation permettant d'utiliser un modèle objet au-dessus d'une base de données relationnelle.

JPA associe un graphe de classes Java aux tables d'une base de données relationnelle par le biais :

- de fichiers de configuration xml
- d'annotations depuis Java 5 (c'est la méthode préférée)

Pour la suite nous utiliserons le modèle suivant :

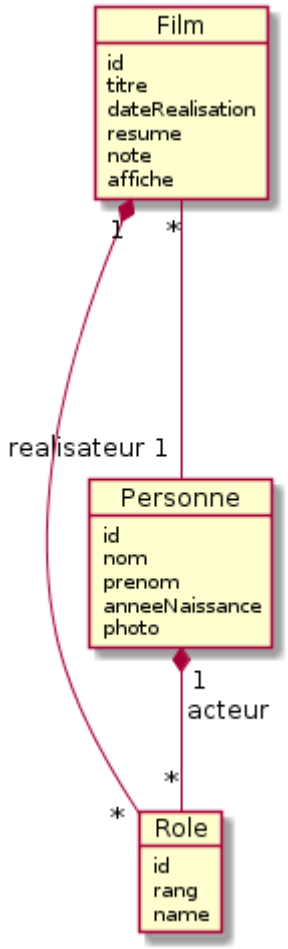


Figure 1. Modèle d'analyse du domaine

Dans ce modèle un acteur peut avoir plusieurs rôles dans un film, ce n'est pa courant, mais ça arrive : dans "Docteur Folamour" Peter Sellers joue à la fois Colonel Lionel Mandrake, Président Merkin Muffley et Docteur Folamour.

Parfois, la base de données préexiste à l'application à développer.

Par exemple, voici un schéma d'une base de données, défini dans le dialecte H2, en phase avec le modèle d'analyse :

Listing 1. Script SQL de la base cinema

```
CREATE TABLE PERSONS
(
    ID BIGINT AUTO_INCREMENT PRIMARY KEY NOT NULL,
    SURNAME VARCHAR(60) NOT NULL,
    GIVENNAME VARCHAR(40),
    BIRTH_YEAR INTEGER,
    IMAGE_PATH VARCHAR(80)
);

CREATE TABLE FILMS
(
    ID BIGINT AUTO_INCREMENT PRIMARY KEY NOT NULL,
    TITLE VARCHAR(50),
    RATING DECIMAL(2,1),
    IMAGE_PATH VARCHAR(120),
    SUMMARY CLOB,
    FILM_DIRECTOR BIGINT,
    CONSTRAINT FILMS_PERSONS_ID_FK FOREIGN KEY (FILM_DIRECTOR)
        REFERENCES PERSONS (ID)
);

CREATE TABLE PLAY
(
    ID BIGINT AUTO_INCREMENT PRIMARY KEY NOT NULL,
    PERSON_ID BIGINT NOT NULL,
    FILM_ID BIGINT NOT NULL,
    RANK INTEGER NOT NULL,
    NAME VARCHAR(90),
    CONSTRAINT PLAY___FK_PERSON FOREIGN KEY (PERSON_ID)
        REFERENCES PERSONS (ID) ON DELETE CASCADE ON UPDATE CASCADE,
    CONSTRAINT PLAY_FILMS_ID_FK FOREIGN KEY (FILM_ID)
        REFERENCES FILMS (ID)
);
```

Une application JPA se divise en deux parties :

- Définir le modèle de données (graphe d'objets)
- Gérer l'accès aux données (créer, récupérer, modifier et supprimer les données)

Définir le modèle

Une traduction de l'analyse

Le modèle qui nous intéresse ici est celui centré sur la logique métier du client, les données qu'ils manipulent. Ces données sont regroupées en **entités**, qui, dans le code, sont représentées par des **classes**, dites **métier**, parfois stéréotypées <<entity>> dans les diagrammes d'analyse.

