스프링부트로 RestFulAPI 구현하기

11장 JPA 심화 2 Querydsl

- Querydsl

박명회

11장 Querydsl

Querydsl 장점

- 고정된 SQL문이 아닌 조건에 맞게 동적으로 쿼리를 생성할수 있습니다.
- 비슷한 쿼리를 재사용할 수 있으며 제약 조건 조립 및 가독성을 향상시킬수 있습니다.
- 문자열이 아닌 자바 소스 코드로 작성하기 때문에 컴파일 시점에 오류를 발견할 수 있습니다.
- IDE 도움을 받아서 자동완성 기능을 이용할 수 있어 생산성 향상에 도움이 됩니다.

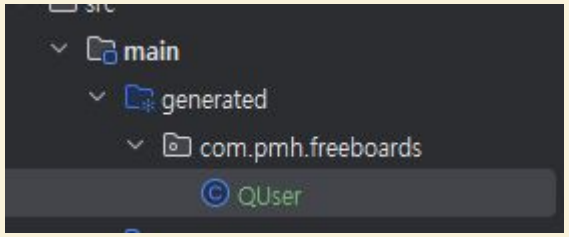
Querydsl을 사용하기 위해서는 의존성을 추가 해야 합니다.

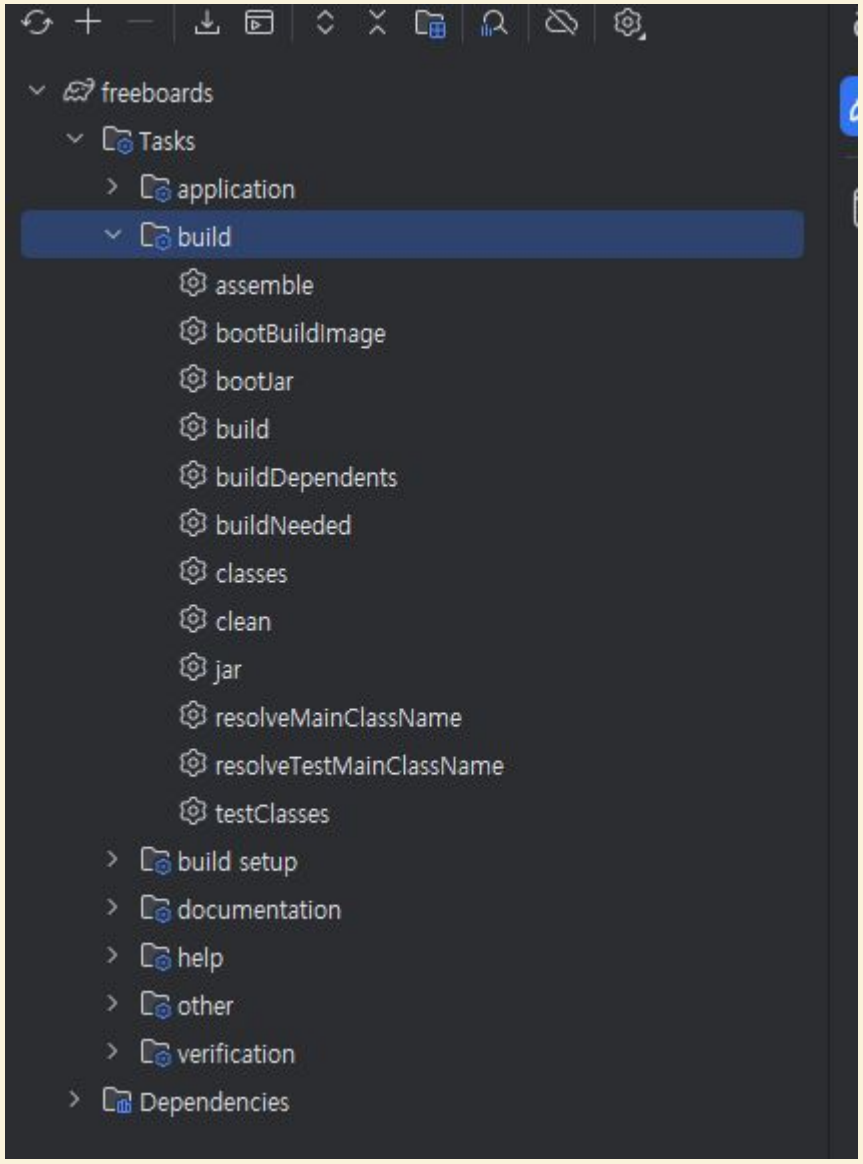
```
//querydsl 설정 추가
    implementation 'com.querydsl:querydsl-jpa:5.0.0:jakarta'
    annotationProcessor "com.querydsl:querydsl-apt:5.0.0:jakarta"
    annotationProcessor "jakarta.annotation:jakarta.annotation-api"
    annotationProcessor "jakarta.persistence:jakarta.persistence-api"
```

11장 build.gradle 수정

```
// Querydsl 설정부
def generated = 'src/main/generated'

// querydsl QClass 파일 생성 위치를 지정
tasks.withType(JavaCompile) {
    options.getGeneratedSourceOutputDirectory().set(file(generated))
}

// java source set 에 querydsl QClass 위치 추가
sourceSets {
    main.java.srcDirs += [ generated ]
}

// gradle clean 시에 QClass 디렉토리 삭제
clean {
    delete file(generated)
}
```

