@Query 사용 방법

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

**영속성 컨텍스트의 개념과 이점**

영속성 관리

JPA 에서 가장 중요한 2가지

* 객체와 관계형 DB 매핑하기 (Object Relational Mapping) - 설계 관련
* 영속성 컨텍스트 - JPA 내부 동작

EntityManagerFactory와 EntityManager

* EntityManagerFactory는 고객의 요청이 올 때마다 (thread가 하나 생성될 때마다) EntityManager를 생성한다.
* EntityManager는 내부적으로 DB connection pool을 사용해서 DB에 접근한다.

EntityManagerFactory

* JPA 는 EntityManagerFactory 를 만들어야 한다.
* application loading 시점에 DB 당 딱 하나만 생성되어야 한다. EntityManagerFactory entityManagerFactory = Persistence.createEntityManagerFactory("hello");
* entityManagerFactory.close();

WAS 가 종료되는 시점에 EntityManagerFactory 를 닫는다.

그래야 내부적으로 Connection pooling 에 대한 Resource 가 Release 된다.

EntityManager

* 실제 Transaction 단위를 수행할 때마다 생성한다.
* 즉, 고객의 요청이 올 때마다 사용했다가 닫는다.
* thread 간에 공유하면 안된다. (사용하고 버려야 한다.)
* entityManager.close();

Transaction 수행 후에는 반드시 EntityManager 를 닫는다.

그래야 내부적으로 DB Connection 을 반환한다.

EntityTransaction

* Data 를 “변경”하는 모든 작업은 반드시 Transaction 안에서 이루어져야 한다.

단순한 조회의 경우는 상관없음.

* EntityTransaction tx = entityManager.getTransaction();
* tx.begin(); : Transaction 시작
* tx.commit(); : Transaction 수행
* tx.rollback(); : 작업에 문제가 생겼을 시

**영속성 컨텍스트(Persistence Context)란**

JPA를 이해하는데 가장 중요한 용어이다. 논리적인 개념으로, 눈에 보이지 않는다.

* “Entity를 영구 저장하는 환경”이라는 뜻
* EntityManager.persist(entity); 의 동작 설명
* 실제로는 DB에 저장하는 것이 아니라 영속성 컨텍스트를 통해서 Entity를 영속화한다는 뜻이다.
* 정확히 말하면 persist() 시점에는 Entity를 영속성 컨텍스트에 저장하는 것이다.
* EntityManager를 통해서 영속성 컨텍스트에 접근한다.
* EntityManager가 생성되면 1:1로 영속성 컨텍스트가 생성된다.

**엔티티의 생명주기 (Entity LifeCycle)**

* **비영속(new/transient)**

영속성 컨텍스트와 전혀 관계가 없는 상태

객체를 생성’만’ 한 상태

텍스트, 스크린샷, 폰트, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

* **영속(managed)**

영속성 컨텍스트에 저장된 상태

Entity가 영속성 컨텍스트에 의해 관리되는 상태

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

EntityManager.persist(entity);

영속 상태가 된다고 바로 DB에 쿼리가 날라가지 않는다. (즉, DB 저장 X)

transaction.commit();

트랜잭션의 commit 시점에 영속성 컨텍스트에 있는 정보들이 DB에 쿼리로 날라간다.

* 준영속(detached)

영속성 컨텍스트에 저장되었다가 분리된 상태

영속성 컨텍스트에서 지운 상태

텍스트, 폰트, 스크린샷, 라인이(가) 표시된 사진

자동 생성된 설명

* 준영속(detached)

영속성 컨텍스트에 저장되었다가 분리된 상태

* 삭제(Removed)

영속성 컨텍스트에서 지운 상태

텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명

**엔티티의 생명주기**

도표, 텍스트, 원, 라인이(가) 표시된 사진

자동 생성된 설명

비영속(new/transient) : 영속성 컨텍스트와 전혀 관계가 없는 새로운 상태

영속(managed) : 영속성 컨텍스트에 관리되는 상태

준영속(detached) : 영속성 컨텍스트에 저장되었다가 분리된 상태

삭제(remove) : 삭제된 상태

**OSIV(Open Session In View)**

JPA에서는 OEIV(Open EntityManager In View), 하이버네이트에선 OSIV(Open Session In View)라고 한다. 하지만 관례상 둘 다 OSIV로 부른다.

**OSIV 동작 원리**

OSIV의 동작 방식에 대해서 Spring Framework가 제공하는 OSIV을 통해 알아보겠다.

