Spring Data JDBC

Spring Data JDBC ist eine leistungsstarke Bibliothek, die die Datenbankinteraktionen in einer Spring-Anwendung durch einen leichtgewichtigen, annotierungsbasierten Ansatz zur Arbeit mit relationalen Datenbanken vereinfacht. Es ist weniger vorurteilsbehaftet als Spring Data JPA und konzentriert sich auf eine einfachere, explizitere Art und Weise, Objekte auf Datenbanktabellen abzubilden. Im Folgenden werde ich Ihnen die Grundlagen der Verwendung erläutern.

1. Projekt einrichten

Um loszulegen, müssen Sie Spring Data JDBC in Ihr Projekt einbinden. Wenn Sie Maven verwenden, fügen Sie die folgende Abhängigkeit zu Ihrer `pom.xml` hinzu:

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

Sie benötigen auch einen JDBC-Treiber für Ihre Datenbank (z.B. H2, MySQL, PostgreSQL). Zum Beispiel für H2:

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

Wenn Sie Gradle verwenden, wären die Äquivalente:

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

2. Datenbank konfigurieren

Konfigurieren Sie die Datenbankverbindung in Ihrer `application.properties` oder `application.yml`. Für eine H2-In-Memory-Datenbank könnte es so aussehen:

```
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
```

Für eine echte Datenbank wie PostgreSQL passen Sie die URL, den Benutzernamen und das Passwort entsprechend an.

3. Domain-Modell definieren

Erstellen Sie eine einfache Entitätsklasse, um eine Tabelle in Ihrer Datenbank darzustellen. Spring Data JDBC verwendet Konventionen, bei denen der Klassenname auf den Tabellennamen (standardmäßig Kleinbuchstaben) und die Felder auf Spalten abgebildet werden.

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

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

    // Standardkonstruktor (von Spring Data JDBC erforderlich)
    public Person() {}

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

    // Getter und Setter
    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` markiert den Primärschlüssel.
- Spring Data JDBC erwartet einen Konstruktor ohne Argumente.
- Die Tabelle wird `person` genannt, es sei denn, sie wird überschrieben.

4. Repository erstellen

Definieren Sie eine Schnittstelle, die `CrudRepository` erweitert, um grundlegende CRUD-Operationen zu handhaben:

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

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

Das war's! Sie müssen es nicht implementieren –Spring Data JDBC erstellt die Implementierung zur Laufzeit.

5. Repository verwenden

Injizieren Sie das Repository in einen Service oder Controller und verwenden Sie es:

```
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("Gespeicherte Person mit ID: " + person.getId());
    }

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

6. Anwendung starten

Wenn Sie Spring Boot verwenden, erstellen Sie eine Hauptklasse mit `@SpringBootApplication` und führen Sie sie aus:

```
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 konfiguriert die Datenquelle und Spring Data JDBC automatisch.

7. Optional: Benutzerdefinierte Abfragen

Wenn Sie benutzerdefinierte Abfragen benötigen, können Sie Methoden zu Ihrem Repository mit `@Query` hinzufügen:

```
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);
}
```

Wichtige Punkte zu beachten

- **Kein Lazy Loading:** Im Gegensatz zu JPA unterstützt Spring Data JDBC kein Lazy Loading. Beziehungen (z.B. eins-zu-viele) benötigen eine explizite Modellierung mit Aggregatwurzeln.
- **Aggregatwurzel:** Jede Entität ist Teil eines Aggregats und nur die Wurzel wird direkt gespeichert oder abgerufen.
- **Schemaerstellung:** Spring Data JDBC erstellt Tabellen nicht automatisch. Sie müssen das Schema bereitstellen (z.B. über `schema.sql` in `src/main/resources`):

```
CREATE TABLE person (
    id BIGINT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
```

```
    first_name VARCHAR(255),
    last_name VARCHAR(255)
);
```

Beispiel-Workflow

1. Starten Sie Ihre App.
2. Speichern Sie ein `Person`-Objekt mit `repository.save()`.
3. Rufen Sie es mit `repository.findById()` oder `findAll()` ab.

Das sind die Grundlagen! Wenn Sie tiefer in Beziehungen, benutzerdefinierte Abbildungen oder etwas Spezielles eintauchen möchten, lassen Sie es mich wissen, und ich werde die Erklärung entsprechend anpassen. Wie läuft es bisher für Sie?