실전! 스프링 부트와 JPA 활용1 - 웹 애플리케이션 개발v2.1

#인강/jpa활용편/활용편1

인프런 강의: 실전! 스프링 부트와 JPA 활용1 - 웹 애플리케이션 개발
인프런: https://www.inflearn.com

버전 수정 이력

v2.1 - 2020-12-27

스프링 부트 버전 업 2.3.x ➡ 2.4.x
junit4 적용 방법 변경

v2.0 - 2020-11-09

스프링 부트 최신 버전 사용 2.1.x ➡ 2.3.x
자바 버전 8 ➡ 11 사용

v1.13 - 2020-10-29

- OrderService에서 itemService ➡ itemRepository로 수정

v1.12 - 2020-10-15

- ${T(jpabook...) 부분 띄어쓰기 PDF 오류 수정

v1.11 - 2020-10-08

- junit5에서 실행시 테스트 없는 오류 보충 설명

v1.10 - 2020-10-01

- 회원 엔티티 분석 그림: Order ➡ Delivery 단방향을 양방향으로 수정

v1.9 - 2020-09-04

- yml 띄어쓰기 주의 내용 추가

v1.8 - 2020-08-12

- H2 데이터베이스 버전 기본을 1.4.199 ➜ **1.4.200으로 사용**

v1.7 - 2020-06-30

- H2 데이터베이스 버전 1.4.200 ➜ 1.4.199로 버전 지정 내용 추가

v1.6 - 2020-06-11

- `updateItemForm.html`에 isbn, submit 누락 수정
- 도움주신 분: OMG

v1.5 - 2020-05-22

- form-control 복사 붙이기 이슈 추가

v1.4 - 2020-05-08

- 스프링 부트 2.1.x 버전을 사용해주세요.

v1.3 - 2020-01-06

H2 데이터베이스 설치

- MVCC 옵션 제거

v1.2 - 2020-01-03

IntelliJ Gradle 대신에 자바 직접 실행

- 내용 추가

주문서비스개발 코드 보충

- delivery.setStatus(DeliveryStatus.READY); 코드 추가
- 도움주신 분: 강프로그래머

v1.1 - 2019-10-30

스프링 부트 버전 변경

2.1.6 ➜ 2.1.9

H2 데이터베이스 버전 명시

Version 1.4.199을 사용해주세요.

도메인 모델과 테이블 설계.연관관계 매핑 분석

도움주신 분: 열심

- 기존: **주문과 배송:** 일대일 단방향 관계다. `Order.delivery` 를 `ORDERS.DELIVERY_ID` 외래 키와 매핑한다.
- 변경: **주문과 배송:** 일대일 양방향 관계다. `Order.delivery` 를 `ORDERS.DELIVERY_ID` 외래 키와 매핑한다.

목차

- 웹 계층 개발
  - 홈 화면과 레이아웃
  - 회원 등록
  - 회원 목록 조회
  - 상품 등록
  - 상품 목록
  - 상품 수정
  - 변경 감지와 병합(merge)
  - 상품 주문
  - 주문 목록 검색, 취소

프로젝트 환경설정

- 프로젝트 생성
- 라이브러리 살펴보기
- View 환경 설정
- H2 데이터베이스 설치
- JPA와 DB 설정, 동작확인

프로젝트 생성

- 스프링 부트 스타터(https://start.spring.io/)
- 사용 기능: web, thymeleaf, jpa, h2, lombok, validation
  - groupId: jpabook
  - artifactId: jpashop

> **스프링 부트 스타터 설정 필독! 주의!**
> 스프링 부트 버전은 2.4.x 버전을 선택해주세요.
> 자바 버전은 11을 선택해주세요.
> Validation (JSR-303 validation with Hibernate validator) 모듈을 꼭! 추가해주세요.(영상에 없습니다.)

