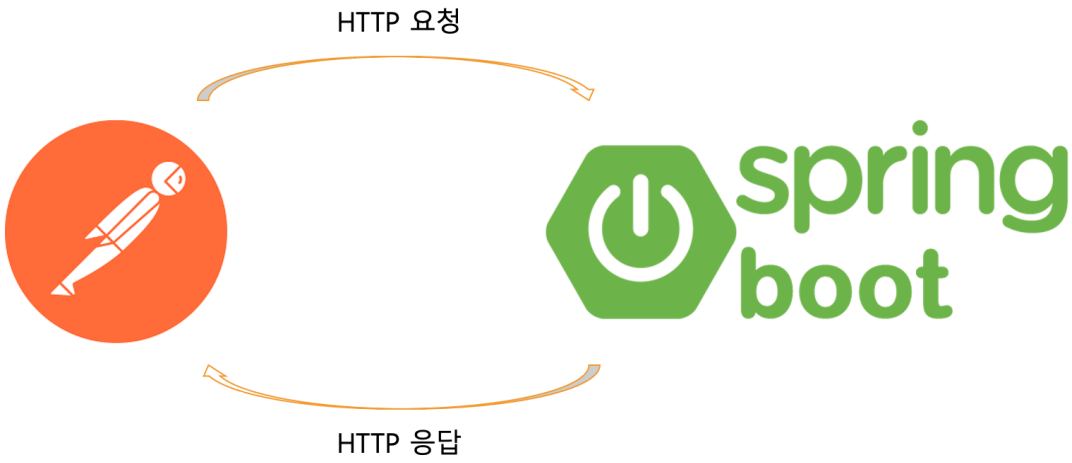
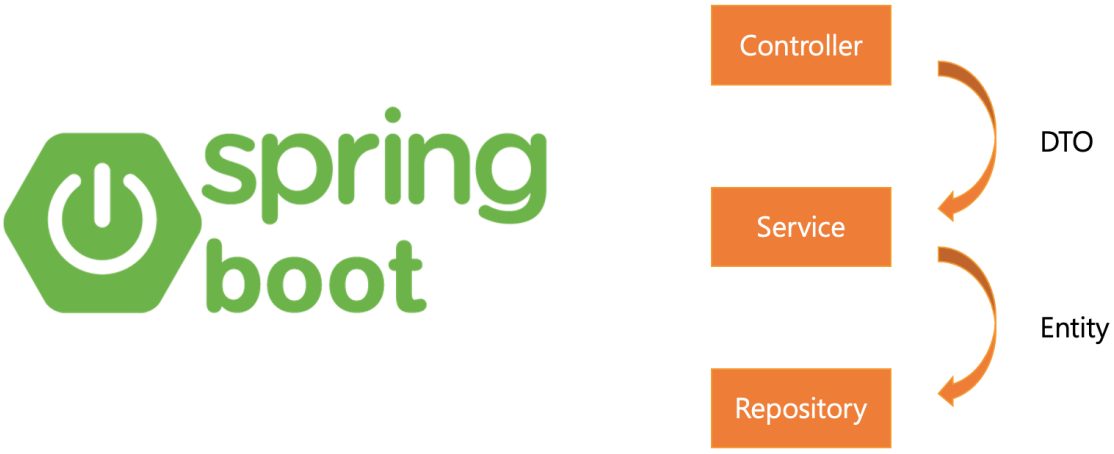
Chapter 7-3 Spring Boot Test

Testing에 대하여
Test Driven Development
[@Bean, @Configuration]
[@Component]
JPA 영속성 컨텍스트의 쿼리 실행 방식
JUnit에서의 @Transactional

Testing에 대하여



산출물의 기능적 측면의 테스트



개별 코드 단위의 테스트

단위 테스트(Unit Test) : 응용 프로그램에서 테스트 가능한 가장 작은 소프트웨어를 실행하여 예상대로 동작하는지 확인하는 테스트

- 클래스 각각의 함수들이 잘 작동하는지!