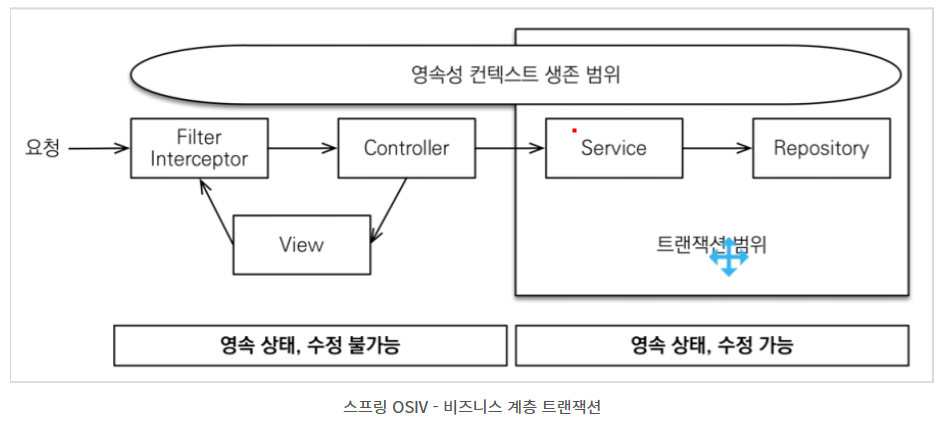
스프링이 제공하는 OSIV 클래스는 서블릿 필터에서 적용할지 스프링 인터셉터에서 적용할지에 따라 원하는 클래스를 선택해서 사용하면 된다.

* 필터 : org.springframework.orm.jpa.support.OpenEntityManagerInViewFilter
* 인터셉터 : org.springframework.orm.jpa.support.OpenEntityManagerInViewInterceptor

기본적으로 Spring Boot에서는 spring.jpa.open-in-view라는 설정을 통해 Open Session In View를 설정하고 있다. 이 값이 true이면 OpenEntityManagerInViewInterceptor를 인터셉터로 등록하여 Open Session In View를 적용하는 것을 알 수 있다.

스프링 프레임워크가 제공하는 OSIV는 비즈니스 계층에서 트랜잭션을 사용하는 OSIV다.

영속성 컨텍스트는 사용자의 요청 시점에서 생성이 되지만, 데이터를 쓰거나 수정할 수 있는 트랜잭션은 비즈니스 계층에서만 사용할 수 있도록 트랜잭션이 일어난다.



* spring.jpa.open-in-view : true 기본값

Spring Boot JPA 의존성을 주입 받아 어플리케이션을 구성할 경우 spring.jpa.open-in-view의 기본값인 true로 지정되어 있어 OSIV가 적용된 상태로 어플리케이션이 구성된다.

동작 원리는 다음과 같다.

* 클라이언트의 요청이 들어오면 서블릿 필터나, 스프링 인터셉터에서 영속성 컨텍스트를 생성한다. 단 이 시점에서 트랜잭션은 시작하지 않는다.
* 서비스 계층에서 @Transeactional로 트랜잭션을 시작할 때 위에서 미리 생성해둔 영속성 컨텍스트를 찾아와서 트랜잭션을 시작한다.
* 서비스 계층이 끝나면 트랜잭션을 커밋하고 영속성 컨텍스트를 플러시한다. 이 시점에 트랜잭션은 끝내지만 영속성 컨텍스트는 종료되지 않는다.
* 컨트롤러와 뷰까지 영속성 컨텍스트가 유지되므로 조회한 엔티티는 영속 상태를 유지한다.
* 서블릿 필터나, 스프링 인터셉터로 요청이 돌아오면 영속성 컨텍스트를 종료한다. 이때 플러시를 호출하지 않고 바로 종료한다.

서비스 계층에서 트랜잭션이 끝나면 컨트롤러와 뷰에는 트랜잭션이 유지되지 않는 상태이다. 엔티티를 변경하지 않고 단순히 조회만 할 때는 트랜잭션이 없어도 동작하는데, 이것을 트랜잭션 없이 읽기(Nontransactional reads)라 한다. 하여 만약 프록시를 뷰 렌더링하는 과정에 초기화(Lazy loading)가 일어나게 되어도 조회 기능이므로 트랜잭션이 없이 읽기가 가능하다.

영속성 컨텍스트는 기본적으로 트랜잭션 범위 안에서 엔티티를 조회하고 수정할 수 있다.

영속성 컨텍스트는 트랜잭션 범위 밖에서 엔티티를 조회만 할 수 있다. 이것을 트랜잭션 없이 읽기(Nontransactional reads)라 한다.

만약 트랜잭션 범위 밖인 컨트롤러와 뷰에서 엔티티를 수정하여도 영속성 컨텍스트의 변경 감지에 의한 데이터 수정이 다음 2가지 이유로 동작하지 않는다.

* 영속성 컨텍스트의 변경 내용을 데이터베이스에 반영하려면 영속성 컨텍스트를 플러시(flush)해야 한다. 스프링이 제공하는 OSIV는 요청이 끝나면 플러시를 호출하지 않고 em.close()로 영속성 컨텍스트만 종료시켜 버린다.
* 프레젠테이션 계층에서 em.flush()를 호출하여 강제로 플러시해도 트랜잭션 범위 밖이므로 데이터를 수정할 수 없다는 예외가 일어난다.

→ javax.persistence.TransactionRequiredException

**OSIV 사용시 주의점**



spring.jpa.open-in-view의 값을 기본값(true)으로 어플리케이션을 구동하면, 어플리케이션 시작 시점에 위와 같은 warn 로그를 남기게 된다.