> **필독! 주의!**
> 잘 안되면 다음에 나오는 `build.gradle` 파일을 그대로 복사해서 사용해주세요. 강의 영상과 차이가 있습니다.
>
> - 스프링 부트 버전이 2.1.x ➡ 2.4.x로 업그레이드 되었습니다.

- validation 모듈이 추가되었습니다. (최신 스프링 부트에서는 직접 추가해야 합니다.)
- 자바 버전이 1.8 ➡ 11로 업그레이드 되었습니다.

`build.gradle` Gradle 전체 설정

```
plugins {
  id 'org.springframework.boot' version '2.4.1'
  id 'io.spring.dependency-management' version '1.0.10.RELEASE'
  id 'java'
}

group = 'jpabook'
version = '0.0.1-SNAPSHOT'
sourceCompatibility = '11'

configurations {
  compileOnly {
      extendsFrom annotationProcessor
  }
}

repositories {
  mavenCentral()
}

dependencies {
  implementation 'org.springframework.boot:spring-boot-starter-data-jpa'
  implementation 'org.springframework.boot:spring-boot-starter-validation'
  implementation 'org.springframework.boot:spring-boot-starter-thymeleaf'
  implementation 'org.springframework.boot:spring-boot-starter-web'

  compileOnly 'org.projectlombok:lombok'
  runtimeOnly 'com.h2database:h2'

  annotationProcessor 'org.projectlombok:lombok'
  testImplementation 'org.springframework.boot:spring-boot-starter-test'
  //JUnit4 추가
  testImplementation("org.junit.vintage:junit-vintage-engine") {
```

버전은 스프링부트가 궁합이 맞는 버전 알아서 잘 가져다 씀ㅇㅇ;;

```
        exclude group: "org.hamcrest", module: "hamcrest-core"
    }

}

test {
    useJUnitPlatform()
}
```

필독! 주의!

강의 영상이 JUnit4를 기준으로 하기 때문에 `build.gradle` 에 있는 다음 부분을 꼭 직접 추가해주세요. 해당 부분을 입력하지 않으면 JUnit5로 동작합니다. JUnit5를 잘 알고 선호하시면 입력하지 않아도 됩니다.

```
//JUnit4 추가
testImplementation("org.junit.vintage:junit-vintage-engine") {
    exclude group: "org.hamcrest", module: "hamcrest-core"
}
```

- 동작 확인
  - 기본 테스트 케이스 실행
  - 스프링 부트 메인 실행 후 에러페이지로 간단하게 동작 확인(`http://localhost:8080')

롬복 적용 롬복사용시 필수사항

1. Prefrences ➡ plugin ➡ lombok 검색 실행 (재시작)
2. Prefrences ➡ Annotation Processors 검색 ➡ Enable annotation processing 체크 (재시작)
3. 임의의 테스트 클래스를 만들고 @Getter, @Setter 확인

IntelliJ Gradle 대신에 자바 직접 실행

> 참고: 강의에 이후에 추가된 내용입니다.

