스프링 부트와 JPA 활용 1

1. 프로젝트 환경 설정

1.1 사용 라이브러리

web(톰캣 포함)

thymeleaf(template engine)

jpa

h2(DB)

lombok(getter, setter같은 중복 코드를 annotation을 통해 쉽게 만들어주는 패키지)

validation

1.2 JPA와 DB 설정

.yaml 파일과 .properties 파일 중에 고르면 되는데, 설정이 많아지면 yaml 파일이 편하다.

```
#yaml 파일
spring:
  datasource:
    url: jdbc:h2:tcp://localhost/~/jpashop
    username: sa
    password:
    driver-class-name: org.h2.Driver

  jpa:
    hibernate:
      ddl-auto: create
# ddl-auto:create는 애플리케이션 실행 시점에 테이블을 드랍하고 새로 생성해주는 기능
    properties:
      hibernate:
#     show_sql: true
# System.out을 통해 log를 찍는다.
        format_sql: true
logging:
  level:
    org.hibernate.SQL: debug
#log를 logger를 통해 찍는다.
```

만약 설정 파일이 궁금하면 SPRING 공식 문서를 봐라!

커맨드와 쿼리를 분리해라. 사이드 이펙트가 없도록, 저장을 하고 나면 가급적이면 return값을 member 전체를 return하는 것이 아니라, ID 정도만 return한다.

entity manager를 통한 모든 데이터 변경은 항상 transaction 안에서 이루어져야 한다. 스프링 기본편에 잘 설명되어 있음!

@Transactional annotation이 @Test와 함께 있으면 테스트가 끝난 후 db를 roll back 시킨다.

만약 @Rollback(false) annotation이 같이 붙어 있으면 Rollback 하지 않고 commit 한다.

query parameter 찍는 법

application.yml에서 org.hibernate.type : trace로 주면, value의 로그를 찍어준다.

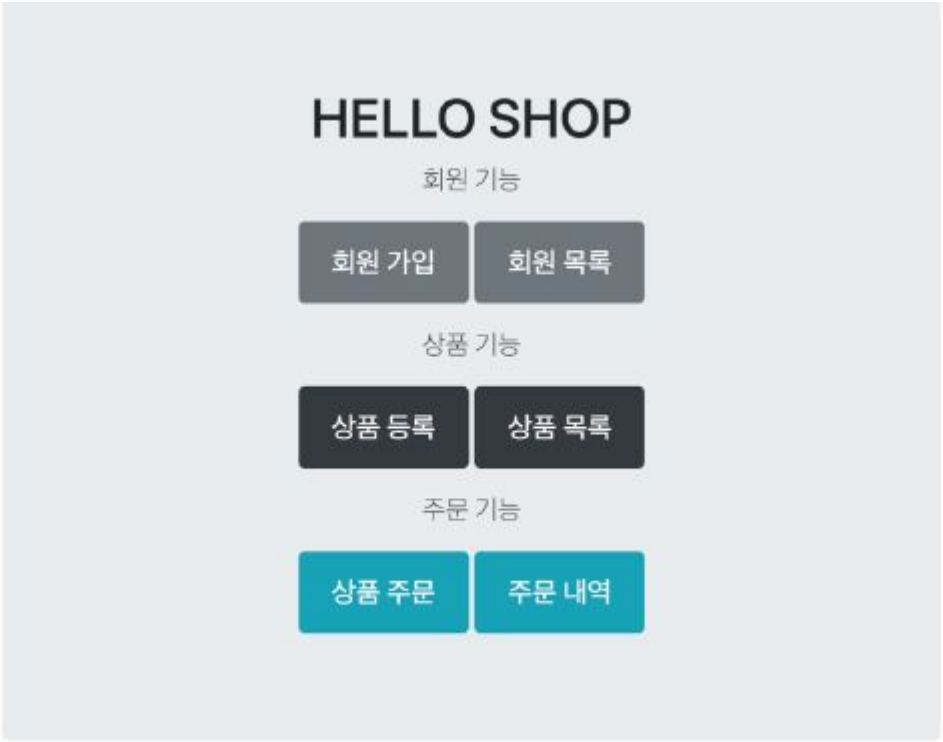
또는 https://github.com/gavlyukovskiy/spring-boot-data-source-decorator의 P6spy를 사용해서 로그를 찍는다.

gradle.build에서 버전이 있는 라이브러리는 스프링에서 미리 버전이 설정되지 않은 라이브러리들이다. 버전이 없는 라이브러리는 스프링에서 자동으로 잘 맞는 버전을 지정해 두었다.

