JPA 에서 QueryDSL 사용하기위해 JPAQuery 인스턴스 생성방법

http://ojc.asia, http://ojcedu.com

1. JPAQuery 를 직접 생성하기

■ **JPAQuery 인스턴스 생성하기**

QueryDSL의 JPAQuery API를 사용하려면 JPAQuery 인스턴스를 생성하면 된다.

```
// entityManager는 JPA의 EntityManage
JPAQuery<?> query = new JPAQuery<Void>(entityManager);
```

2. JPAQueryFactory 를 이용한 JPAQuery 인스턴스 생성

■ QueryDSL을 이용하여 JPA에서 쿼리를 사용한다면 JPAQueryFactory를 이용하면 되는데, JPAQueryFactory는 JPQLQueryFactory 인터페이스를 구현했으며, JPAQuery 인스턴스를 포함하여 다양한 방법으로 쿼리 할 수 있다.

■ JPAQueryFactory의 모습은 아래와 같다. 생성할 때 EntityManager만 인자로 넣어 생성할 수도 있고, JPQLTemplate도 같이 인자로 줘서 생성 할 수 있다.

```
package com.querydsl.jpa.impl;

import javax.annotation.Nullable;
import javax.inject.Provider;
import javax.persistence.EntityManager;

import com.querydsl.core.Tuple;
import com.querydsl.core.types.EntityPath;
import com.querydsl.core.types.Expression;
import com.querydsl.core.types.dsl.Expressions;
import com.querydsl.jpa.JPQLQueryFactory;
import com.querydsl.jpa.JPQLTemplates;

/**
```

```
 * Factory class for query and DML clause creation
 *
 * @author tiwe
 *
 */
public class JPAQueryFactory implements JPQLQueryFactory  {

    @Nullable
    private final JPQLTemplates templates;

    private final Provider<EntityManager> entityManager;

    public JPAQueryFactory(final EntityManager entityManager) {
        this.entityManager = new Provider<EntityManager>() {
            @Override
            public EntityManager get() {
                return entityManager;
            }
        };
        this.templates = null;
    }

    public JPAQueryFactory(JPQLTemplates templates, final EntityManager entityManager) {
        this.entityManager = new Provider<EntityManager>() {
            @Override
            public EntityManager get() {
                return entityManager;
            }
        };
        this.templates = templates;
    }

    public JPAQueryFactory(Provider<EntityManager> entityManager) {
        this.entityManager = entityManager;
        this.templates = null;
    }

    public JPAQueryFactory(JPQLTemplates templates, Provider<EntityManager> entityManager) {
        this.entityManager = entityManager;
```

```
        this.templates = templates;
    }

    @Override
    public JPADeleteClause delete(EntityPath<?> path) {
        if (templates != null) {
            return new JPADeleteClause(entityManager.get(), path, templates);
        } else {
            return new JPADeleteClause(entityManager.get(), path);
        }
    }

    @Override
    public <T> JPAQuery<T> select(Expression<T> expr) {
        return query().select(expr);
    }

    @Override
    public JPAQuery<Tuple> select(Expression<?>... exprs) {
        return query().select(exprs);
    }

    @Override
    public <T> JPAQuery<T> selectDistinct(Expression<T> expr) {
        return select(expr).distinct();
    }

    @Override
    public JPAQuery<Tuple> selectDistinct(Expression<?>... exprs) {
        return select(exprs).distinct();
    }

    @Override
    public JPAQuery<Integer> selectOne() {
        return select(Expressions.ONE);
    }

    @Override
    public JPAQuery<Integer> selectZero() {
```

```
        return select(Expressions.ZERO);
    }

    @Override
    public <T> JPAQuery<T> selectFrom(EntityPath<T> from) {
        return select(from).from(from);
    }

    @Override
    public JPAQuery<?> from(EntityPath<?> from) {
        return query().from(from);
    }

    @Override
    public JPAQuery<?> from(EntityPath<?>... from) {
        return query().from(from);
    }

    @Override
    public JPAUpdateClause update(EntityPath<?> path) {
        if (templates != null) {
            return new JPAUpdateClause(entityManager.get(), path, templates);
        } else {
            return new JPAUpdateClause(entityManager.get(), path);
        }
    }

    @Override
    public JPAQuery<?> query() {
        if (templates != null) {
            return new JPAQuery<Void>(entityManager.get(), templates);
        } else {
            return new JPAQuery<Void>(entityManager.get());
        }
    }

}
```

■ 스프링 설정파일에서 @Bean으로 빈을 등록해야 하는데 스프링 부트인 경우 메인에서 다음

과 같이 빈으로 등록하면 된다.

```
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;
import com.querydsl.jpa.JPQLTemplates;
import com.querydsl.jpa.impl.JPAQueryFactory;

@SpringBootApplication
public class JpaqueryfactoryExamApplication {

	public static void main(String[] args) {
		SpringApplication.run(JpaqueryfactoryExamApplication.class, args);
	}

	@PersistenceContext
	EntityManager em;

	@Bean
	public JPAQueryFactory queryFactory() {
		//return new JPAQueryFactory(JPQLTemplates.DEFAULT, em);
		return new JPAQueryFactory(em);
	}
}
```

■ Repository 구현체에서는 다음과 같이 빈을 주입받아 사용하면 된다.

```
	@Autowired
	JPAQueryFactory queryFactory;

	/* Emp 테이블에서 job을 조건으로 이름 내림차순으로 검색 */
	public List<Emp> selectByJobOrderByEnameDesc(String job) {

		List<Emp> emps = queryFactory.selectFrom(emp)
					.where(emp.job.eq(job))
					.orderBy(emp.ename.desc()).fetch();
		return emps;
```

```
        }
```