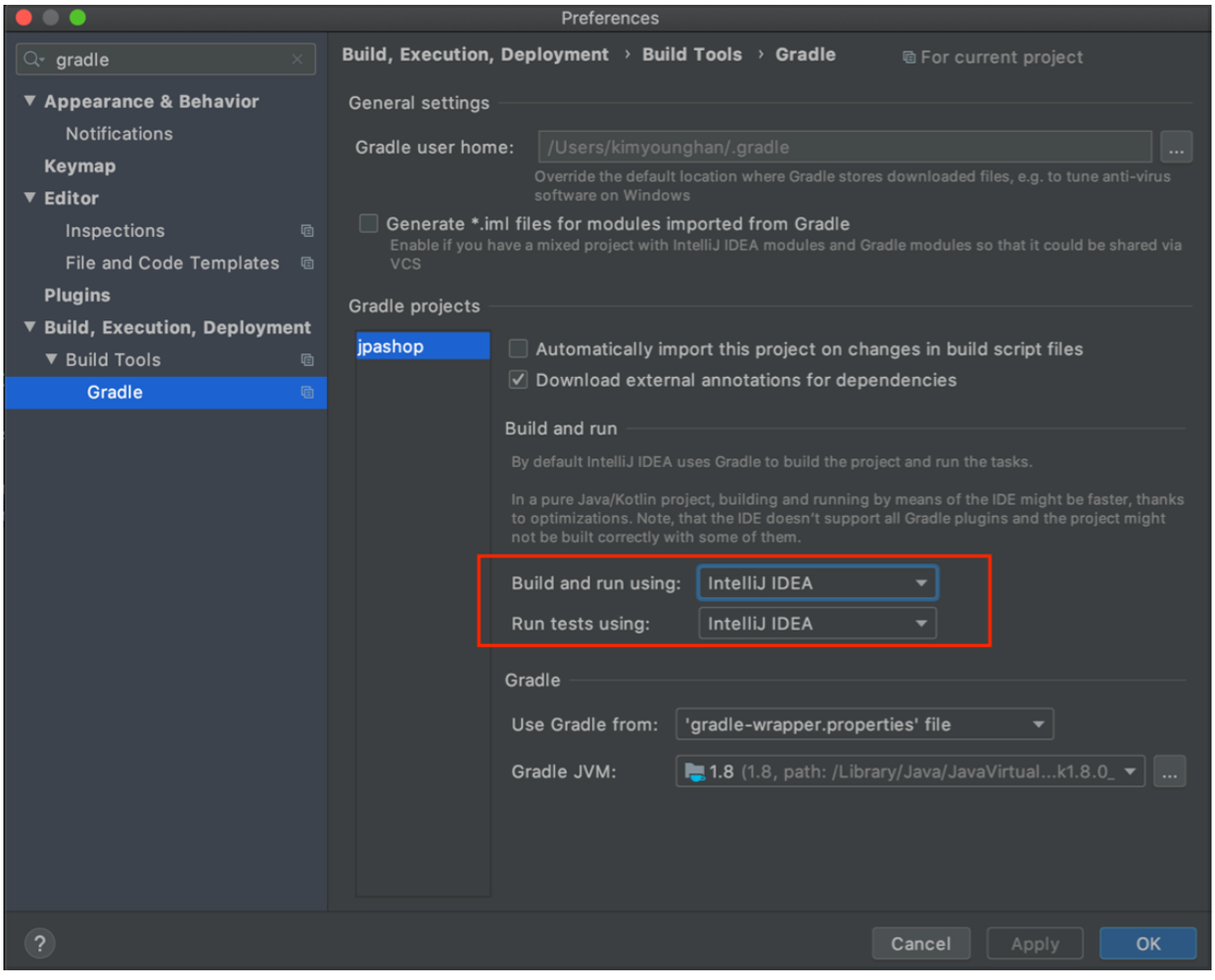
최근 IntelliJ 버전은 Gradle로 실행을 하는 것이 기본 설정이다. 이렇게 하면 실행속도가 느리다. 다음과 같이 변경하면 자바로 바로 실행해서 실행속도가 더 빠르다.

Preferences ➜ Build, Execution, Deployment ➜ Build Tools ➜ Gradle

Build and run using: Gradle ➜ IntelliJ IDEA

Run tests using: Gradle ➜ IntelliJ IDEA

설정 이미지



라이브러리 살펴보기

gradle 의존관계 보기

```
./gradlew dependencies –configuration compileClasspath
```

스프링 부트 라이브러리 살펴보기

- spring-boot-starter-web
  - spring-boot-starter-tomcat: 톰캣 (웹서버)
  - spring-webmvc: 스프링 웹 MVC
- spring-boot-starter-thymeleaf: 타임리프 템플릿 엔진(View)
- spring-boot-starter-data-jpa
  - spring-boot-starter-aop

