Java Spring & AWS
Sep 7, 2021 - Oct 8, 2021
Monday to Friday
9:30 AM ET - 4:30 PM ET

Agenda: Day -11

- Database access using spring
  - Spring Data JPA

What is JPA?

- Java Persistence API (JPA) is a Java application programming interface specification that describes the management of relational data in applications using Java Platform, Standard Edition and Java Platform, Enterprise Edition.

- Various frameworks like Hibernate, EclipseLink and Apache OpenJPA provide object relational mapping according to JPA standards. This tutorial guides you to develop a simple Java Persistence API (JPA) based database application using Hibernate framework and MySQL database.

What is Hibernate?

- **Hibernate is an object-relational mapping tool that provides an implementation of JPA. Hibernate is one of the most mature JPA implementations** around

Note that JPA is a specification and Hibernate is a JPA provider or implementation.

What Is Spring Data JPA?

- Spring Data is a part of the Spring Framework. The goal of Spring Data repository abstraction is to significantly reduce the amount of boilerplate code required to implement data access layers for various persistence stores.

- Spring Data JPA is not a JPA provider. It is a library/framework that adds an extra layer of abstraction on the top of our JPA provider (like Hibernate).

What Is the Difference Between Hibernate and Spring Data JPA?

- Hibernate is a JPA implementation, while Spring Data JPA is a JPA Data Access Abstraction. Spring Data offers a solution to GenericDao custom implementations. It can also generate JPA queries on your behalf through method name conventions.

- With Spring Data, you may use Hibernate, Eclipse Link, or any other JPA provider. A very interesting benefit is that you can control transaction boundaries declaratively using the @Transactional annotation.

- Spring Data JPA is not an implementation or JPA provider, it's just an abstraction used to significantly reduce the amount of boilerplate code required to implement data access layers for various persistence stores.

Some of the cool features provided by Spring Data JPA are:

- Create and support repositories created with Spring and JPA
- Support QueryDSL and JPA queries
- Audit of domain classes
- Support for batch loading, sorting, dynamical queries
- Supports XML mapping for entities
- Reduce code size for generic CRUD operations by using CrudRepository

When to use Spring Data JPA?

I would say that if you need to quickly create a JPA-based repository layer that is mainly for CRUD operations, and you do not want to create abstract DAO, implementing interfaces, Spring Data JPA is a good choice.

When to use Spring Data JPA?

Examples: steps to follow

- Spring Data JPA Maven Dependencies
- Spring Configuration Classes/file
- Identify proper Model Class
- Spring Data JPA Repository[create a class]
- create Spring Service Class
- Spring Controller Class
- Spring Data JPA Testing

Model Class

```
@Entity
@Table(name = "people")
public class Person
{
……
}
```

```
public interface PersonRepository<P> extends CrudRepository<Person, Long>
{
   List<Person> findByFirstName(String firstName);
}
```

By inheriting from CrudRepository, we can call many methods without the need to implement them our self. Some of these methods are:

- **save**
- **findOne**
- **exists**
- **findAll**
- **count**
- **delete**
- **deleteAll**

Spring Data JPA Repository

We can also define our own methods. These method names should use special keywords such as “find”, “order” with the name of the variables. Spring Data JPA developers have tried to take into account the majority of possible options that you might need. In our example findByFirstName(String firstName) method returns all entries from table where field first_name equals to firstName.

This is one of the most important feature of Spring Data JPA because it reduces a lot of boiler plate code. Also the chances of errors are less because these Spring methods are well tested by many projects already using them.

Spring Service Class

Now that our Spring Data JPA code is ready, next step is to create service class and define methods that we will have to work with database table.

@Transactional : it indicates that the method will be executed in the transaction. Spring will take care of transaction management

```
@Service
public class PersonService {
    @Autowired
    PersonRepository<Person> personRepository;
    @Transactional
    public List<Person> getAllPersons() {
        return (List<Person>) personRepository.findAll();
    }
    @Transactional
    public List<Person> findByName(String name) {
        return personRepository.findByFirstName(name);
    }
    @Transactional
    public Person getById(Long id) {
            return personRepository.findOne(id);
    }
```


Annotations

@Entity signifies that the entity object has significance all by itself it doesn't require any further association with any other object. Where as @Embeddable object doesn't carry any significance all by itself, it needs association with some other object.

Any queries?