JPA & HIBERNATE

@rohanthapa

Java Persistence API (JPA)

The Java Persistence API (JPA) is a specification for object-relational mapping (ORM) in Java.

It standardizes the way Java objects are mapped to relational database tables, simplifying data access and manipulation.

Key Concepts:

1. **Entity:**
   - An entity represents a table in a database.
   - Each entity instance corresponds to a row in the table.

```java
@Entity
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false, unique = true)
    private String username;

    @Column(nullable = false)
    private String password;

    // Getters and setters
}
```

Key Concepts:

2. **EntityManager:**

- The EntityManager is the primary interface for interacting with the persistence context.
- It provides methods for CRUD operations, querying, and managing entity instances.

```
@PersistenceContext
private EntityManager entityManager;

public void saveUser(User user) {
    entityManager.persist(user);
}
```

Key Concepts:

3. **Persistence Unit:**
   - A persistence unit defines a set of entity classes that are managed together.
   - It is configured in the persistence.xml file.

4. **JPQL (Java Persistence Query Language):**
   - JPQL is similar to SQL but operates on entity objects rather than database tables.
   - It supports queries for entity retrieval and manipulation.

```
@Query("SELECT u FROM User u WHERE u.username = ?1")
Optional<User> findByUsername(String username);
```

Advantages of JPA

- **Standardization:** JPA provides a standardized approach to ORM, ensuring compatibility and consistency across different JPA implementations.

- **Productivity:** JPA reduces boilerplate code, allowing developers to focus on business logic rather than data access code.

- **Portability:** Applications written using JPA are portable across different databases and environments.

Hibernate

Hibernate is an open-source **ORM framework** that implements **JPA**.

It provides advanced features and optimizations for database interactions.

Key Features

1.Automatic Table Creation:

- Hibernate can automatically generate database tables based on entity mappings.

```
spring.jpa.hibernate.ddl-auto=update
```

Key Features

2. Caching:

- Hibernate supports first-level and second-level caching to improve performance.
- First-level cache is associated with the session and is enabled by default.
- Second-level cache is associated with the session factory and is optional.

```java
@Cacheable
@Entity
public class Product {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false)
    private String name;
}
```

Key Features

3. Lazy Loading:

- Lazy loading delays the loading of related entities until they are accessed.

```
@OneToMany(mappedBy = "user", fetch = FetchType.LAZY)
private Set<Order> orders;
```

4. Batch Processing:

- Hibernate can optimize batch operations to improve performance.

```
Session session = sessionFactory.openSession();
Transaction tx = session.beginTransaction();
for (int i = 0; i < 1000; i++) {
    User user = new User( ... );
    session.save(user);
    if (i % 20 == 0) {
        session.flush();
        session.clear();
    }
}
tx.commit();
session.close();
```

Key Features

5.Custom SQL:

- Hibernate allows the use of native SQL queries if needed.

```
@Query(value = "SELECT * FROM users WHERE username =
:username", nativeQuery = true)
Optional<User> findByUsernameNative(@Param("usernam
e") String username);
```

Advantage of Hibernate

- **Rich Feature Set:** Hibernate offers many advanced features not covered by JPA, such as caching, lazy loading, and batch processing.

- **Maturity and Stability:** Hibernate is widely adopted and has been extensively tested in various enterprise applications.

- **Community and Support:** Hibernate has an active user community and extensive documentation.

Integrating JPA and Hibernate with Spring Boot

Spring Boot simplifies the setup and configuration of JPA and Hibernate. Here's how to get started:

Dependencies: Add the necessary dependencies to your **pom.xml** (for Maven) or **build.gradle** (for Gradle).

Maven:

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <scope>runtime</scope>
</dependency>
```

Integrating JPA and Hibernate with Spring Boot

Basic Setup:

1. **Entity Class**: Define your entity class with JPA annotations.

```java
@Entity
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false, unique = true)
    private String username;

    @Column(nullable = false)
    private String password;

    // Getters and setters
}
```

Integrating JPA and Hibernate with Spring Boot

2. Repository Interface: Create a repository interface for data access.

```
public interface UserRepository extends JpaRepository<User, Long> {
    Optional<User> findByUsername(String username);
}
```

3. Application Properties: Configure the database connection in application.properties.

```
spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=password
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
```

Common JPA and HIbernate Annotations

1.**@Entity**:

- Marks a class as an entity.

```
@Entity
public class User { ... }
```

2. **@Table:**

- Specifies the table name

```
@Entity
@Table(name = "users")
public class User { ... }
```

3. **@Id:**

- Specifies the primary key.

```
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
private Long id;
```

Common JPA and HIbernate Annotations

4. **@GeneratedValue:**
   - Specifies the generation strategy for the primary key.

```
@GeneratedValue(strategy = GenerationType.IDENTITY)
```

5. **@Column**:
   - Specifies the column details.

```
@Column(nullable = false, unique = true)
private String username;
```

6. **@OneToOne, @OneToMany, @ManyToOne, @ManyToMany:**
   - **D**efines relationships between entities

```
@OneToMany(mappedBy = "user")
private Set<Order> orders;
```

Common JPA and HIbernate Annotations

7. **@JoinColumn:**
   - Specifies the foreign key column

```
@JoinColumn(name = "user_id")
```

8. **@Query:**
   - Defines custom JPQL or native SQL queries

```
@Query("SELECT u FROM User u WHERE u.username = ?1")
Optional<User> findByUsername(String username);
```

9. **@Cacheable:**
   - Enables second-level caching for an entity

```
@Cacheable
@Entity
public class Product {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false)
    private String name;
}
```

Conclusion

JPA and Hibernate are essential tools for modern Java applications, offering a robust and flexible framework for data persistence and management.

By integrating JPA and Hibernate with Spring Boot, developers can leverage the power of these technologies while enjoying the benefits of Spring Boot's simplicity and ease of use.

Whether you are building a simple CRUD application or a complex enterprise system, understanding and utilizing JPA and Hibernate will significantly enhance your development process and application performance.

Thank You