    - spring-boot-starter-jdbc
        - HikariCP 커넥션 풀 (부트 2.0 기본) HikariCP 중요!
    - hibernate + JPA: 하이버네이트 + JPA
    - spring-data-jpa: 스프링 데이터 JPA
- spring-boot-starter(공통): 스프링 부트 + 스프링 코어 + 로깅
    - spring-boot
        - spring-core
    - spring-boot-starter-logging
        - logback, slf4j

테스트 라이브러리

- spring-boot-starter-test
    - junit: 테스트 프레임워크 Test에 필수 라이브러리
    - mockito: 목 라이브러리
    - assertj: 테스트 코드를 좀 더 편하게 작성하게 도와주는 라이브러리
    - spring-test: 스프링 통합 테스트 지원 이것도 필수 라이브러리

- 핵심 라이브러리
    - 스프링 MVC
    - 스프링 ORM
    - JPA, 하이버네이트
    - 스프링 데이터 JPA
- 기타 라이브러리
    - H2 데이터베이스 클라이언트
    - 커넥션 풀: 부트 기본은 HikariCP
    - WEB(thymeleaf)
    - 로깅 SLF4J & LogBack
    - 테스트

참고: 스프링 데이터 JPA는 스프링과 JPA를 먼저 이해하고 사용해야 하는 응용기술이다.

View 환경 설정

thymeleaf 템플릿 엔진 Spring은 타임리프를 밀어줌

- thymeleaf 공식 사이트: https://www.thymeleaf.org/
- 스프링 공식 튜토리얼: https://spring.io/guides/gs/serving-web-content/
- 스프링부트 메뉴얼: https://docs.spring.io/spring-boot/docs/2.1.6.RELEASE/reference/html/boot-features-developing-web-applications.html#boot-features-spring-mvc-template-

engines

- 스프링 부트 thymeleaf viewName 매핑
  - `resources:templates/` +{ViewName}+ `.html`

localhost:8080/hello를 치면 GetMapping("hello")를 거쳐 return hello->hello.html을 랜더링해서 보여주게 된다.

```
@Controller
public class HelloController {


    @GetMapping("hello")
    public String hello(Model model) {
        model.addAttribute("data", "hello!!");
        return "hello";
    }
}
```

model에 데이타를 실어서 view에 보낼 수 있음.

return은 화면이름 resources/templates/ hello.html로 넘어가는것임.

thymeleaf 템플릿엔진 동작 확인(hello.html)

정적컨텐츠->static 랜더링->templates

```
<!DOCTYPE HTML>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Hello</title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>
<body>
<p th:text="'안녕하세요. ' + ${data}" >안녕하세요. 손님</p>
</body>
</html>
```

위치: `resources/templates/hello.html`

- index.html 하나 만들기
  - `static/index.html` 기본 welcome page

```
<!DOCTYPE HTML>
<html xmlns:th="http://www.thymeleaf.org">
```

```
<head>
    <title>Hello</title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>
<body>
Hello
<a href="/hello">hello</a>
</body>
</html>
```

> 참고: `spring-boot-devtools` 라이브러리를 추가하면, `html` 파일을 컴파일만 해주면 서버 재시작 없이 View 파일 변경이 가능하다.
> 인텔리J 컴파일 방법: 메뉴 build ➡ Recompile

매번 번거롭게 서버재실행 필요xxxx

H2 데이터베이스 설치

개발이나 테스트 용도로 가볍고 편리한 DB, 웹 화면 제공

> **주의! Version 1.4.200를 사용해주세요.**
> 1.4.200 버전 다운로드 링크

- 윈도우 설치 버전: https://h2database.com/h2-setup-2019-10-14.exe
- 윈도우, 맥, 리눅스 실행 버전: https://h2database.com/h2-2019-10-14.zip

- https://www.h2database.com
- 다운로드 및 설치
- 데이터베이스 파일 생성 방법
  - `jdbc:h2:~/jpashop` (최소 한번)
  - `~/jpashop.mv.db` 파일 생성 확인
  - 이후 부터는 `jdbc:h2:tcp://localhost/~/jpashop` 이렇게 접속

chmod 755 h2.sh 잊지말기

DB파일을 생성할 경로를 지정

> 주의: H2 데이터베이스의 MVCC 옵션은 H2 1.4.198 버전부터 제거되었습니다. **1.4.200 버전에서는 MVCC 옵션을 사용하면 오류가 발생**합니다.

