Querydsl 적용하기

날짜	
상태	문서
설명	
유형	
작성일시	@2021년 4월 19일 오전 9:15
종료일	

👉 Querydsl이란 JPQL의 빌더 역할을 하는 오픈 소스이다.

사용 이유

1) type-safe

Querydsl은 도메인 모델을 QClass라고하는 생성된 typesafe 클래스에 매핑한다. 따라서 입력되는 값의 타입을 올바르게 매핑 할 수 있다.

2) 잘못된 쿼리를 컴파일 이전 단계에서 알 수 있다

xml을 이용한 쿼리를 작성했을 때 잘못된 쿼리를 build해도 에러가 발생되지 않고, 실제로 그 쿼리를 호출해야만 쿼리의 에러 여부를 알 수 있다. Querydsl은 이런 문제를 사전에 방지 할 수 있다.

```
--XML
SELECTt from 잘못된테이블명!! --에러가 발생하지 않는다
```

3) 이넘, CONSTANT를 이용해 type이 의미하는 바를 한눈에 알 수 있다.

```
--XML
select * from store where type = 1 -- type이 의미하는 걸 알기 어렵다
--Querydsl
selectFrom(store).where(store.type.eq(storeConstant.영업중)) --type이 의미하는 바를 알 수 있다.
--Querydsl
selectFrom(store).where(store.type.eq(new Integer(1))) --type이 Integer타입인걸 알 수 있다.
```

4) 실제 SQL문과 유사하다.

조회 쿼리를 사용할 때 기존에 프로젝트에서 사용하던 Specification에 비하면 코드를 읽기와 쓰기 모두 쉬워진 걸 알 수 있다.

아래의 예제 코드를 보고 비교해보자.

출처 : https://spring.io/blog/2011/04/26/advanced-spring-data-jpa-specifications-and-querydsl/

- Specification을 사용 했을 때 코드

```
//Specification
public CustomerSpecifications {

  public static Specification<Customer> customerHasBirthday() {
    return new Specification<Customer> {
      public Predicate toPredicate(Root<T> root, CriteriaQuery query, CriteriaBuilder cb) {
        return cb.equal(root.get(Customer_.birthday), today);
      }
    };
```

```
  }

  public static Specification<Customer> isLongTermCustomer() {
    return new Specification<Customer> {
      public Predicate toPredicate(Root<T> root, CriteriaQuery query, CriteriaBuilder cb) {
        return cb.lessThan(root.get(Customer_.createdAt), new LocalDate.minusYears(2));
      }
    };
  }
}
```

- Querydsl을 사용 했을 때 코드

```
//Querydsl
QCustomer customer = QCustomer.customer;
LocalDate today = new LocalDate();
BooleanExpression customerHasBirthday = customer.birthday.eq(today);
BooleanExpression isLongTermCustomer = customer.createdAt.lt(today.minusYears(2));
```

사용법

0. build.gradle gradle 버젼 6.7 세팅

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

group 'com.start.pilotproject'
version '1.0.4-SNAPSHOT-'
sourceCompatibility = 11
configurations {
    compileOnly {
        extendsFrom annotationProcessor
    }
}

// querydsl 적용
def generated='src/main/generated'
sourceSets {
    main.java.srcDirs += [ generated ]
}

tasks.withType(JavaCompile) {
    options.annotationProcessorGeneratedSourcesDirectory = file(generated)
}

clean.doLast {
    file(generated).deleteDir()
}

repositories {
    mavenCentral()
    jcenter()
}


dependencies {
    implementation('org.springframework.boot:spring-boot-starter-web')
    implementation('org.springframework.boot:spring-boot-starter-thymeleaf')
    // lombok
    implementation('org.projectlombok:lombok')
    annotationProcessor('org.projectlombok:lombok')
    testImplementation('org.projectlombok:lombok')
    testAnnotationProcessor('org.projectlombok:lombok')

    implementation('com.oracle.ojdbc:ojdbc8')
    implementation('org.springframework.boot:spring-boot-starter-data-jpa')

    //querydsl
    implementation("com.querydsl:querydsl-jpa")
    implementation("com.querydsl:querydsl-core")
    annotationProcessor("com.querydsl:querydsl-apt:${dependencyManagement.importedProperties['querydsl.version']}:jpa") // querydsl JP
    annotationProcessor("jakarta.persistence:jakarta.persistence-api:2.2.3")
    annotationProcessor("jakarta.annotation:jakarta.annotation-api:1.3.5")
```

```
    //h2
    implementation("com.h2database:h2")
    testImplementation("com.h2database:h2")


    testImplementation('org.springframework.boot:spring-boot-starter-test')
}

test {
    useJUnitPlatformm()
}
```

1. Entity를 생성합니다.

```
@Getter
@NoArgsConstructor
@Entity
public class Customer {
    private String firstName;
    private String lastName;
}
```

2. Gradle에서 등록한 Task를 통해 querydsl에 사용할 Qclass를 생성합니다.
    - 설정마다 다르겠지만 저희는 현재 domain project를 compile시 Qclass들이 generated폴더에 생성되게 설정 되어있고 clean시 generated의 파일들이 지워집니다.
3. Repository를 사용합니다
    - 사용방법은 여러가지가 있지만 상속/구현이 없는 Querydsl만을 이용한 Repository구성이 좋아보입니다
    - Qclass를 import 하여 사용합니다
    - 객체를 생성하여 사용할수도 있습니다.

```
import static com.valuelinku.domain.example.QCustomer.customer;//QClass import

@RequiredArgsConstructor
@Repository
public class CustomerQueryRepository {
    private final JPAQueryFactory queryFactory;
    //QCustomer customer = QCustomer.customer; QClass생성
    public List<Customer> findByFirstName(String firstName) {
        return queryFactory.selectFrom(customer)
                .where(customer.firstName.eq(firstName))
                .fetch();
    }
}
```

4. 조회 결과의 반환에는 .fetch()가 쓰입니다

```
fetch()  - > list 반환
fetchOne()  -> 단건 반환. 결과가 없으면 null 반환하고, 단건 이상이면 nonUniqueException 발생
fetchFirst() -> 단건 반환. 여러건이 나올때는 처음 데이터 반환. 결과 없으면 null
fetchResults() -> QueryResults 반환. 페이징처리를 위한 함수
fetchCount() -> row count 반환
```