06 데이터베이스 연동

6.2 ORM

- Object Relational Mapping 객체 관계 매핑
  - 객체지향언어에서 객체와 RDB(Relational Database)의 테이블을 자동으로 매핑하는 방법
  - orm은 이 둘의 불일치와 제약사항을 해결하는 역할
  - orm을 이용하면 쿼리문 작성이 아닌 코드(메서드)로 데이터 조작가능
- 단점:
  - 세분성: 클래스가 테이블 수보다 많아질 수 있음
  - 상속성: rdbms에는 상속이라는 개념이 없음ㅁ
  - 식별성: rdbms는 키본키로 동일성 정의. 자바는 두 객체의 값이 같아도 다르다고 판단 할 수 있음(식별과 동일성)
  - 연관성: 객체지향언어는 객체를 참조함으로써 연관성을 나타내지만 rdbms에는 외래키를 삽입. 객체는 방향성 존재하지만 rdbms에는 양방향
  - 탐색: 자바는 객체 참조같은 연결 수단 활용 (그래프 형태), rdbms에서는 쿼리를 최소화하고 조인을 통해 여러 테이블을 로드하고 값을 추출

6.3 JPA

- Java Persistence API
  - 자바 진영의 orm 기술 표준으로 채택된 인터페이스의 모음

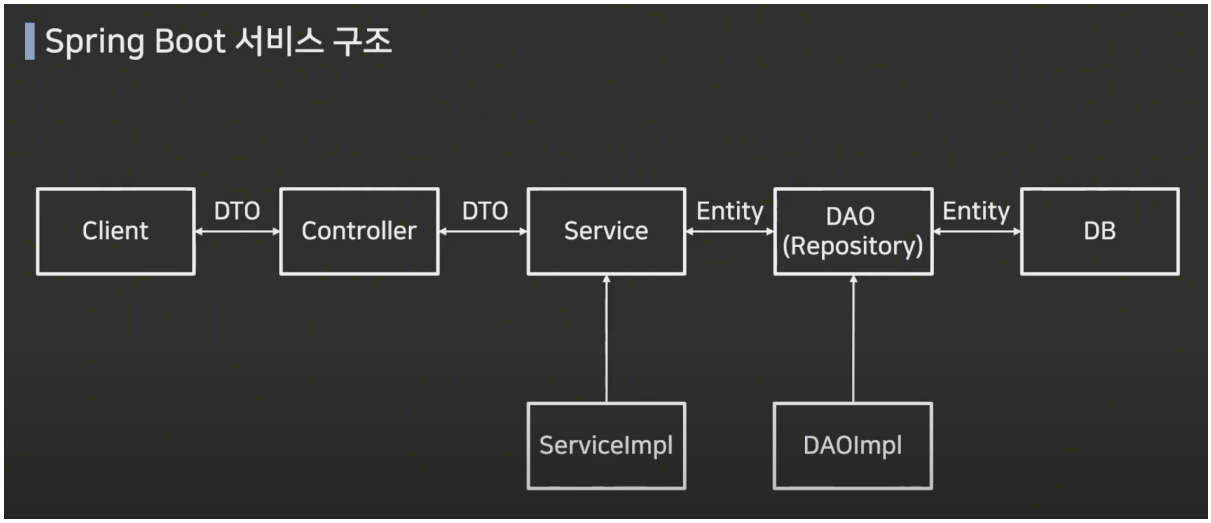
6.6 데이터베이스 연동

1. 프로젝트 생성
   - lombok, Spring Configuration Processor, Spring Web, Spring Data JPA, MariaDB Driver
   - config/SwaggerConfiguration.java
   - resources/logback-spring.xml
   - application.properites 파일에 데이터베이스 관련 설정 추가

```
spring.datasource.driverClassName=org.mariadb.jdbc.Driver
spring.datasource.url=jdbc:mariadb://database-1.chebyqrxrs5q.ap-northeast-2.rds.amazonaws.com:3306/yunji
spring.datasource.username=ssafy
spring.datasource.password=qwer1672

spring.jpa.hibernate.ddl-auto=create
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true
```