JPA와 DB 설정, 동작확인

`main/resources/application.yml`

application.properties삭제 후 yml사용-> 설정파일이 많아지면 yml이 편리하다 함.

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

  jpa:
    hibernate:
      ddl-auto: create
    properties:
      hibernate:
#        show_sql: true
        format_sql: true

logging.level:
  org.hibernate.SQL: debug
#  org.hibernate.type: trace
```

<- 데이터베이스설정에 관련된 커넥션은 설정이 됨

create모드->자동으로 테이블을 만들어주는 모드

- spring.jpa.hibernate.ddl-auto: create
  - 이 옵션은 애플리케이션 실행 시점에 테이블을 drop 하고, 다시 생성한다.

> 참고: 모든 로그 출력은 가급적 로거를 통해 남겨야 한다.
> `show_sql` : 옵션은 `System.out` 에 하이버네이트 실행 SQL을 남긴다.
> `org.hibernate.SQL` : 옵션은 logger를 통해 하이버네이트 실행 SQL을 남긴다.

로그들은 로거들로 남겨야해서 show는 안쓴다

> **주의!** `application.yml` 같은 `yml` 파일은 띄어쓰기(스페이스) 2칸으로 계층을 만듭니다. 따라서 띄어쓰기 2칸을 필수로 적어주어야 합니다.
> 예를 들어서 아래의 `datasource` 는 `spring:` 하위에 있고 앞에 띄어쓰기 2칸이 있으므로 `spring.datasource` 가 됩니다. 다음 코드에 주석으로 띄어쓰기를 적어두었습니다.

yml 띄어쓰기 주의

```yaml
spring: #띄어쓰기 없음
  datasource: #띄어쓰기 2칸
    url: jdbc:h2:tcp://localhost/~/jpashop #4칸
    username: sa
```

```
    password:
    driver-class-name: org.h2.Driver

  jpa: #띄어쓰기 2칸
    hibernate: #띄어쓰기 4칸
      ddl-auto: create #띄어쓰기 6칸
    properties: #띄어쓰기 4칸
      hibernate: #띄어쓰기 6칸
#         show_sql: true    #띄어쓰기 8칸
        format_sql: true  #띄어쓰기 8칸

logging.level: #띄어쓰기 없음
  org.hibernate.SQL: debug #띄어쓰기 2칸
#  org.hibernate.type: trace #띄어쓰기 2칸
```

실제 동작하는지 확인하기

회원 엔티티

```
@Entity
@Getter @Setter
public class Member {

    @Id @GeneratedValue
    private Long id;
    private String username;
    ...
}
```

Id->식별자
GeneratedValue->자동생성

회원 리포지토리

```
@Repository
public class MemberRepository {

    @PersistenceContext
```

스프링빈에 등록

이 애노테이션이 있으면 스프링 부트가
em을 알아서 주입해준다.

우리는 그냥 쓰기만 하면됨.