통합 테스트(Integration Test) : 단위 테스트보다 더 큰 동작을 달성하기 위해 여러 모듈들을 모아 이들이 의도대로 협력하는지 확인하는 테스트

- 클래스들이 서로 상호작용을 잘 하는 지

인수 테스트(Acceptance Test) : 사용자 스토리(시나리오)에 맞춰 수행하는 테스트

```
testImplementation 'org.springframework.boot:spring-boot-starter-test'
```

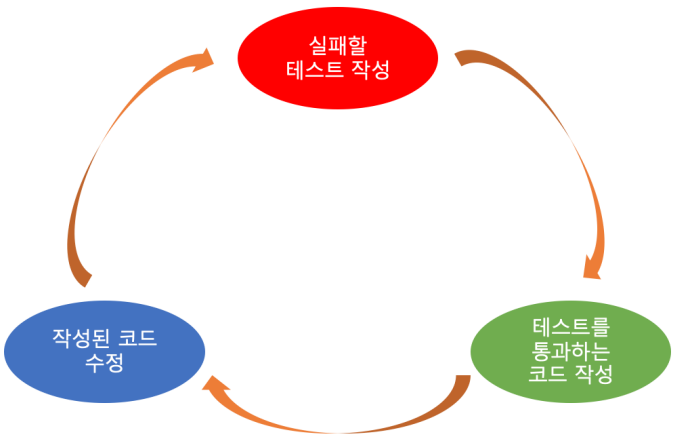
- JUnit: 사실상의(de-facto) Java 어플리케이션 Testing 표준 라이브러리
- Spring Test: Spring 어플리케이션 Test 지원 라이브러리
- AssertJ: 가독성 높은 Test 작성을 위한 라이브러리
- Hamcrest: Test 진행시 제약사항 설정을 위한 라이브러리
- Mockito: Test용 Mock 라이브러리
- JSONassert: JSON용 Assertion 라이브러리
- JsonPath: JSON 데이터 확인용 라이브러리

Test Driven Development

Test Driven Development

테스트 주도 개발

실제 작동하는 코드 이전에 통과해야할 테스트를 우선 만드는 개발 방식



1. 테스트 작성
2. 테스트 실행 및 실패
3. 테스트를 통과하는 코드 작성
4. 테스트 통과 확인
5. 코드 정리 (리펙토링)

`@Configuration`

[@Bean, @Configuration]

- 개발자가 직접 제어가 불가능한 외부 라이브러리 또는 설정을 위한 클래스를 Bean으로 등록할 때 @Bean 어노테이션을 활용
- 1개 이상의 @Bean을 제공하는 클래스의 경우 반드시 @Configuration을 명시해 주어야 함

[@Component]

- 개발자가 직접 개발한 클래스를 Bean으로 등록하고자 하는 경우 @Component 어노테이션을 활용

`@EnableJpaAuditing`

*@SpringBootTest가 아닌 `@RunWith(SpringRunner.class)`를 사용하는 이유 ?

→ @SpringBootTest를 사용하면 application context를 전부 로딩해서 자칫 잘못하면 무거운 프로젝트로서의 역할을 할 수 있음

`@RunWith(SpringRunner.class)`를 사용하면 @Autowired, @MockBean 에 해당되는 것들만 application context를 로딩하게 되므로 Junit4에서는 필요한 조건에 맞춰서 사용!

`@RunWith(SpringRunner.class)`

: SpringRunner에 대한 별칭으로 SpringJUnit4ClassRunner, JUnit 테스트 라이브러리를 SpringTestContext Framework와 결합한다. 결합한 이것을 `@RunWith(SpringRunner.class)`라고 일컫는다.

`@WebMvcTest(PostController.class)`

- MVC를 위한 테스트, 컨트롤러가 예상대로 동작하는지 테스트하는데 사용된다.

(To test whether Spring MVC controllers are working as expected, use the @WebMvcTest annotation.)

- @WebMvcTest 어노테이션을 사용시 다음 내용만 스캔 하도록 제한한다. (보다 가벼운 테스팅이 가능하다.)

