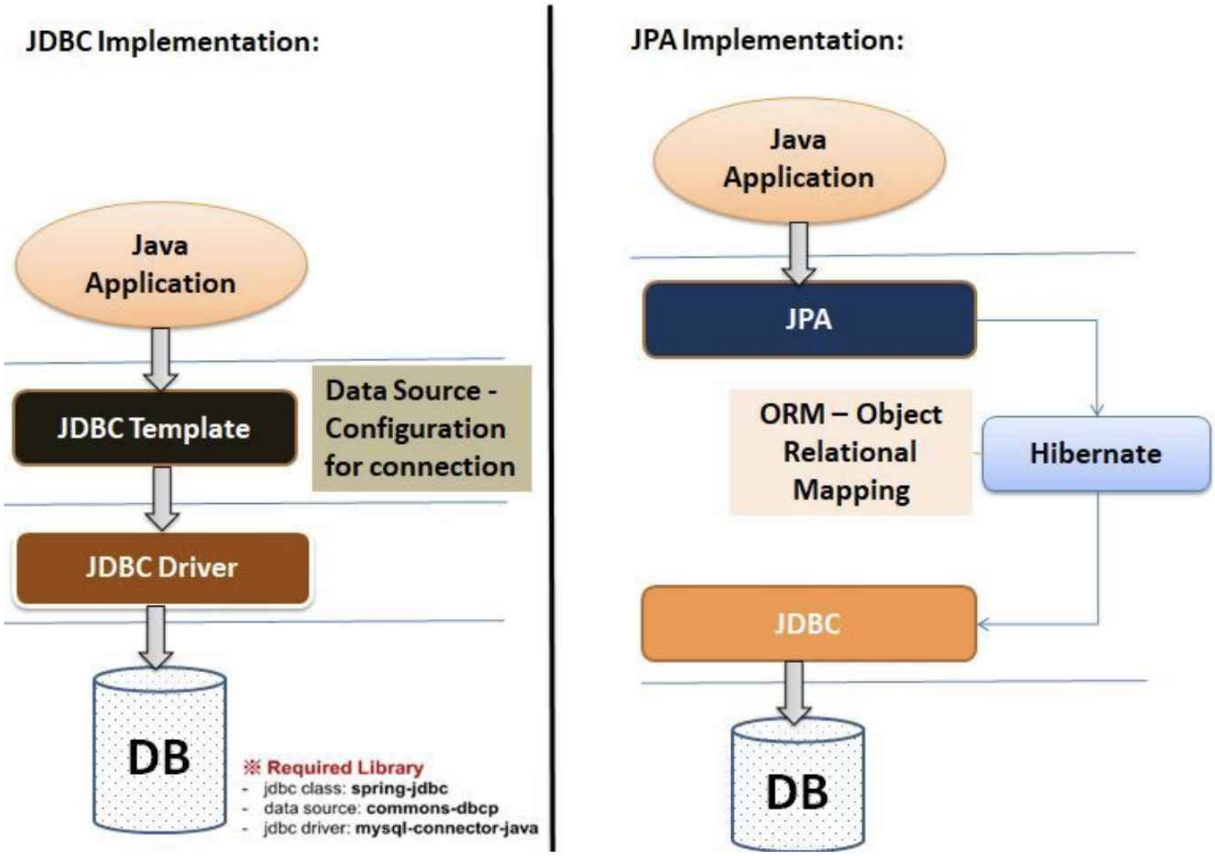
12-Spring Data JPA

Author: Vincent Lau

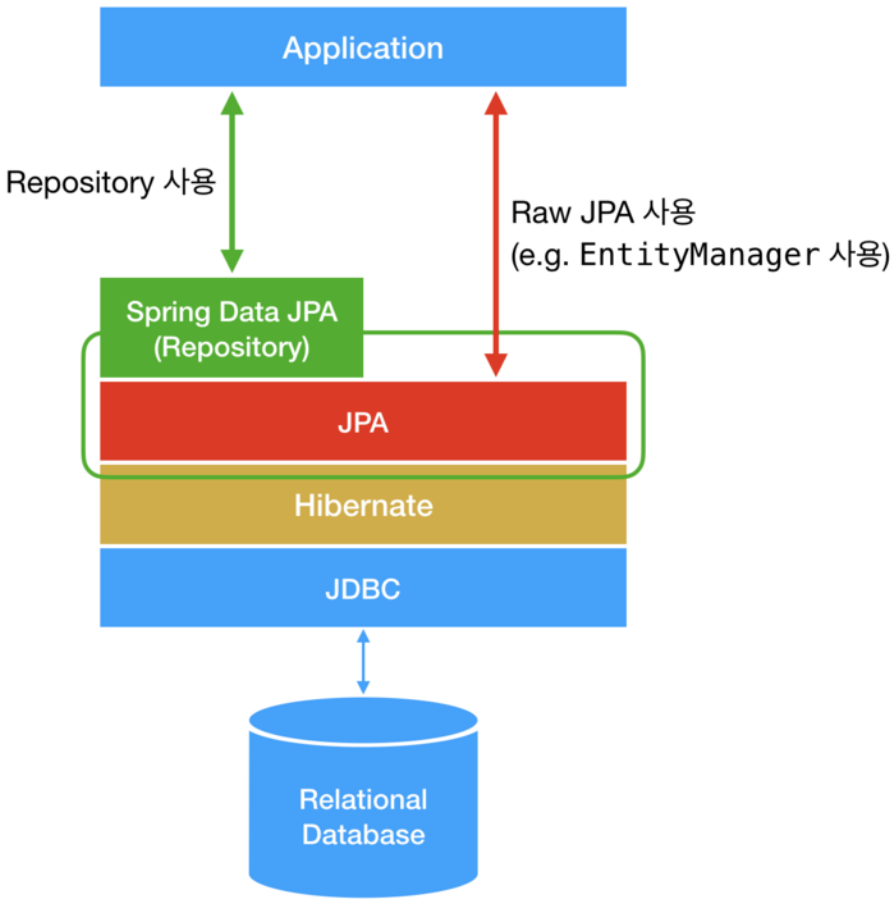
Learning Objectives

> Understand Spring Data & Spring Data JPA.
>
> Understand the Object Relational Mapping & Hibernate Implementation.

Traditional JDBC with Database



Repository & Raw JPA with Database



Spring Data

`Spring Data` is a subproject of the larger Spring Framework ecosystem that aims to simplify and enhance data access and persistence in a Spring-based application. It provides a consistent and feature-rich programming model for working with different data sources, including relational databases, NoSQL databases, and more. Spring Data promotes a more efficient and developer-friendly approach to data access and persistence.

Spring Data offers several modules tailored to various data access technologies, such as **Spring Data JPA (for Java Persistence API), Spring Data MongoDB, Spring Data Redis, Spring Data Neo4j**, and others. Each module integrates seamlessly with Spring applications and provides common, reusable data access components.

A few of the most popular Spring Data projects include:

- ***Spring Data JPA*** - JPA persistence against a **relational** database
- ***Spring Data MongoDB*** - Persistence to a Mongo document database
- ***Spring Data Redis*** - Persistence to a Redis key-value store

Spring Data JPA

`Spring Data JPA` is a module of Spring Data specifically designed for simplifying data access and persistence using the Java Persistence API (JPA). JPA is a Java standard for object-relational mapping (ORM), allowing developers to work with databases using Java objects and entities.

Spring Data JPA builds on top of JPA and provides additional features and simplifications to streamline data access operations in JPA-based applications. It helps eliminate much of the boilerplate code traditionally associated with JPA while promoting best practices in data access.

Key features of Spring Data JPA

Repository Interfaces (No-Code)

Spring Data JPA introduces repository interfaces that allow developers to perform common database operations (such as CRUD) without writing the actual implementation. These interfaces can be automatically implemented by Spring Data JPA. **It reduces the need to implement read and write operations for boilerplate codes.**

Query Methods (Generated JPQL Query)

Developers can define query methods by method naming conventions, reducing the need to write SQL or JPQL queries manually. Spring Data JPA translates method names into appropriate

queries.

Custom Query Methods

You can define custom query methods by adding `@Query` annotations with JPQL or SQL queries. This is useful for more complex queries.

Pagination and Sorting

Spring Data JPA provides built-in support for paginated query results and sorting.

Specification and Criteria API

You can create complex queries using specifications and the Criteria API, offering dynamic query generation.

Auditing

Spring Data JPA supports automatic auditing of entities, recording who created and modified records and when.

Transactions

It integrates seamlessly with Spring's transaction management, ensuring data consistency.