```
    EntityManager em;

    public Long save(Member member) {
        em.persist(member);
        return member.getId();
    }

    public Member find(Long id) {
        return em.find(Member.class, id);
    }
}
```

김영한식st-(return member를 하지않은 이유)보통 저장하고 return값을 안줌.쿼리와 커맨드를 분리?? 하지만 id정도는 괜찮 ㅇㅇ;

테스트

```
import jpabook.jpashop.domain.Member;
import jpabook.jpashop.repository.MemberRepository;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.context.junit4.SpringRunner;
import org.springframework.transaction.annotation.Transactional;
import javax.persistence.EntityManager;

@RunWith(SpringRunner.class)
@SpringBootTest
public class MemberRepositoryTest {

    @Autowired MemberRepository memberRepository;

    @Test
    @Transactional
    @Rollback(false)
    public void testMember() {
        Member member = new Member();
        member.setUsername("memberA");
        Long savedId = memberRepository.save(member);
```

Test에 Transactional있으면 롤백됨 test가 아니면 정상작동

junit한테 spring관련된것을 테스트 할 것이라는 것을 알려줌.

Sibal SpringBootTest말고 SpringBootApplication햇다가 두시간 뻘짓함

트랜잭션은 스프링것을 권장.

```
        Member findMember = memberRepository.find(savedId);

        Assertions.assertThat(findMember.getId()).isEqualTo(member.getId());

Assertions.assertThat(findMember.getUsername()).isEqualTo(member.getUsername())
;
        Assertions.assertThat(findMember).isEqualTo(member); //JPA 엔티티 동일성 보
장
    }
}
```

참고 - 엔티티라는 용어는 때로는 클래스를 의미하는 경우도 있고, 클래스에 의해 생성된 인스턴스를 의미하는 경우가 있습니다.

같은 트랜잭션안에서 저장하고 조회하면 영속성 컨텍스트가 똑같음.같은 영속성 컨텍스트안에서는 id값이 같으면 같은 엔티티로 인식함.

아니 같은 엔티티인스턴스라고 하는게 맞는거아니냐 헷갈려..

> **주의!** @Test는 JUnit4를 사용하면 org.junit.Test를 사용하셔야 합니다. 만약 JUnit5 버전을 사용하면 그것에 맞게 사용하시면 됩니다.

- Entity, Repository 동작 확인
- jar 빌드해서 동작 확인

오류이유 - em을 통한 모든 데이터 변경은 트랜잭션안에서 이루어져야 한다.하지만 테스트클래스에는 트랜잭션이 없다.

> 오류: 테스트를 실행했는데 다음과 같이 테스트를 찾을 수 없는 오류가 발생하는 경우
> `No tests found for given includes: [jpabook.jpashop.MemberRepositoryTest] (filter.includeTestsMatching)`
> 해결: 스프링 부트 2.1.x 버전을 사용하지 않고, 2.2.x 이상 버전을 사용하면 Junit5가 설치된다. 이때는 `build.gradle` 마지막에 다음 내용을 추가하면 테스트를 인식할 수 있다. Junit5 부터는 `build.gradle` 에 다음 내용을 추가해야 테스트가 인식된다.

`build.gradle` 마지막에 추가

```
test {
    useJUnitPlatform()
}
```

cmd에서 db실행하면서 다른명령어를 사용하는 단축키좀 찾자.

> 참고: 스프링 부트를 통해 복잡한 설정이 다 자동화 되었다. `persistence.xml` 도 없고, `LocalContainerEntityManagerFactoryBean` 도 없다. 스프링 부트를 통한 추가 설정은 스프링 부트 메뉴얼을 참고하고, 스프링 부트를 사용하지 않고 순수 스프링과 JPA 설정 방법은 자바 ORM 표준 JPA 프

> 로그래밍 책을 참고하자.

어마어마한 꿀팁

쿼리 파라미터 로그 남기기

- 로그에 다음을 추가하기 `org.hibernate.type` : SQL 실행 파라미터를 로그로 남긴다.
- 외부 라이브러리 사용
  - https://github.com/gavlyukovskiy/spring-boot-data-source-decorator

스프링 부트를 사용하면 이 라이브러리만 추가하면 된다.

```
implementation 'com.github.gavlyukovskiy:p6spy-spring-boot-starter:1.5.6'
```

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

도메인 분석 설계

요구사항 분석