Spring Data and JPA

Introduction to Spring Data and JPA

With Spring Data JPA, developers can write data access code using JPA and benefit from Spring Data's repository abstraction and other features. Spring Data JPA provides a set of base repository interfaces, such as JpaRepository and CrudRepository, which offer a range of methods for performing common data access operations, such as querying, saving, and deleting entities.

Spring Data JPA also supports query creation from method names, allowing developers to write queries using a simple method naming convention rather than complex JPQL or SQL queries. Additionally, Spring Data JPA supports pagination, sorting, and auditing, among other features.

Overall, Spring Data and JPA combine to provide a powerful and flexible approach to data access in Spring applications, allowing developers to write concise, efficient, and portable data access codes.

JDBC vs Hibernate vs Spring Data vs Spring Data JPA

JDBC, Hibernate, Spring Data, and Spring Data JPA are all technologies used in Java development for data access. Here are the differences between them:

- JDBC (Java Database Connectivity) is a low-level API for accessing relational databases in Java. It provides standard classes and interfaces for connecting to a database, executing SQL statements, and managing database transactions. Developers must write raw SQL statements and manually handle exceptions and database connections.

- Hibernate is a popular object-relational mapping (ORM) framework that provides a higher-level abstraction for data access than JDBC. It maps Java objects to database tables and provides a set of annotations and APIs for defining the mapping. Hibernate supports many relational databases and provides a range of features, such as lazy loading, caching, and optimistic locking.

- Spring Data is a higher-level abstraction for data access that provides a consistent programming model for different data stores, such as relational databases, NoSQL databases, and cloud-based data stores. It offers a base repository interface set that provides generic CRUD operations and query methods translated to

database-specific queries at runtime. Spring Data also supports pagination, sorting, auditing, and other features.

- Spring Data JPA is a Spring Data module that integrates with the Java Persistence API (JPA). JPA is a standard API for ORM in Java and provides a set of annotations and interfaces for mapping Java objects to relational databases. Spring Data JPA offers a set of base repository interfaces that extend the JPA repository interfaces and provide additional features, such as query method generation from method names and support for pagination, sorting, and auditing.

In short, JDBC is a low-level API, Hibernate is an ORM framework, Spring Data is a higher-level abstraction for data access, and Spring Data JPA is a module of Spring Data that provides integration with JPA for ORM. Each technology offers a different abstraction and functionality for data access in Java applications.

Spring Data and JPA

Spring Data JPA is a framework that simplifies the development of data access layers in Java applications. It provides a way to interact with databases using object-oriented techniques and reduces boilerplate code.

1. Repository: The Repository interface in Spring Data JPA provides a set of methods to perform CRUD (Create, Read, Update, and Delete) operations on the database. It acts as a mediator between the application and the data source. You can define custom methods in your repository interface to perform complex queries or operations.

2. CRUD Repository: The CrudRepository interface extends the Repository interface and provides additional methods to perform CRUD operations. It has predefined methods like save(), findById(), findAll(), deleteById(), etc.

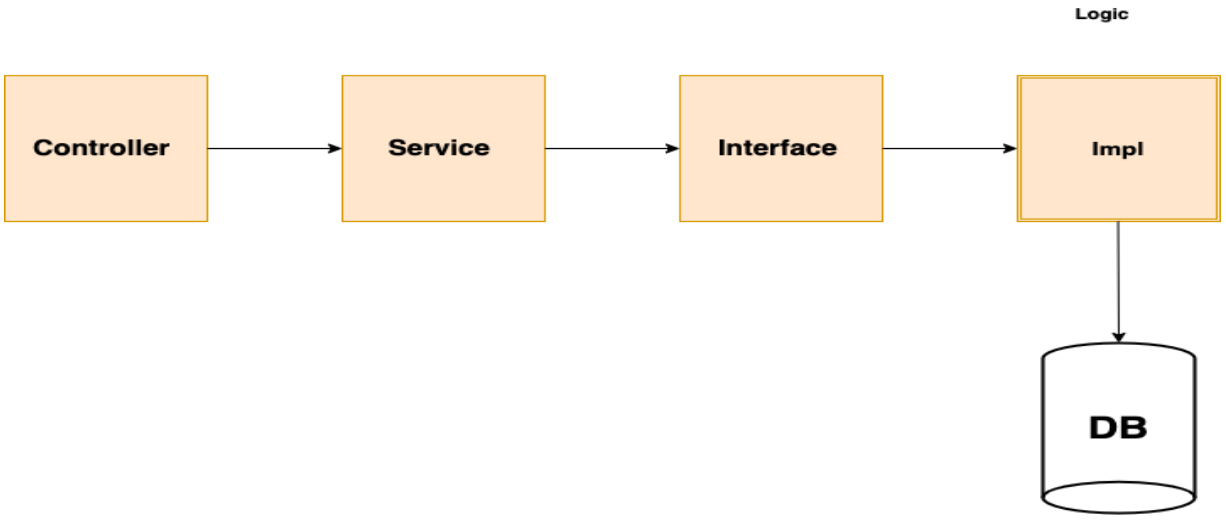
3. Paging and Sorting: The PagingAndSortingRepository interface extends the CrudRepository interface and provides methods for pagination and sorting. It has methods like findAll(Pageable pageable) and findAll(Sort sort) to get data with pagination and sorting.

