Spring Data Rest

Achref El Mouelhi

Docteur de l'université d'Aix-Marseille
Chercheur en Programmation par contrainte (IA)
Ingénieur en Génie logiciel

elmouelhi.achref@gmail.com

Spring Boot & REST

Rappel

- Dans une application Spring MVC, le contrôleur, annoté par `@RestController`, utilise le repository pour assurer le service REST.
- Avec Spring Boot Data REST, il est possible de se passer de contrôleur et d'annoter directement le repository pour assurer le service REST.

Spring Boot & REST

Création de projet Spring Boot

- Aller dans `File > New > Other`
- Chercher `Spring`, dans `Spring Boot` sélectionner `Spring Starter Project` et cliquer sur `Next >`
- Saisir
  - `RestSpringBoot` dans `Name`,
  - `com.example` dans `Group`,
  - `RestSpringBoot` dans `Artifact`,
  - `com.example.demo` dans `Package`
- Cliquer sur `Next >`
- Chercher et cocher les cases correspondantes aux `Spring Data JPA`, `MySQL Driver`, `Spring Boot DevTools` et `Rest Repositories` puis cliquer sur `Next >`
- Valider en cliquant sur `Finish`

Explication

- Le package contenant le point d'entrée de notre application (la classe contenant le `puclic static void main`) est `com.example.demo`
- Tous les autres packages `dao`, `model`... doivent être dans le package `demo`.

Et si on part d'un projet existant, il faudra ajouter les dépendances suivantes

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <scope>runtime</scope>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-rest</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-test</artifactId>
  <scope>test</scope>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-devtools</artifactId>
  <scope>runtime</scope>
</dependency>
```

Créons une entité `Personne` dans le package `com.example.demo.model`

```
@Entity @Table(name="personnes")
public class Personne {
  @Id @GeneratedValue
  private Long num;
  private String nom;
  private String prenom;
  public Long getNum() {
    return num;
  }
  public void setNum(Long num) {
    this.num = num;
  }
  public String getNom() {
    return nom;
  }
  public void setNom(String nom) {
    this.nom = nom;
  }
  public String getPrenom() {
    return prenom;
  }
  public void setPrenom(String prenom) {
    this.prenom = prenom;
  }
}
```

Préparons le DAO de la classe `Personne`

```
package com.example.demo.dao;

import org.springframework.data.jpa.repository.
   JpaRepository;

import com.example.model.Personne;

public interface PersonneRepository extends
   JpaRepository<Personne, Long> {

}
```

Dans `application.properties`, on ajoute les données concernant la connexion à la base de données et la configuration de `Hibernate`

```
spring.datasource.url = jdbc:mysql://localhost:3306/myBase?useUnicode=
  true&useJDBCCompliantTimezoneShift=true&useLegacyDatetimeCode=false&
  serverTimezone=UTC
spring.datasource.username =root
spring.datasource.password =root
spring.jpa.hibernate.ddl-auto = update
spring.jpa.show-sql = true
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.
  MySQL5Dialect
```

Cette partie (`?useUnicode=true&useJDBCCompliantTimezoneShift=true &useLegacyDatetimeCode=false&serverTimezone=UTC`) dans la chaîne de connexion est ajoutée pour éviter un bug du connecteur `MySQL` concernant l'heure système.

L'ajout de la propriété `spring.jpa.hibernate.naming.physical-strategy = org.hibernate.boot.model.naming.PhysicalNamingStrategyStandardImpl` permet de forcer **Hibernate** à utiliser les mêmes noms pour les tables et les colonnes que les entités et les attributs.

Spring Boot & REST

Pour que notre repository soit un repository REST, il faut ajouter l'annotation `@RepositoryRestResource`

```
package com.example.demo.dao;

import org.springframework.data.jpa.repository.JpaRepository;

import com.example.model.Personne;

@RepositoryRestResource(
collectionResourceRel="personnes",
itemResourceRel="personne",
path="personnes")
public interface PersonneRepository extends JpaRepository <
   Personne, Long> {

}
```

Pour tester

Il faut aller sur `http://localhost:8080/personnes` ou sur `http://localhost:8080/personnes/1`.

Pour ajouter une personne

- utiliser Postman en précisant la méthode et l'url `http://localhost:8080/personnes`
  - dans `Headers`, préciser la clé `Content-Type` et la valeur `application/json`
  - dans `Body`, cocher `raw` et sélectionner `JSON(application/json)`

Exemple de valeurs à persister

```
{
    "nom": "dalton",
    "prenom": "jack"
}
```

Pour définir une méthode personnalisée

```
package com.example.demo.dao;

import org.springframework.data.jpa.repository.JpaRepository;

import com.example.model.Personne;

@RepositoryRestResource(
collectionResourceRel="personnes",
itemResourceRel="personne",
path="personnes")
public interface PersonneRepository extends JpaRepository <
   Personne, Long> {

  List <Personne> findByNom(@Param("nom") String nom);

}
```

Pour tester

Il faut aller sur

`http://localhost:8080/personnes/search/findByNom?nom=dalton`

Exercice 1

Tester, avec Postman, les trois méthodes **HTTP** `put` et `delete` qui permettront de modifier ou supprimer une personne.

Exercice 2

Créer une application **Angular** qui permet à un utilisateur, via des interfaces graphiques) la gestion de personnes (ajout, modification, suppression, consultation et recherche) en utilisant les web services définis par **Spring**.