Il est fort pratique de représenter l'analyse sous la forme d'un diagramme de classes UML, car, en conception (design), l'analyse permet au développeur de se consacrer sur les détails des entités sans perdre le fil de leurs relations.

Associer une table à une classe, les entités (entities)

Listing 2. Classe Person associée à la table PERSONS (sans référence à d'autres entités)

```
package org.vincimelun.cinemajpa.model;

import javax.persistence.*;
import java.util.Collection;
import java.util.Objects;

@Entity ①
@Table(name = "PERSONS") ②
public class Personne
{
    @Id ③
    @GeneratedValue(strategy = GenerationType.IDENTITY) ④
    @Column(name = "ID", nullable = false)
    private long id; ⑤
    @Basic ⑥
    @Column(name = "SURNAME", nullable = false, length = 60) ⑦
    private String nom;
    @Basic
    @Column(name = "GIVENNAME", nullable = true, length = 40)
    private String prenom;
    @Basic
    @Column(name = "BIRTH_YEAR", nullable = true)
    private Integer anneeNaissance;
    @Basic
    @Column(name = "IMAGE_PATH", nullable = true, length = 80)
    private String photo;

    public long getId() {
        return id;
    }

    public void setId(long id) {
        this.id = id;
    }

    public String getNom() {
        return nom;
    }

    public void setNom(String surname) {
        this.nom = surname;
    }
```

```
    public String getPrenom() {
        return prenom;
    }

    public void setPrenom(String givenname) {
        this.prenom = givenname;
    }

    public Integer getAnneeNaissance() {
        return anneeNaissance;
    }

    public void setAnneeNaissance(Integer birthYear) {
        this.anneeNaissance = birthYear;
    }

    public String getPhoto() {
        return photo;
    }

    public void setPhoto(String imagePath) {
        this.photo = imagePath;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Personne that = (Personne) o;
        return id == that.id &&
                Objects.equals(nom, that.nom) &&
                Objects.equals(prenom, that.prenom) &&
                Objects.equals(anneeNaissance, that.anneeNaissance) &&
                Objects.equals(photo, that.photo);
    }

    @Override
    public int hashCode() {
        return Objects.hash(id, nom, prenom, anneeNaissance, photo);
    }

    @Override
    public String toString() {
        return "Personne{" +
                "nom='" + nom + '\'' +
                ", prenom='" + prenom + '\'' +
                ", anneeNaissance=" + anneeNaissance +
                ", photo='" + photo + '\'' +
                '}';
    }
```

```
}
```

① `@Entity` déclare la classe comme un objet persistant associé par défaut à la table de même nom (à la casse près).

② `@Table` déclare le nom de la table associée à la classe, indispensable si les deux noms diffèrent comme c'est le cas ici.

③ `@Id` déclare l'attibut comme clé primaire, au moins un attribut doit être marqué par cette annotation

④ `@GeneratedValue` indique que la valeur est générée automatiquement par le SGBD

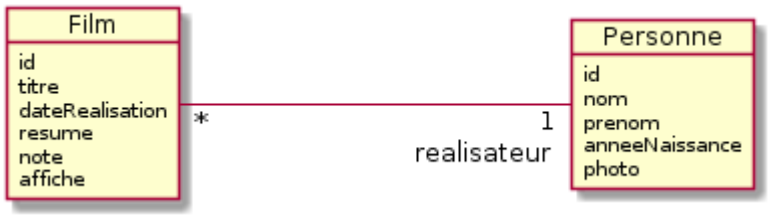
⑤ L'attribut associé à la propriété, l'annotation se fait soit sur les getters, soit sur les attributs, mais on ne mélange pas les styles. De préférence on annote les attributs, ça facilite la lecture de la classe.

⑥ `@Basic` désigne une propriété associée à un type de base.

⑦ `@Column` permet d'établir la correspondance entre la propriété de la classe et la colonne de la table, ainsi que certaines règles de validation commme l'interdiction de nullité, la longueur, le type…

Gérer les associations un vers plusieurs (`ManyToOne` et `OneToMany`)

Dans le modèle un film a un réalisateur et un seul alors qu'une personne peut avoir réalisé plusieurs films.