4. JPA Repository: The JpaRepository interface extends the PagingAndSortingRepository interface and provides additional methods to work with JPA (Java Persistence API). It has methods like flush(), deleteInBatch(), saveAndFlush(), etc., that work with JPA's EntityManager.

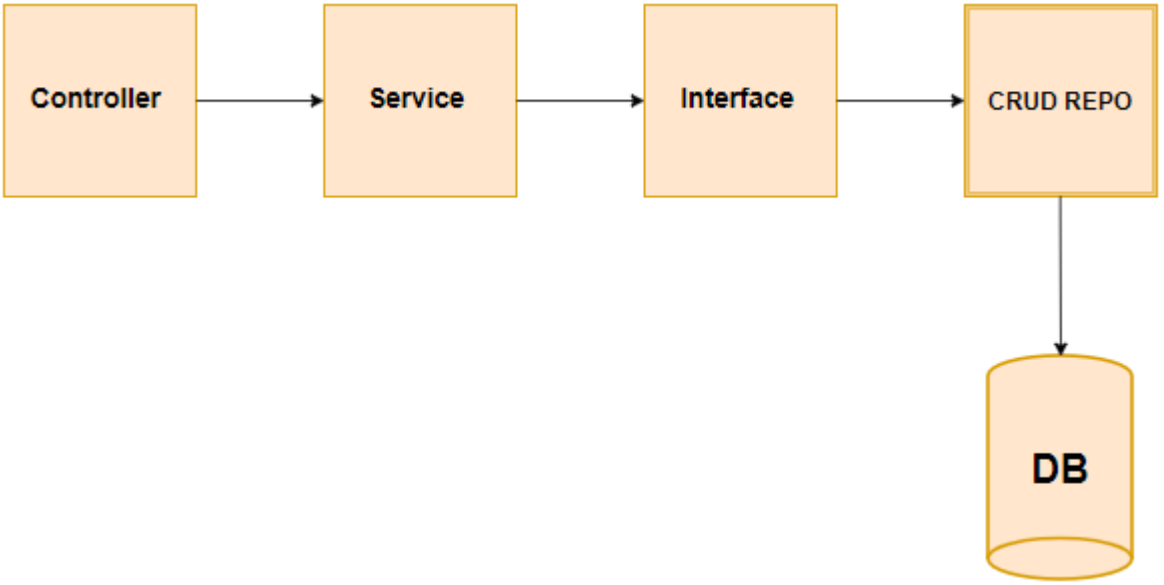
Overall, Spring Data JPA provides much functionality to simplify database access in Java applications. The repository interfaces provide a consistent way to interact with the database and reduce boilerplate code. The paging and sorting support makes it easy to work with large data sets, and the JPA repository provides additional functionality for working with JPA.

Understanding the flow

Earlier



Now



Implementing the API

Now we are required to make some changes in the previous project to use Crud Repository

itemRepository

```
package com.cn.cnkart.dal;

import org.springframework.data.repository.CrudRepository;

import com.cn.cnkart.entity.Item;

public interface ItemRepository extends CrudRepository<Item, Integer> {

}
```

itemDetailsRepostory

```
package com.cn.cnkart.dal;

import org.springframework.data.repository.CrudRepository;

import com.cn.cnkart.entity.ItemDetails;

public interface ItemDetailsRepository extends CrudRepository<ItemDetails,
Integer>{

}
```

Application.yml file

```
spring:
 datasource:
   url: jdbc:mysql://localhost:3306/CNkart
   username: root
   password: password
   driver-class-name: com.mysql.cj.jdbc.Driver
 jpa:
   hibernate.ddl-auto: update
```

itemService

```
package com.cn.cnkart.service;

import java.util.ArrayList;
import java.util.List;

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

import com.cn.cnkart.dal.ItemRepository;
import com.cn.cnkart.entity.Item;

@Service
public class ItemService {

  @Autowired
  ItemRepository itemRepository;

  public Item getItemById(int id) {
     return itemRepository.findById(id).get();
  }


  public void saveItem(Item item) {
     itemRepository.save(item);
  }

  public void delete(int id) {
     itemRepository.deleteById(id);
  }

  public void update(Item updateItem) {
     itemRepository.save(updateItem);
  }

  public List<Item> getItem() {
     List<Item> itemList = new ArrayList<>();
     itemRepository.findAll().forEach(item -> itemList.add(item));
     return itemList;
  }

}
```

itemDetail service

```
package com.cn.cnkart.service;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import com.cn.cnkart.dal.ItemDetailsRepository;

@Service
public class ItemDetailsService {

  @Autowired
  ItemDetailsRepository itemDetailsRepository;

  public void delete(int id) {
     itemDetailsRepository.deleteById(id);
  }
}
```

Conclusion

In conclusion, Spring Data JPA is a powerful tool for Java developers who need to interact with relational databases. It simplifies creating and executing database queries and provides several features that make working with databases more efficient and effective.

By using Spring Data JPA, developers can write less code, reduce the amount of boilerplate required to interact with databases and take advantage of the advanced features provided by the framework, such as caching, lazy loading, and auditing.

Overall, Spring Data JPA is a valuable addition to the Spring ecosystem that can help developers build robust and scalable applications more quickly and easily.

Instructor Codes

- CNKart Application

References

1. JDBC vs Hibernate vs JPA vs Spring Data JPA
2. Spring Data JPA Official Documentation
3. CRUDRepository example