6.7 엔티티 설계



- dto:
- entity(Domain):
  - 데이터베이스에 쓰일 컬럼과 여러 엔티티간의 연관관계를 정의
  - 데이터베이스의 테이블을 하나의 엔티티로 생각해도 무방함
  - 서비스에서 dto 를 가지고 엔티티를 만든다.
  - 데이터베이스 테이블에 1:1 매핑이 된다
  - 이 클래스의 필드는 각 테이블 내부의 컬럼을 의미
- Repository:
  - entity에 의해 생성된 db에 db와 통신하는 메소드를 사용하기 위한 인터페이스
  - service와 db를 연결하는 고리의 역할 수행
  - db에 적용하고자 하는 crud를 정의하는 영역
- DAO
  - db에 접근하는 객체를 의미(persistance layer)
  - service 가 db에 연결할 수 있게 해주는 역할
  - service 와 dao 는 인터페이스로 설계 , 실제 로직은 impl에 있다)
  - db를 사용하여 데이터를 조회하거나 조작하는 기능을 전담
  - - 접근하는 본질은 repository가 갖고 있고 그 메소드를 활용하는게 dao!
- DTO
  - dto는 vo(value object)로 불리기도 하며 계층간 데이터교환을 위한 객체를 의미
  - vo의 경우 read only 의 개념을 가지고 있다

6.8 리포지토리 엔티티 설계

```
package com.springboot.jpa.data.repository;
```

```
import com.springboot.jpa.data.entity.Product;
import org.springframework.data.jpa.repository.JpaRepository;
                                                    //매핑할 엔티티, @Id 필드
public interface ProductRepository extends JpaRepository<Product, Long> {


}
```

엔티티: 데이터베이스의 테이블과 구조 생성

리포지토리: 엔티티가 생성한 데이터베이스에 접근

레포지토리를 생성하려면 접근하려는 테이블과 매핑되는 엔티티에 대한 인터페이스를 생성하고 `JpaRepository` 를 상속

- JpaRepository 에서 제공하는 메소드
    - findAll(); findAllById, saveAll, flush, saveAndFlush, saveAllAndFlush, deleteInBatch, deleteAllByInBatch, deleteAllInBatch, getOne, getById, findAll, findAll

6.9 DAO 설계

데이터베이스에 접근하기 위한 로직을 관리하기 위한 객체

[서비스 - (dao) - 리포지토리] but 규모가 작은 서비스에서는 dao 를 별도 설계하지않기도 함

cf) dao 쿼리 수준 / repository 테이블 수준

- dao 는 일반적으로 '인터페이스-구현체' 구성으로 생성'
- 서비스 레이어에서 dao 인터페이스 선언
- ProductDAO Interface / ProductDAOImpl Class

- dao 를 클라이언트 요청과 연결하려면 컨트롤러와 서비스를 생성해야함

6.10 DAO 연동을 위한 컨트롤러와 서비스 설계

dao 메서드를 호출하고 그 외 비즈니스 로직을 수행하는 서비스 레이어를 생성한 후 컨트롤러 생성

서비스 클래스 만들기

- 도메인 모델을 활용해 애플리케이션에서 제공하는 핵심 기능을 제공
- dao와 마찬가지로 추상화해서 구성
- ProductService Interface / ProductServiceImpl
- dao에서 구현한 기능을 서비스 인터페이스에서 호출해 결과값을 가져오게함
- 서비스에서는 클라이언트가 요청한 데이터를 적절하게 가공해서 컨트롤러에게 넘기는 역할
- dto객체와 엔티티 객체를 각 레이어에 변환해 전달하는 역할

컨트롤러 생성

- 컨트롤러는 클라이언트로부터 요청을 받고 해당 요청에 대해 서비스 레이어에 구현된 적절한 메서드를 호출해서 결괏값을 받는다. 요청과 응답을 전달하는 역할만 맡는 것이 좋다