DAOUTECH

영속성 컨텍스트

플랫폼서비스팀

김설한

2021.09.01

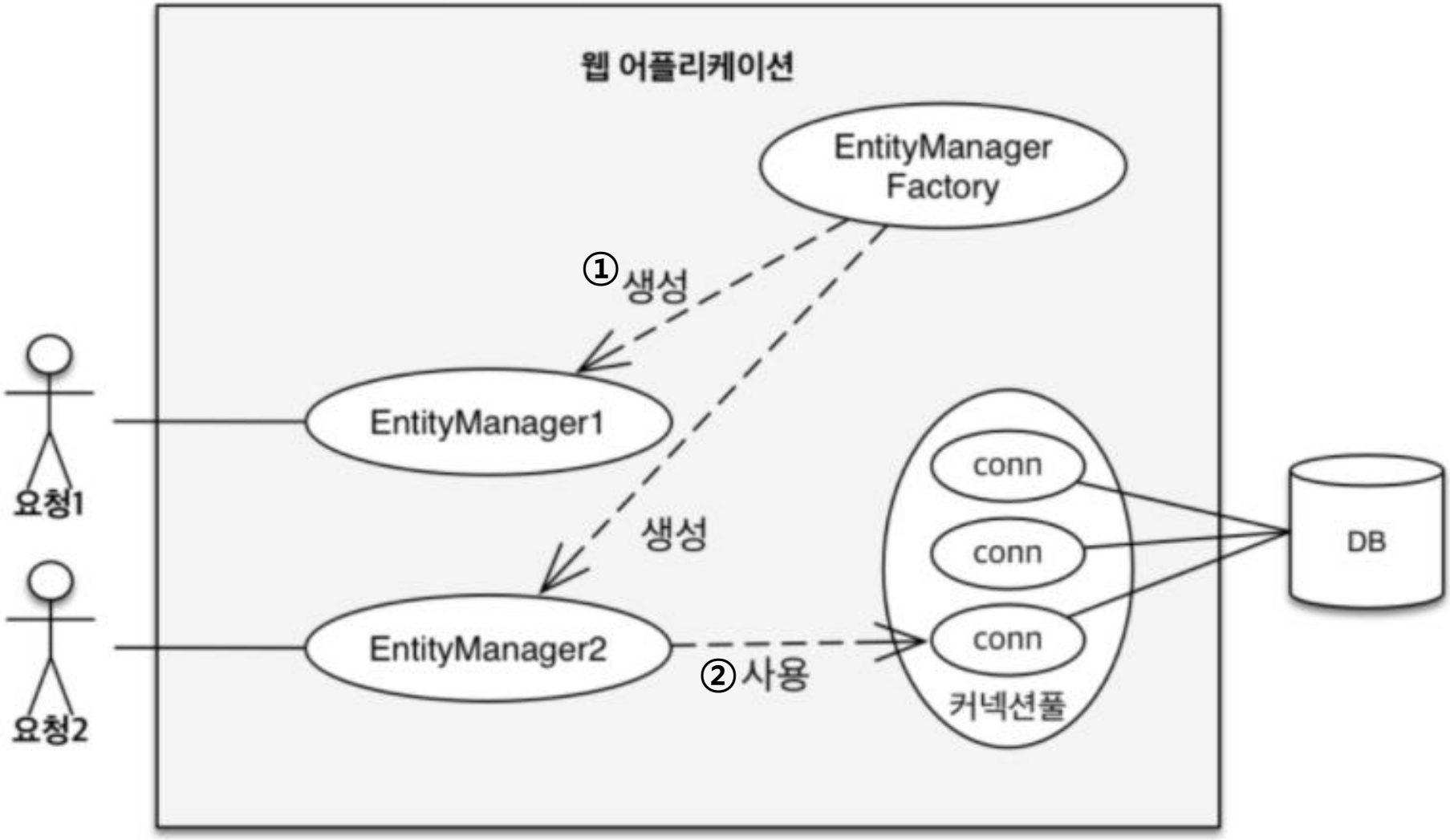
• 목차

- 엔티티 매니저 & 엔티티 매니저 팩토리
- 영속성 컨텍스트
- 엔티티 생명 주기
- 영속성 컨텍스트의 특징
- 스프링 데이터 JPA의 영속성 컨텍스트