@Controller, @ControllerAdvice, @JsonComponent, Converter, GenericConverter, Filter, HandlerInterceptor, WebMvcConfigure

- MockBean, MockMVC를 자동 구성하여 테스트 가능하도록 한다.

- Spring Security의 테스트도 지원 한다.

- @WebMvcTest를 사용하기 위해 테스트할 특정 컨트롤러 클래스를 명시 하도록 한다.

● 장점

- WebApplication 관련된 Bean들만 등록하기 때문에 통합 테스트보다 빠르다.

- 통합 테스트를 진행하기 어려운 테스트를 진행 가능하다.

ex) 결제 모듈 API를 콜하며 안되는 상황에서 Mock을 통해 가짜 객체를 만들어 테스트 가능.

● 단점

- 요청부터 응답까지 모든 테스트를 Mock 기반으로 테스트하기 때문에 실제 환경에서는 제대로 동작하지 않을 수 있다.

`@SpringBootTest` : insert query가 실행

`@DataJpaTest` : insert query가 실행되지 않음

JPA 영속성 컨텍스트의 쿼리 실행 방식

- 영속성 컨텍스트는 `flush()`가 실행되기 전까지 실행될 쿼리를 가지고 있다가 한번에 쿼리를 실행하도록 설계되어있습니다.
- `flush()`가 실행되기 전까지 쿼리가 실행되지 않는다는 것은 `@Transactional`안에서 Rollback이 되어야 하는 상황이라면 애초에 쿼리가 실행조차 되지 않을수 있다는 것입니다.

JUnit에서의 @Transactional

- JUnit을 통한 Test가 실행된다면 `@Transactional`이 적용되어있는 로직이 실행된다면 SELECT를 제외한 모든 쿼리는 Rollback 대상으로 취급합니다.

`@DataJpaTest`
로 실행한 테스트는 `@Transacitonal`
이 자동으로 설정

플러시 (Flush)
영속성 컨텍스트의 변경 내용을 DB 에 반영하는 것을 말한다.
Transaction commit 이 일어날 때 flush가 동작하는데, 이때 쓰기 지연 저장소에 쌓아 놨던 INSERT, UPDATE, DELETE SQL들이 DB에 날라간다.
주의! 영속성 컨텍스트를 비우는 것이 아니다.
쉽게 얘기해서 영속성 컨텍스트의 변경 사항들과 DB의 상태를 맞추는 작업이다.
플러시는 영속성 컨텍스트의 변경 내용을 DB에 동기화한다.

- 플러시의 동작 과정
  1. 변경을 감지한다. (Dirty Checking)
  2. 수정된 Entity를 쓰기 지연 SQL 저장소에 등록한다.
  3. 쓰기 지연 SQL 저장소의 Query를 DB에 전송한다. (등록, 수정, 삭제 Query)
  - flish가 발생한다고 해서 commit이 이루어지는 것이 아니고 flush 다음에 실제 commit이 일어난다.
  - 플러시가 동작할 수 있는 이유는 데이터베이스 트랜잭션(작업 단위)이라는 개념이 있기 때문이다.
  트랜잭션이 시작되고 해당 트랜잭션이 commit 되는 시점 직전에만 동기화 (변경 내용을 날림) 해주면 되기 때문에, 그 사이에서 플러시 매커니즘의 동작이 가능한 것이다.
  - [참고] JPA는 기본적으로 데이터를 맞추거나 동시성에 관련된 것들은 데이터베이스 트랜잭션에 위임한다.

테스트케이스 작성

`given` 어떤 데이터가 준비되어있다.

PostEntity가 존재할 때

`when` 어떤 행위가 일어났을 때 (함수 호출 등)

경로에 Get 요청이 오면

`then` 어떤 결과가 올 것인지

PostDto가 반환된다

```
@RunWith(SpringRunner.class)
@SpringBootTest(
        webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT,
        classes = JpaApplication.class
)
@AutoConfigureMockMvc
@EnableAutoConfiguration(exclude = SecurityAutoConfiguration.class)
@AutoConfigureTestDatabase
```