L'association de `Film` vers `Personne` est de type `ManyToOne`. Cette même association vue côté `Personne` est de type `OneToMany` vers `Film`.

Si l'association est bidirectionnelle, la classe côté `ManyToOne` (dans notre cas `Film`) est dite propriétaire (*owner*) de l'association car sa table associée détient la clé étrangère de la relation. Le côté non propriétaire, ici `Personne`, doit utiliser l'élément *mappedBy* de l'annotation pour spécifier l'attribut du côté propriétaire. Pour les données en base, le système assure la cohérence des liens entre objets en mémoire. Par contre, en cas gestion mémoire de ces liens par la logique applicative, la cohérence est du ressort du développeur.

Listing 3. La classe Film sans les méthodes

```
package org.vincimelun.cinemajpa.model;


import javax.persistence.*;
import java.math.BigDecimal;
import java.util.Collection;
import java.util.Objects;

@Entity
@Table(name="FILMS")
public class Film {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "ID", nullable = false)
    private long id;
    @Basic
    @Column(name = "TITLE", nullable = true, length = 50)
    private String titre;
    @Basic
    @Column(name = "RATING", nullable = true, precision = 1)
    private BigDecimal note;
    @Basic
    @Column(name = "IMAGE_PATH", nullable = true, length = 120)
    private String afficheNom;
    @Basic
    @Lob
    @Column(name = "SUMMARY", nullable = true)
    private String resume;
    @ManyToOne(fetch = FetchType.EAGER) ①
    @JoinColumn(name = "FILM_DIRECTOR") ②
    private Personne realisateur; ③

    ...

}
```

① L'annotation `@ManyToOne` permet de savoir que l'objet annoté doit être retrouvé dans une autre table. Le paramètre `FetchType` permet de savoir s'il faut immédiatement retrouver l'objet lié (`EAGER`) ou s'il faut le retrouver seulement lorsqu'il est accédé dans l'application (`LAZY`). La deuxième option diffère la requête SQL jusqu'à ce que l'application cherche à accéder à l'objet Personne

② L'annotation `@JoinColumn` permet à l'application de déterminer quelle colonne dans la table sert de clé étrangère pour retrouver l'objet lié

③ Si `FILM_DIRECTOR` est une clé étrangère et un entier, l'objet associé est lui de type Personne. Avec JPA le développeur gère un graphe d'objets, pas une base de données SQL.

Listing 4. La classe Person sans les méthodes

```
package org.vincimelun.cinemajpa.model;

import javax.persistence.*;
import java.util.Collection;
import java.util.Objects;

@Entity
@Table(name = "PERSONS")
public class Personne
{
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "ID", nullable = false)
    private long id;
    @Basic
    @Column(name = "SURNAME", nullable = false, length = 60)
    private String nom;
    @Basic
    @Column(name = "GIVENNAME", nullable = true, length = 40)
    private String prenom;
    @Basic
    @Column(name = "BIRTH_YEAR", nullable = true)
    private Integer anneeNaissance;
    @Basic
    @Column(name = "IMAGE_PATH", nullable = true, length = 80)
    private String photo;
    @OneToMany(cascade= CascadeType.ALL, mappedBy = "realisateur") ①
    private Collection<Film> filmsRealises; ②

    ...
}
```

① `@OneToMany` indique qu'une instance de Personne peut être liée à plusieurs instances de Film en tant que réalisateur, c'est l'objet `realisateur` dans Film qui représente ce lien

② Comme une personne peut avoir réalisé plusieurs films, ces films sont stockés dans une collection qui peut être vide si la personne n'a réalisé aucun film.