01

엔티티 매니저 &
엔티티 매니저 팩토리

엔티티 매니저 & 엔티티 매니저 팩토리



엔티티 매니저 & 엔티티 매니저 팩토리

- **EntityManager**
  - 엔티티의 CRUD 등 관련된 모든 일을 처리
  - 여러 스레드가 동시에 접근하면 동시성 문제가 발생.
  - 스레드 간 공유 절대 불가
  - DB 연결이 필요한 시점까지 커넥션을 얻지 않는다.

- **EntityManagerFactory**
  - EntityManager를 만드는 공장
  - 일반적으로 하나의 DB를 사용하는 application은 하나만 생성

02

영속성 컨텍스트

영속성 컨텍스트

- 사전적 의미 : Entity를 영구 저장하는 환경
- 애플리케이션이 DB에서 꺼내온 객체를 보관하는 역할 (Collection과 유사)
- EntityManager를 통해 Entity를 조회하거나 저장할 때, 보관하고 관리하는 역할
- EntityManager를 생성할 때 하나씩 만들어진다.

03

엔티티 생명 주기

엔티티 생명 주기

- 엔티티의 생명 주기는 4가지로 나눌 수 있다.

  [1] 비영속 (new/transient)

  : 영속성 컨텍스트와 전혀 관계 없는 새로운 상태. Entity 객체를 new만 해 둔 상태.

  [2] 영속 (managed)

  : 영속성 컨텍스트에 관리되는 상태.

  [3] 준영속 (detached)

  : 영속성 컨텍스트에 저장되었다가 분리된 상태.

  [4] 삭제 (removed)

  : DB에서 엔티티가 삭제된 상태,

엔티티 생명 주기

※ **영속 (managed)**

- 엔티티 매니저를 통해 엔티티를 영속성 컨텍스트에 저장한 상태
- 영속성 컨텍스트가 엔티티를 관리하므로 영속 상태가 된다.
- 스프링 컨테이너는 트랜잭션 범위와 영속성 컨텍스트의 생존 범위가 같기에,

  @Transactional 내부는 영속 상태가 된다.

04

영속성 컨텍스트 특징

영속성 컨텍스트 특징

- 식별자 값
  - @Id로 테이블의 기본 키와 매핑한 값
  - 영속성 컨텍스트는 엔티티를 식별자 값으로 구분하기에 반드시 있어야 한다.

- 플러시 (Flush)
  - 영속성 컨텍스트의 변경 내용을 데이터에 반영하는 것
  - 영속성 컨텍스트의 엔티티를 지우는게 아닌, 변경 내용을 데이터베이스에 동기화 하는 것
  - 영속성 컨텍스트를 플러시 하는 방법

    [1] em.flush() : 직접 호출

    [2] 트랜잭션 커밋 : 플러시 자동 호출

    [3] JPQL 쿼리 실행 : 플러시 자동 호출

영속성 컨텍스트 특징

- 영속성 컨텍스트가 엔티티를 관리하면, 아래와 같은 장점들이 있다.

