JPA 04 자바 표준 ORM 표준 JPA 프로그래밍

김영한
2022.08.04

프록시

```
Member member = em.find(Member.class, 1L);
print(member);
printMemberAndTeam(member);
```

```
private static void print(Member member) {
   System.out.println("member = " + member.getUsername());
}

private static void printMemberAndTeam(Member member) {
   String username = member.getUsername();
   System.out.println("username = " + username);
   Team team = member.getTeam();
   System.out.println("team.getName() = " + team.getName());
}
```

이런문제를 해결하기 위해 지연로딩이 필요

em.getReference(Member.class, member.getId()); 를 사용했을 때 가짜 객체를 채워넣어서 필요한 쿼리를 날린다.

```
Member reference = em.getReference(Member.class, member.getId());
System.out.println("reference = " + reference.getClass());
```

reference = class hellojpa.Member$HibernateProxy$jK2Kq8Ky

하이버네이트가 만든 가짜 엔티티가 생성된다. 가짜 (프록시)

프록시 특징

- 실제 객체의 참조(target)를 보관
- 프록시 객체를 호출하면 실제 Entity를 참조해서 가져온다.(DB 조회)

초기화 - > 실제 Entity 생성을 영속성 컨텍스트에 요청하는 작업. 이 이뤄지고 프록시 객체가 조회된걸 사용한다.

- 프록시 객체는 처음 사용할 때 한 번만 초기화
- 프록시 객체를 초기화 할 때. 프록시 객체가 실제 엔티티로 바뀌는게 아니다! 초기화 되면 프록시 객체를 통해서 실제 엔티티에 접근 가능하다.
- 프록시 객체는 원복 객체랑 동일하지 않기 때문에 == 비교로 같지 않다 대신 instanceOf로 비교 가능하다.
- 영속성 컨텍스트에 찾는 엔티티가 이미 있으면 em.getReference()를 호출해도 실제 엔티티 반환 **[JPA**는 같은 영속성 컨텍스트에서 같은 엔티티에대해 무조건 같은 엔티티라는 것을 보장해 준다.**]** { 나중에 em.find를 해서 호출하면 프록시로 반환한다.}
- 영속성 컨텍스트의 도움을 받을 수 없는 준영속 상태일 때, 프록시를 초기화 하면 문제 발생 [엔티티로 관리안하는데 DB접근하려는 모양새가 된다]

프록시 확인

초기화 여부 확인

```
System.out.println("is Loaded : " +
emf.getPersistenceUnitUtil().isLoaded(reference));
System.out.println("reference.getUsername() = " + reference.getUsername());
System.out.println("is Loaded : " +
emf.getPersistenceUnitUtil().isLoaded(reference));
```

초기화 전 false - > getUserName 이후 -> true

프록시 클래스 확인

```
System.out.println("reference = " + reference.getClass());
```

강제 초기화

```
System.out.println("is Loaded : " +
emf.getPersistenceUnitUtil().isLoaded(reference));
Hibernate.initialize(reference);    → 강제 초기화
System.out.println("is Loaded : " +
emf.getPersistenceUnitUtil().isLoaded(reference));
```

즉시로딩과 지연 로딩

member를 조회할 때 team 도 함께 조회해야하나?

지연 로딩

지연 로딩시

```
Team team = new Team();
team.setName("teamA");
em.persist(team);

Member member = new Member();
member.setUsername("hello");
member.setTeam(team);
em.persist(member);

em.flush();
em.clear();

Member m = em.find(Member.class, member.getId());
System.out.println("m.getTeam().getClass() = " + m.getTeam().getClass());
→ 팀은 프록시! 팀 프록시 객체가 조회 되고
m.getTeam().getName(); → 이 순간에 Member내부 Team 프록시를 초기화 한다.
```

즉시 로딩

멤버와 팀을 자주 사용한다면 ? EAGER (즉시 로딩) ⇒ 프록시가 아닌 진짜 클래스를 가져온다.

실무에서는 즉시 로딩 조심**!**

- **-** 가급적 지연 로딩만 사용**(** 실무 **)**
- 예상치 못한 SQL 발생
- **-** 즉시 로딩은 **JPQL**에서 **N+1** 문제를 일으킨다**.**

  [필요한 테이블 **(1)** → 추가 **N**개 **(**팀이 **N**개 **FK**로 엮여있으면**)]**

- **-** **@ManyToOne, @OneToOne** 은 기본이 즉시로딩 → **LAZY**로 변경
- @OneToMany, @ManyToMany는 기본이 지연로딩

실무에선**?**

- 우선 전체를 LAYZ

1. fetch join (패치 조인)
2. 엔티티 그래프 기능 3. . . .

영속성 전이 **: CASCADE**

특정 엔티티를 영속 상태로 만들 때 연관된 엔티티도 함께 영속 상태로 만들고 싶을 때

예: 부모 엔티티를 저장할 때 자식 엔티티도 함께 저장.

```
@OneToMany(mappedBy = "parent", cascade = CascadeType.ALL)
private List<Child> childList = new ArrayList<>();
```

parent 를 em.persist(parent) 로 해도 set한 child 들도 영속화 된다.

주의점

영속성 전이는 연관관계를 매핑하는 것과는 아무관계 없다.

ALL, PERSIST 옵션 정도만 사용하는게 좋다.

하나의 부모가 자식들을 관리하면 의미가 있다.

예시) 게시판과 첨부파일에서는 사용할 수 있지만 첨부 파일이 여러곳에서 관리되는 관계라면 사용하면 안된다. 즉, 소유자가 하나 일때 사용해야한다.단일 엔티티에 종속적일 때 사용해도 무방하다.

정리 : 라이프 사이클이 거의 유사할 때, 단일 엔티티에 종속적일때 (단일 소유일 때)

고아 객체

부모 엔티티와 연관관계가 끊어진 자식 엔티티를 자동으로 삭제하는 기능

orphanRemoval = true

```
@OneToMany(mappedBy = "parent", cascade = CascadeType.ALL, orphanRemoval = true)
private List<Child> childList = new ArrayList<>();
```

```
Child child1 = new Child();
Child child2 = new Child();

Parent parent = new Parent();
parent.addChild(child1);
parent.addChild(child2);

em.persist(parent);

em.flush();
em.clear();

Parent findParent = em.find(Parent.class, parent.getId());
findParent.getChildList().remove(0);
```

→ 첫번째 자식이 자식 테이블에서도 delete 쿼리가 날아간다.

주의점

- 부모를 제거하면 자식 엔티티도 전부 제거하기 때문에 Cascade.REMOVE처럼 동작하게 된다.
- 조심히 사용해야한다. 특정엔티티가 개인소유일 때만 사용해야 한다.

영속성 전이 **+** 고아 객체

- 부모 엔티티를 통해서 생명주기를 관리하게 된다.
- DDD 도메인 주도 설계 Aggregate Root 개념을 구현할 때 유용