Gérer l'accès aux données, les dépôts (repositories)

Présentation

Si les classes Entity permettent de définir les structures de données et la façon dont les objets sont liés aux tables SQL, elles ne permettent pas de manipuler les tables : créer, lire, mettre à jour ou supprimer des données. En anglais ces actions sont connues sous le nom de `CRUD` (Create, Read, Update, Delete). Les objets qui permettent de faire des opérations de type `CRUD` sur une base de données sont appelés des DAO (Data Access Object).

Créer des DAOs est une tâche répétitive et ingrate, 90% du code est similaire d'un DAO à l'autre. Spring propose une méthode standard pour gérer les DAOs au travers d'objets, ou plutôt d'interfaces, de type `CrudRepository`, `PagingAndSortingRepository ` ou `JpaRepository` qui hérite de la classe précédente. Dans les cas simples `CrudRepository` suffit largement.

L'interface CrudRepository

Permet d'effectuer toutes les opérations de base d'un DAO :

- `long count()` : compte le nombre d'entités disponibles
- `void delete(T entity)` : supprime l'entité passée en paramètre
- `void deleteAll()` : supprime toutes les entités
- `void deleteById(ID id)` : supprime une entité avec l'id passé en paramètre
- `void existsById(ID id)` : retourne vrai si une entité avec l'id passé en paramètre existe
- `Iterable<T> findAll()` : retourne toutes les instances du type
- `Iterable<T> findAllById(Iterable<ID> id)` :
- `Optional<T> findById(ID id)` : retrouve une entité par son id
- `<S extends T> S save(S entity)` : sauvegarde une entité donnée
- `<S extends T> Iterable<S> saveAll(Iterable<S> entities)` : sauvegarde toutes les entités passées en paramètre.

Ci-dessous un exemple de CrudRepository :

```
package org.vincimelun.cinemajpa.dao;

import org.springframework.data.repository.CrudRepository;
import org.vincimelun.cinemajpa.model.Personne;

public interface PersonneRepository extends CrudRepository<Personne, Long> {
}
```

Configurer un projet Spring Boot pour utiliser JPA

```
# Configuration de Freemarker
spring.freemarker.template-loader-path= classpath:/templates
spring.freemarker.suffix= .ftl

# Connexion à la base de données
spring.datasource.url=jdbc:h2:file:c:/db/cinema ①
#spring.datasource.url=jdbc:h2:file:~/db/cinema ②
spring.datasource.driver-class-name=org.h2.Driver ③
spring.datasource.username=sa ④
spring.datasource.password= ⑤
spring.jpa.hibernate.ddl-auto=create-drop ⑥
spring.datasource.initialization-mode=always ⑦
spring.h2.console.enabled=true ⑧
```

① Chaîne JDBC de connexion à la base de données, version Windows

② La même version Linux ou Mac

③ Driver JDBC à utiliser

④ Utilisateur pour accéder à la source de données JDBC

⑤ Mot de passe de l'utlisateur

⑥ Le schéma de la base de données est généré à partir des entités JPA, à chaque exécution de l'application les tables sont supprimées et recrées, l'option `create-drop` n'est valable que pour la phase de développement, en production on utilise l'option `none`

⑦ Si un fichier `data.sql`est présent, il est automatiquement utilisé pour importer les données qu'il contient dans la base de données

⑧ Permet d'accéder à la console H2 une fois que l'application est lancée à l'URL `http://127.0.0.1/h2-console`, il s'agit d'une application web intégrée au moteur H2 permettant de manipuler la base de données

Travail à faire

A partir du projet `cinemajpa` présent sur la clé :

- Reprendre l'application les templates que vous aves créé pour l'application `cinema` de la semaine dernière et créez la partie contrôleur de `cinemajpa`
- Pour l'entité Personne créez un formulaire qui permette de créer ou de modifier une personne, pour la photo d'une nouvelle personne vous utliserez pour l'instant l'image de substitution `person.png`
- Même chose pour les films, mais en utilisant cette fois `poster.png` comme image de substitution
- Enfin imaginez une interface permettant de gérer les rôles associés à un film.