  [1] 1차 캐시

  [2] 동일성 보장

  [3] 트랜잭션을 지원하는 쓰기 지원

  [4] 변경 감지

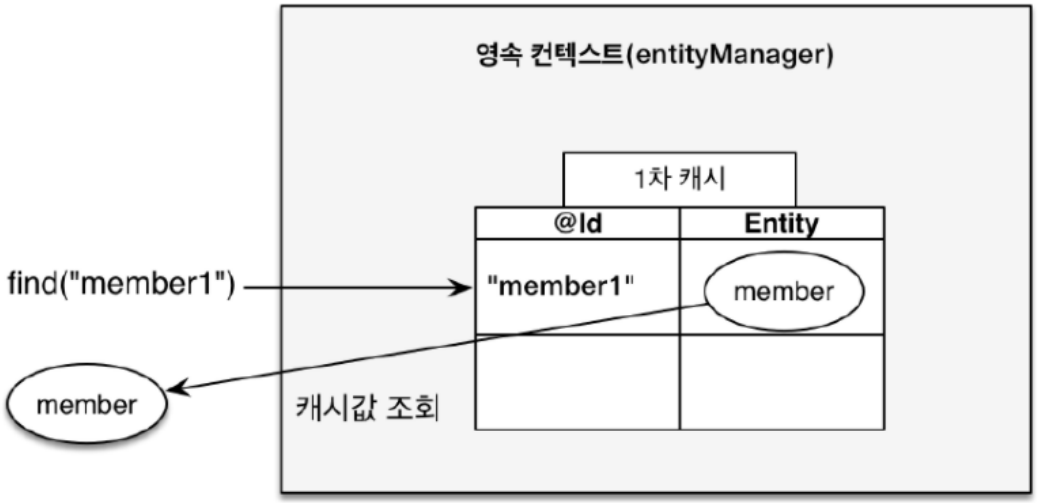
  [5] 지연 로딩

영속성 컨텍스트 특징

1) 1차 캐시

- 영속성 컨텍스트 내부에는 캐시(1차 캐시)가 있다.
- 영속 상태의 엔티티를 이 곳에 저장한다.
- 1차 캐시의 키는 식별자 값이고 값은 엔티티 인스턴스이다.
- 엔티티를 조회할 때 1차 캐시부터 조회하기에 DB 엑세스 횟수를 줄일 수 있게 된다.

```
Member member = new Member();
member.setId("member1");
member.setUsername("회원1");

//1차 캐시에 저장됨
em.persist(member);

//1차 캐시에서 조회
Member findMember = em.find(Member.class, "member1");
```



영속성 컨텍스트 특징

2) 영속 엔티티의 동일성 보장

```
Member a = em.find(Member.class, "member1");
Member b = em.find(Member.class, "member1");

System.out.println(a == b); //동일성 비교 true
```

※ 동일성 비교 : 실제 인스턴스가 같다. ==를 사용해 비교한다.

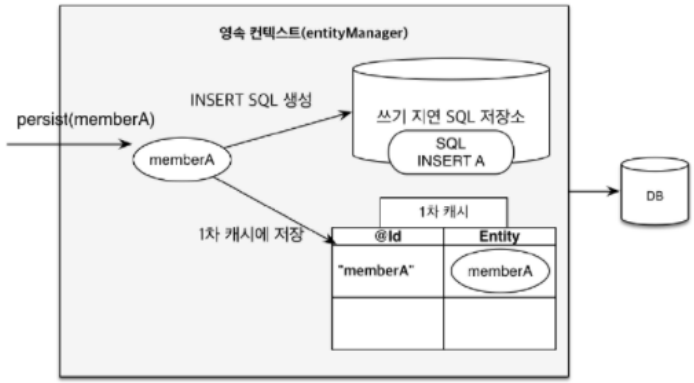
- 이는 1차 캐시로 인해 가능한 특징이다.
- 자바 컬렉션에서 조회하듯 JPA가 영속 엔티티의 동일성을 보장해준다.
- 1차 캐시에 동일한 키 값으로 들어있는 엔티티는 재 저장하지않기 때문에 동일성이 보장된다.