그런데 위 동작 방식처럼 프록시를 초기화하는 작업을 Service 계층에서 끝내지 않고도 렌더링 시 자동으로 해결하게 해주는 장점이 있는 OSIV전략에 왜 경고를 줄까?

OSIV 전략은 트랜잭션 시작처럼 최초 데이터베이스 커넥션 시작 시점부터 API 응답이 끝날 때 까지 영속성 컨텍스트와 데이터베이스 커넥션을 유지한다. 그래서 View Template이나 API 컨트롤러에서 지연 로딩이 가능하다.

지연 로딩은 영속성 컨텍스트가 살아있어야 가능하고, 영속성 컨텍스트는 기본적으로 데이터베이스 커넥션을 유지한다. 이것 자체가 큰 장점이다.

그런데 이 전략은 너무 오랜시간동안 데이터베이스 커넥션 리소스를 사용하기 때문에, 실시간 트래픽이 중요한 애플리케이션에서는 커넥션이 모자랄 수 있다. 이것은 결국 장애로 이어진다.

예를 들어서 컨트롤러에서 외부 API를 호출하면 외부 API 대기 시간 만큼 커넥션 리소스를 반환하지 못하고, 유지해야 한다.

→ OISV의 치명적인 단점, 커넥션을 영속성 컨텍스트가 종료될 때까지 1:1로 계속 물고 있음

텍스트, 도표, 스크린샷, 라인이(가) 표시된 사진

자동 생성된 설명

spring.jpa.open-in-view: false (OSIV 종료)

OSIV를 끄면 트랜잭션을 종료할 때 영속성 컨텍스트를 닫고, 데이터베이스 커넥션도 반환한다. 따라서 커넥션 리소스를 낭비하지 않는다.

OSIV를 끄면 모든 지연로딩을 트랜잭션 안에서 처리해야 한다. 따라서 지금까지 작성한 많은 지연 로딩 코드를 트랜잭션 안으로 넣어야 하는 단점이 있다. 그리고 view template에서 지연로딩이 동작하지 않는다.

결론적으로 트랜잭션이 끝나기 전에 지연 로딩을 강제로 호출해 두어야 한다.

커멘드와 쿼리 분리

실무에서 OSIV를 끈 상태로 복잡성을 관리하는 좋은 방법이 있다. 바로 Command와 Query를 분리하는것이다.

보통 비즈니스 로직은 특정 엔티티 몇 개를 등록하거나 수정하는 것이므로 성능이 크게 문제가 되지 않는다. 그런데 복잡한 화면을 출력하기 위한 쿼리는 화면에 맞추어 성능을 최적화 하는 것이 중요하다. 하지만 그 복잡성에 비해 핵심 비즈니스에 큰 영향을 주는 것은 아니다.

그래서 크고 복잡한 애플리케이션을 개발한다면, 이 둘의 관심사를 명확하게 분리하는 선택은 유지보수 관점에서 충분히 의미 있다.

단순하게 설명해서 다음처럼 분리하는 것이다.

OrderService

OrderService: 핵심 비즈니스 로직

OrderQueryService: 화면이나 API에 맞춘 서비스 (주로 읽기 전용 트랜잭션 사용)

보통 서비스 계층에서 트랜잭션을 유지한다. 두 서비스 모두 트랜잭션을 유지하면서 지연 로딩을 사용할 수있다.

매우 중요

OSIV 설정을 yml 파일로 글로벌하게 적용시키고 싶지 않을 경우(프로젝트가 단일 모듈로 되어 있어서 함께 배포가 되어야 한다면) OpenEntityManagerInViewFilter 나 OpenEntityManagerInViewInterceptor를 직접 스프링에 등록하면 되고, 등록하면서 적용되는 URL Path를 적용하고 싶은 곳으로만 잡아주면 된다.

**OSIV 정리**

특징

OSIV는 클라이언트 요청이 들어올 때 영속성 컨텍스트를 생성해서 요청이 끝날 때까지 같은 영속성 컨텍스를 유지한다. 하여 한 번 조회된 엔티티는 요청이 끝날 때까지 영속 상태를 유지한다.

엔티티 수정은 트랜잭션이 있는 계층에서만 동작한다. 트랜잭션이 없는 프레젠테이션 계층은 지연 로딩을 포함해 조회만 할 수 있다.

단점

영속성 컨텍스트와 DB 커넥션은 1:1로 물고있는 관계이기 때문에 프레젠테이션 로직까지 DB 커넥션 자원을 낭비하게 됨.

OSIV를 적용하면 같은 영속성 컨텍스트를 여러 트랜잭션이 공유하게될 수도 있다.

프레젠테이션에서 엔티티를 수정하고 비즈니스 로직을 수행하면 엔티티가 수정될 수 있다.

프레젠테이션 계층에서 렌더링 과정에서 지연 로딩에 의해 SQL이 실행된다. 따라서 성능 튜닝시에 확인해야 할 부분이 넓어진다.