11장 build.gradle 수정





#클린시 Q클래스 삭제
gradle clean

#실행시 Q클래스 생성
gradle build

#폴더에 Q클래스 생성
src/main/generated

11장 JPAQueryFactory 객체 주입

```
import com.querydsl.jpa.impl.JPAQueryFactory;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;

@Configuration
public class JpaConfig {

    @PersistenceContext
    private EntityManager entityManager;

    @Bean
    public JPAQueryFactory jpaQueryFactory() {
        return new JPAQueryFactory(entityManager);
    }
}
```

11장 QueryFactory 테스트 코드

```
@Test
public void testFindUsersByName() {
    // given
    QUser qUser = QUser.user;

    // when
    List<User> foundUsers = queryFactory.selectFrom(qUser)
                                .where(qUser.name.contains("John"))
                                .fetch();

    // then
    assertThat(foundUsers).hasSize(2);
    assertThat(foundUsers.get(0).getName()).contains("John");
    assertThat(foundUsers.get(1).getName()).contains("John");
}
```

11장 Queryldsql 함수

키워드	설명
List<T> fetch()	조회 결과 리스트 반환
T fetchOne()	조회 대상이 1건인 경우 제네릭으로 지정한 타입 반환
T fetchFirst()	조회대상중 1건만 반환
Long FetchCount()	조회대상 개수 반환

11장 querydslPredicateExecutor 상속

```
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.querydsl.QuerydslPredicateExecutor;

public interface UserRepository extends JpaRepository<User, Long>, QuerydslPredicateExecutor<User> {
}
```

키워드	설명
long count(Predicate)	조건에 맞는 데이터의 총 개수 반환
boolean exists(Predicate)	조건에 맞는 데이터 존재 여부 반환
Page<T> findAll(Predicate,pageable)	조건에 맞는 페이지 데이터 반환
T findOne(Predicate)	조건에 맞는 데이터 1개 반환

11장 querydslPredicateExecutor 테스트 코드

```
@Test
public void testFindUsersByNameUsingPredicateExecutor() {
    // given
    QUser qUser = QUser.user;
    BooleanExpression predicate = qUser.name.contains("John");

    // when
    Iterable<User> foundUsers = userRepository.findAll(predicate);

    // then
    assertThat(foundUsers).hasSize(2);
    foundUsers.forEach(user -> assertThat(user.getName()).contains("John"));
}
```

11장 querydslPredicateExecutor 테스트 코드

```
@Test
public void testFindUsersByEmailAndAge() {
    // given
    QUser qUser = QUser.user;
    BooleanExpression predicate = qUser.email.endsWith("@example.com")
                                  .and(qUser.age.goe(30));

    // when
    Iterable<User> foundUsers = userRepository.findAll(predicate);

    // then
    assertThat(foundUsers).hasSize(2);
    foundUsers.forEach(user -> {
        assertThat(user.getEmail()).endsWith("@example.com");
        assertThat(user.getAge()).isGreaterThanOrEqualTo(30);
    });
}
```

11장 querydslPredicateExecutor 테스트 코드

```
    @Test
    public void testFindUsersByUsernameOrEmail() {
        // given
        QUser qUser = QUser.user;
        BooleanExpression predicate = qUser.username.eq("johnsmith")
                                      .or(qUser.email.eq("johnathan@example.com"));

        // when
        Iterable<User> foundUsers = userRepository.findAll(predicate);

        // then
        assertThat(foundUsers).hasSize(2);
        foundUsers.forEach(user -> {
            assertThat(user.getUsername().equals("johnsmith") ||
user.getEmail().equals("johnathan@example.com")).isTrue();
        });
    }
```

11장 querydslPredicateExecutor 테스트 코드

```
@Test
public void testFindUsersByAgeBetween() {
    // given
    QUser qUser = QUser.user;
    BooleanExpression predicate = qUser.age.between(20, 30);

    // when
    Iterable<User> foundUsers = userRepository.findAll(predicate);

    // then
    assertThat(foundUsers).hasSize(2);
    foundUsers.forEach(user -> assertThat(user.getAge()).isBetween(20, 30));
}
```

11장 querydslPredicateExecutor 테스트 코드

```
@Test
public void testFindUsersByNameUsingPredicateExecutorWithPaging() {
    // given
    QUser qUser = QUser.user;
    BooleanExpression predicate = qUser.name.contains("길동");
    Pageable pageable = PageRequest.of(0, 2); // 첫 번째 페이지, 페이지 당 2개의 항목

    // when
    Page<User> foundUsersPage = userRepository.findAll(predicate, pageable);
    List<User> foundUsers = foundUsersPage.getContent();

    // then
    assertThat(foundUsersPage.getTotalPages()).isEqualTo(2);
    assertThat(foundUsersPage.getTotalElements()).isEqualTo(3);
    assertThat(foundUsers).hasSize(2);
    foundUsers.forEach(user -> assertThat(user.getName()).contains("길동"));
}
```

감사합니다