영속성 컨텍스트 특징

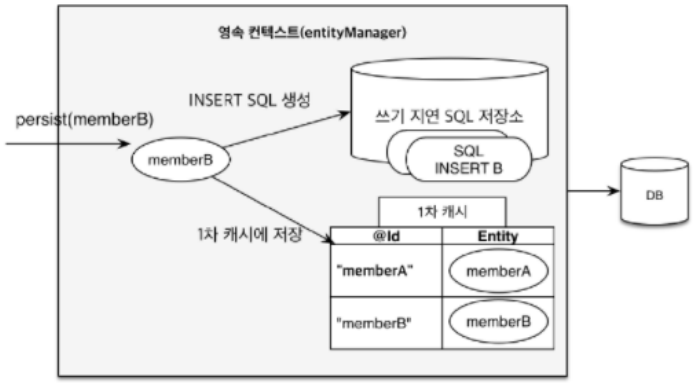
3) 트랜잭션을 지원하는 쓰기 지연

```
EntityManager em = emf.createEntityManager();
EntityTransaction transaction = em.getTransaction();
//엔티티 매니저는 데이터 변경시 트랜잭션을 시작해야함
transaction.begin(); // 트랜잭션 시작

em.persist(memberA);
em.persist(memberB);
//여기까지 INSERT SQL을 DB에 보내지 않음
//JPA가 쿼리 쭉쭉 쌓고 있는다

//커밋하는 순간 DB에 INSERT SQL을 날린다
transaction.commit(); //트랜잭션 커밋
```

em.persist(memberA);



em.persist(memberB);

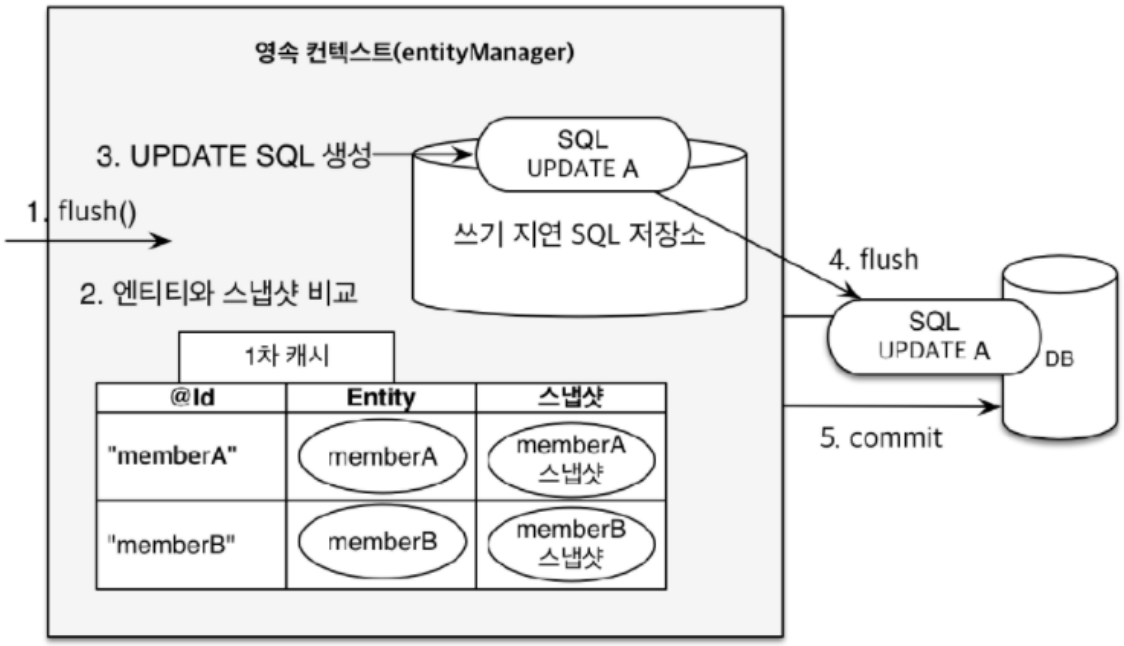


- 엔티티가 영속화 될 때 1차 캐시에 엔티티를 쌓는데, 그 때 DB에 반영할 SQL을 생성해 영속성 컨텍스트 내에 쓰기 지연 SQL 저장소에 쌓아둔다.
- 쓰기 지연 SQL 저장소의 SQL들은 트랜잭션 커밋 시점에 한꺼번에 flush된다.