쿼리 파라미터를 로그로 남기는 외부 라이브러리는 시스템 자원을 사용하므로, 개발 단계에서는 편하게 사용해도 된다. 하지만 운영시스템에 적용하려면 꼭 성능테스트를 하고 사용하는 것이 좋다.

2. 도메인 분석 설계

2.1 요구사항 분석



양방향 로직 보다는, 단방향 로직을 짜야 한다!

객체는 collectin을 이용해 다 대 다 관계를 그냥 만들 수 있지만, 관계형 데이터베이스는 중간에 테이블을 두고 다 대 다 관계를 1 대 다, 다 대 1 관계로 풀어내야한다.

1 대 1 관계에서는 보통 Access가 더 자주 일어나는 테이블에 foreign key를 둔다.

연관관계의 주인은 foreign key와 가까운 테이블로 하면 좋다.

실무에서는 @ManytoMany는 거의 안씀. 필드를 추가할 수 없다.

@Embeddable이 붙은 값 타입은 생성자로 값을 정하고, setter로 변경할 수 없어야 한다.

실무에서는 entity에 getter는 열어 두지만, setter는 열어 두면 안된다.

모든 연관관계는 지연 로딩으로 설정되어야 한다.

특히 X to One 관계는 fetchtype을 lazy로 바꿔줘야한다.

컬렉션은 생성자에서 주입 받지 말고 필드에서 바로 초기화한다.

컬렉션을 생성한 이후 바꾸지 말고, 그냥 사용해라!

3. 도메인 개발

3.1 회원 도메인 개발

JPQL은 SQL이랑 조금 다르다. 기능적으로는 거의 다 같지만, SQL은 table을 대상으로 쿼리를 하지만, JPQL은 entity를 대상으로 쿼리를 내린다.

JPQL에 대해서는 기본편에 잘 설명되어 있다.

@PersistenceContext이 있으면 스프링이 생성한 jpa의 entity manager를 주입해준다.

JPA의 모든 데이터 변경이나 로직들은 가급적이면 @Transactional 안에서 실행되어야한다.

스프링에 있는 @Transactional을 사용하는 것을 권장한다.

@Transactional에 readOnly = true 옵션을 주면, 읽기 전용으로 작동해서 더 최적화 할 수 있다.

읽기에는 가급적이면 readOnly = true 옵션을 줘라!

중복 회원 검증 로직이 있더라도, 동시에 호출되면 통과 될 수 있기 때문에, 중복이 안되는 컬럼에 unique 제약 조건을 줘라!

DI는 생성자 주입을 제일 추천한다!

생성자가 딱 하나만 있는 경우면 @Autowired annotation이 없어도 자동으로 주입 시킨다.

또한, 주입되는 변수는 final로 선언하는 것을 추천한다.

@AllArgsConstructor는 클래스의 모든 필드로 생성자를 선언해준다.

@RequiredArgsConstructor는 final로 선언된 Arg를 사용해서 생성자를 생성해준다.

Required Annotation을 사용하는 것을 추천!

디테일한 내용은 LOMBOK을 참고!

entity manger는 @Autowired는 원래는 안 되고, @PersistenceContex annotation이 존재 해야만 스프링 빈에 등록 시켜준다. 하지만 스프링 부트가 지원해서 @Autowired도 작동한다.

데이터를 가지고 있는 쪽에 비즈니스 로직이 있는 것이 제일 좋다. 그러므로 StockQuantity를 관리 하기 위해 비즈니스 로직을 Item에 직접 넣었다.

1. 객체들이 다른 객체에게 참조되지 않는 private owner이고
2. life cycle이 persist 할 때 같이 해야 한다면,

cascade를 써도 된다. 아니라면 쓰면 안된다. 개념이 안 들어 오면 일단 안 쓰는 게 좋다.

JPA는 객체가 바뀌면 dirty check(변경 내역 감지)가 실행 되어 바뀐 데이터의 update query가 db에 날아간다.

엔티티가 비즈니스 로직을 가지고 객체 지향의 특성을 적극 활용하는 것을 도메인 모델 패턴이라 한다.

엔티티에는 비즈니스 로직이 거의 없고 서비스 계층에서 대부분의 비즈니스 로직을 처리하는 것을 트랜잭션 스크립트 패턴이라 한다.

패턴의 정답은 없다. 하나의 서비스에 두 패턴이 양립하기도 한다.

좋은 테스트는 단위 테스트가 있어야 한다. db에도 mocking이 들어가야 함.

3.2 JPA에서 동적 쿼리를 어떻게 해결해야 하는가?