Here's a simplified example of a Spring Data JPA repository interface and an associated entity class:

Object Relational Mapping

`Object-Relational Mapping (ORM)` is a programming technique that allows developers to work with databases using object-oriented programming languages. ORM bridges the gap between the object-oriented model used in application code and the relational model used in a database. It simplifies data access and persistence by providing a way to interact with the database using objects, making it more natural and intuitive for developers.

Key concepts

Entities

In ORM, entities are Java or C# objects that represent data in the database. Each entity typically corresponds to a table in the database. An entity's attributes are mapped to columns in the table.

Mapping

ORM frameworks provide mechanisms for mapping entities to database tables and mapping entity attributes to table columns. This mapping is defined through configuration or annotations.

Object-Relational Mapping

This is the process of translating data between the object-oriented model and the relational model. ORM frameworks handle this translation automatically, so developers don't need to write SQL queries.

CRUD Operations

ORM simplifies common database operations such as Create, Read, Update, and Delete (CRUD). Developers can use object-oriented methods to perform these operations.

Query Language

ORM frameworks often provide their query languages (e.g., JPQL in Java), which allow developers to write database queries using an object-oriented syntax.

Caching

Many ORM frameworks include caching mechanisms to improve performance by reducing the need to query the database frequently.

Transaction Management

ORM frameworks often integrate with the transaction management of the programming language or platform.

Advantages of ORM

- **Portability:** ORM makes it easier to switch between different database management systems (e.g., MySQL, PostgreSQL, Oracle) without changing the application code.
- **Productivity:** ORM reduces the amount of boilerplate code needed for database access, making development faster and less error-prone.
- **Abstraction:** Developers can work with a higher-level, object-oriented representation of data, which is closer to how they think about the problem domain.
- **Security:** ORM frameworks often provide protection against SQL injection and other security vulnerabilities.

Popular ORM Frameworks

- **Hibernate:** A widely used ORM framework for Java applications.
- **Entity Framework:** An ORM framework for .NET applications.

- **JPA (Java Persistence API):** A standard Java API for ORM, often implemented by frameworks like Hibernate.
- **Django ORM:** The ORM system used in the Django web framework for Python.

Hibernate & Spring Data JPA

`Hibernate` is a popular and powerful object-relational mapping (ORM) framework for Java. When using Spring Data JPA, Hibernate is often the underlying ORM framework used to interact with relational databases. Hibernate simplifies the process of mapping Java objects to database tables and provides a wide range of features for data access and persistence.

Integrated with JPA

1. JPA Provider

Hibernate serves as a Java Persistence API (JPA) provider. JPA is a standardized API for object-relational mapping in Java applications. Spring Data JPA builds on top of JPA and can work with any JPA provider, but Hibernate is a commonly used choice due to its feature set and popularity.

2. Entity Mapping

In Spring Data JPA, you define JPA entities that represent your domain model. These entities are annotated with JPA annotations such as `@Entity`, `@Id`, and `@Column` to map Java classes to database tables.

3. Repository Interfaces

Spring Data JPA provides repository interfaces that extend the `JpaRepository` interface or its subtypes. These repository interfaces offer methods for performing common database operations (e.g., CRUD) on JPA entities. Developers can create custom query methods or use query generation based on method names.

4. Configuration

You configure the data source, entity manager factory, and transaction manager in your Spring Boot application's configuration. Hibernate is typically configured as the JPA provider.

Example of Hibernate is used in Spring Data JPA

```
1 @Entity
2 public class User {
3
4     @Id
```

```
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String username;
    private String email;
    // Other fields, getters, and setters
}

public interface UserRepository extends JpaRepository<User, Long> {

    List<User> findByUsername(String username);

    List<User> findByEmail(String email);

}
```

In this example, the `User` class is a JPA entity mapped to a database table. The `UserRepository` interface extends `JpaRepository`, providing methods for data access.

Spring Data JPA manages the underlying Hibernate framework, allowing you to work with JPA entities and perform database operations without writing extensive SQL queries. **Hibernate handles the translation of JPA queries to SQL, manages entity state, and provides advanced features like caching, lazy loading, and more.**

Overall, Hibernate, in conjunction with Spring Data JPA, simplifies database access in Spring applications and promotes good design practices for data access code. It is a powerful combination for developing data-driven applications in Java.