Spring Data JDBC

Spring Data JDBC est une bibliothèque puissante qui simplifie les interactions avec les bases de données dans une application Spring en fournissant une approche légère et basée sur les annotations pour travailler avec des bases de données relationnelles. Elle est moins directive que Spring Data JPA et se concentre sur une méthode plus simple et plus explicite de mappage des objets aux tables de base de données. Voici un guide des bases de son utilisation.

1. Configurer votre projet

Pour commencer, vous devez inclure Spring Data JDBC dans votre projet. Si vous utilisez Maven, ajoutez la dépendance suivante à votre `pom.xml` :

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jdbc</artifactId>
</dependency>
```

Vous aurez également besoin d'un pilote JDBC pour votre base de données (par exemple, H2, MySQL, PostgreSQL). Par exemple, pour H2 :

```
<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <scope>runtime</scope>
</dependency>
```

Si vous utilisez Gradle, les équivalents seraient :

```
implementation 'org.springframework.boot:spring-boot-starter-data-jdbc'
runtimeOnly 'com.h2database:h2'
```

2. Configurer votre base de données

Dans votre `application.properties` ou `application.yml`, configurez la connexion à la base de données. Pour une base de données H2 en mémoire, cela pourrait ressembler à ceci :

```
spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=
spring.h2.console.enabled=true
```

Pour une base de données réelle comme PostgreSQL, ajustez l'URL, le nom d'utilisateur et le mot de passe en conséquence.

3. Définir votre modèle de domaine

Créez une classe d'entité simple pour représenter une table dans votre base de données. Spring Data JDBC utilise des conventions où le nom de la classe est mappé au nom de la table (en minuscules par défaut), et les champs sont mappés aux colonnes.

```
import org.springframework.data.annotation.Id;

public class Person {
    @Id
    private Long id;
    private String firstName;
    private String lastName;

    // Constructeur par défaut (requis par Spring Data JDBC)
    public Person() {}

    public Person(String firstName, String lastName) {
        this.firstName = firstName;
        this.lastName = lastName;
    }

    // Getters et setters
    public Long getId() { return id; }
    public void setId(Long id) { this.id = id; }
    public String getFirstName() { return firstName; }
    public void setFirstName(String firstName) { this.firstName = firstName; }
    public String getLastName() { return lastName; }
    public void setLastName(String lastName) { this.lastName = lastName; }
}
```

- `@Id` marque la clé primaire.
- Spring Data JDBC attend un constructeur sans arguments.
- La table sera nommée `person` sauf si elle est remplacée.

4. Créer un dépôt

Définissez une interface qui étend `CrudRepository` pour gérer les opérations CRUD de base :

```java
import org.springframework.data.repository.CrudRepository;

public interface PersonRepository extends CrudRepository<Person, Long> {
}
```

C'est tout ! Vous n'avez pas besoin de l'implémenter—Spring Data JDBC génère l'implémentation au moment de l'exécution.

5. Utiliser le dépôt

Injectez le dépôt dans un service ou un contrôleur et utilisez-le :

```java
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

@Service
public class PersonService {
    private final PersonRepository repository;

    @Autowired
    public PersonService(PersonRepository repository) {
        this.repository = repository;
    }

    public void savePerson() {
        Person person = new Person("John", "Doe");
        repository.save(person);
        System.out.println("Saved person with ID: " + person.getId());
    }

    public void listPeople() {
        Iterable<Person> people = repository.findAll();
        people.forEach(p -> System.out.println(p.getFirstName() + " " + p.getLastName()));
    }
}
```

6. Exécuter votre application

Si vous utilisez Spring Boot, créez une classe principale avec `@SpringBootApplication` et exécutez-la :

```java
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Application {
    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}
```

Spring Boot configurera automatiquement la source de données et Spring Data JDBC.

7. Optionnel : Requêtes personnalisées

Si vous avez besoin de requêtes personnalisées, vous pouvez ajouter des méthodes à votre dépôt avec `@Query` :

```java
import org.springframework.data.jdbc.repository.query.Query;
import org.springframework.data.repository.CrudRepository;

public interface PersonRepository extends CrudRepository<Person, Long> {
    @Query("SELECT * FROM person WHERE first_name = :name")
    List<Person> findByFirstName(String name);
}
```

Points clés à retenir

- **Pas de chargement paresseux :** Contrairement à JPA, Spring Data JDBC ne prend pas en charge le chargement paresseux. Les relations (par exemple, un-à-plusieurs) nécessitent un modèle explicite avec des racines d'agrégat.
- **Racine d'agrégat :** Chaque entité fait partie d'un agrégat, et seule la racine est sauvegardée ou récupérée directement.
- **Création de schéma :** Spring Data JDBC ne crée pas automatiquement les tables. Vous devrez fournir le schéma (par exemple, via `schema.sql` dans `src/main/resources`):

```sql
CREATE TABLE person (
    id BIGINT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    first_name VARCHAR(255),
    last_name VARCHAR(255)
);
```

Exemple de flux de travail

1. Démarrez votre application.
2. Enregistrez un objet `Person` en utilisant `repository.save()`.
3. Récupérez-le avec `repository.findById()` ou `findAll()`.

C'est les bases ! Si vous souhaitez approfondir les relations, les mappages personnalisés ou quelque chose de spécifique, faites-le moi savoir et je personnaliserai l'explication. Comment cela fonctionne-t-il pour vous jusqu'à présent ?