Entity & DTO & DAO

개념 및 Spring Boot에서의 활용

홍승민

엔티티(Entity)란?

데이터베이스와 직접 연결되는 객체

엔티티의 특징

- 데이터베이스의 테이블과 매핑됨
- @Entity 어노테이션을 붙여서 선언
- @Id를 사용하여 **PK(기본키)**를 정의
- 데이터 저장, 조회, 업데이트가 가능

엔티티 = **도메인**을 **데이터베이스**에 저장하기 위한 객체

Domain과 Entity의 차이

```java
@Entity
@Table(name = "users")
public class UserEntity {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;
    private String email;

    public UserEntity() {}

    public UserEntity(String name, String email) {
        this.name = name;
        this.email = email;
    }

    // Getter & Setter
}
```

Entity

```java
public class User {
    private String name;
    private String email;

    public User(String name, String email) {
        this.name = name;
        this.email = email;
    }

    public String getUserInfo() {
        return name + " (" + email + ")";
    }
}
```

Domain

Entity를 사용한 Service 코드

```java
@Entity
@Table(name = "users")
public class UserEntity {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;
    private String email;

    public UserEntity() {}

    public UserEntity(String name, String email) {
        this.name = name;
        this.email = email;
    }

    // Getter & Setter
}
```

```java
@Service
@RequiredArgsConstructor
public class UserService {
    private final UserRepository userRepository;

    public UserEntity createUser(String name, String email) {
        UserEntity user = new UserEntity(name, email);
        return userRepository.save(user);
    }

    public Optional<UserEntity> getUserByEmail(String email) {
        return userRepository.findByEmail(email);
    }
}
```

Domain을 사용한 Service 코드

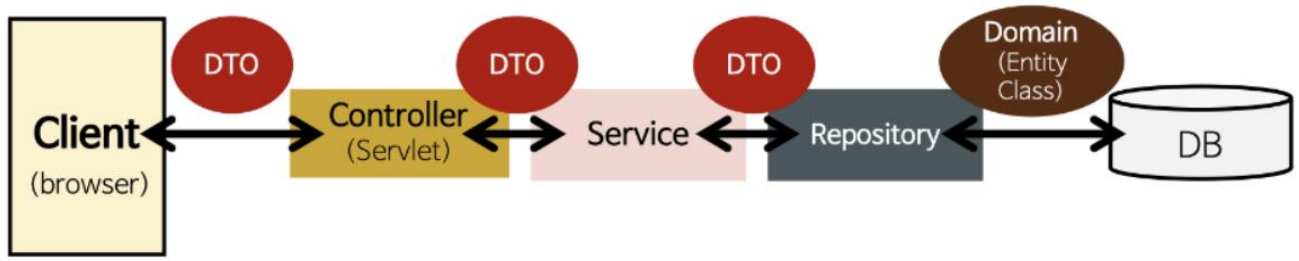
```java
public class User {
    private String name;
    private String email;

    public User(String name, String email) {
        this.name = name;
        this.email = email;
    }

    public String getUserInfo() {
        return name + " (" + email + ")";
    }
}
```

```java
@Service
@RequiredArgsConstructor
public class UserService {
    private final UserRepository userRepository;

    public User createUser(String name, String email) {
        User user = new User(name, email);
        if (!user.isValidEmail()) {
            throw new IllegalArgumentException("Invalid email format");
        }
        UserEntity entity = new UserEntity(user);
        userRepository.save(entity);
        return user;
    }

    public Optional<User> getUserByEmail(String email) {
        return userRepository.findByEmail(email).map(UserEntity::toDomain);
    }
}
```

DTO(Data Transfer Object)란?

사용 목적

- 클라이언트와 서버 간, 혹은 계층 간 **데이터 전달 용도**
- 필요한 데이터만 선별하여 전송 (보안, 성능 최적화)
- 엔티티(Entity)와 분리하여 **불필요한 정보 노출 방지**



DTO를 Entity와 분리하는 이유

```java
public class UserDTO {
    private String name;
    private String email;

    // 생성자, Getter, Setter
}
```

View(클라이언트)와 통신하는 DTO 클래스는 요구사항에 따라 자주 변경되기 때문에.

DAO(Data Access Object)란?

- DAO는 DB와 직접적으로 데이터를 주고받을 수 있게 해줌
- DB에 접근하여 데이터를 CRUD(Create, Read, Update, Delete) 처리
- Service 계층에서 호출하여 비즈니스 로직과 데이터 접근을 분리

```
@Repository
public interface UserDAO extends JpaRepository<UserEntity, Long> {
    Optional<UserEntity> findByEmail(String email);
}
```

DAO와 Repository의 차이

구분	DAO(Data Access Object)	Repository
개념	데이터베이스 접근 계층을 위한 객체	DAO 패턴을 확장한 개념, Spring Data JPA에서 사용
방식	직접 SQL 쿼리를 작성하여 DB와 통신	JPA를 이용해 인터페이스 기반으로 DB 조작
예제	MyBatis, JDBC를 활용하여 SQL 실행	`JpaRepository`, `CrudRepository` 사용
코드 관리	쿼리 로직을 직접 작성해야 함	기본적인 CRUD를 자동으로 제공
Spring Boot에서의 사용	MyBatis 등과 함께 사용	Spring Data JPA에서 주로 사용

DAO와 Repository의 코드 차이

```java
@Repository
public class UserDAO {
    private final JdbcTemplate jdbcTemplate;

    public UserDAO(JdbcTemplate jdbcTemplate) {
        this.jdbcTemplate = jdbcTemplate;
    }

    public UserEntity findByEmail(String email) {
        String sql = "SELECT * FROM users WHERE email = ?";
        return jdbcTemplate.queryForObject(sql, new Object[]{email}, new UserRowMapper());
    }

    public void save(UserEntity user) {
        String sql = "INSERT INTO users (name, email) VALUES (?, ?)";
        jdbcTemplate.update(sql, user.getName(), user.getEmail());
    }
}
```

```java
@Repository
public interface UserRepository extends JpaRepository<UserEntity, Long> {
    Optional<UserEntity> findByEmail(String email);
}
```

즉, **DAO**의 개념을 **Spring Boot**에서는 **Repository**가 대체하고 있다!

Q&A