영속성 컨텍스트 특징

4) 변경 감지

```
EntityManager em = emf.createEntityManager();
EntityTransaction transaction = em.getTransaction();
transaction.begin() //트랜잭션 시작

//영속 엔티티 조회
Member memberA = em.find(Member.class, "memberA");

//영속 엔티티 데이터 수정
memberA.setUsername("hi");
memberA.setAge(10);

//em.update(member) 이런 코드가 있어야하지 않나?

transaction.commit(); //트랜잭션 커밋
```



- 단순히 조회된 데이터를 변경하면, 변경 내용이 DB에도 반영 된다.
- 영속성 컨텍스트 플러시 흐름을 살펴보면 그 이유를 알 수 있다.

영속성 컨텍스트 특징

- 플러시 흐름

  [1] 트랜잭션의 커밋이 발생하면, 먼저 영속성 컨텍스트의 flush()가 호출된다.

  [2] 이 때 영속성 컨텍스트는 1차 캐시와 스냅샷을 비교하며 변경된 엔티티를 찾는다.

  ※ 스냅샷 : 최초로 1차 캐시에 들어온 상태를 저장해 둔 것.

  [3] 변경된 엔티티가 있다면, 수정 쿼리를 생성해 쓰기 지연 SQL 저장소에 저장한다.

  [4] 쓰기 지연 저장소의 (생성, 수정, 삭제)SQL들을 DB에 반영한다.

  [5] 트랜잭션을 커밋한다.

영속성 컨텍스트 특징

5) 지연 로딩 (Lazy Loading)

- JPA는 프록시를 사용한 지연 로딩 기능을 제공 해, 불필요한 연관관계 조회를 막아준다.

영속성 컨텍스트 특징

- 프록시
  - 실제 클래스를 상속받으며 겉 모양(method)이 같아, 사용 시 구분이 안된다.
  - 프록시 객체는 실제 객체의 참조(target)을 보관하는데,

    참조에 실제 접근하기 전까지는 target은 지정되지 않는다.
  - Target 객체를 실제 접근(호출)하면 DB에서 target객체를 조회하고, 프록시 객체의 target 필드를

    조회한 target객체의 주소(참조)값으로 초기화한다.
  - 준 영속성 상태(대표적으로 transaction종료)에 초기화 되지 않은 target에 접근하려 하면

    hibernate는 org.hibernate.LazyInitializationException 예외를 터뜨린다.

    (트랜잭션이 끝나고 영속성 컨텍스트를 조회할 때 많이 발생하는 오류)

참고

Spring Data JPA
영속성 컨텍스트

스프링 데이터 JPA 영속성 컨텍스트

- 스프링 데이터 JPA의 JpaRepository 구현체인 SimpleJpaRepository 를 살펴보면 class 기준 @Transactional 애노테이션이 붙어있다.

```
@Repository
@Transactional(readOnly = true)
public class SimpleJpaRepository<T, ID> implements JpaRepositoryImplementation<T, ID> {
```

※ 변경이 포함된 메서드(save/ delete/ deleteById 등...)들은 따로 @Transactional 처리가 되어 있다.

```
@Transactional
@Override
public <S extends T> S save(S entity) {

    Assert.notNull(entity, message: "Entity must not be null.")

    if entityInformation.isNew(entity) {
        em.persist(entity)
        return entity
    } else {
        return em.merge(entity)
    }
}
```

```
@Override
@Transactional
/unchecked/
public void delete(T entity) {

    Assert.notNull(entity, message: "Entity must not be null!")

    if entityInformation.isNew(entity) {
        return
    }

    val type = ProxyUtils.getUserClass(entity)

    val existing = em.find(type, entityInformation.getId(entity))

    // if the entity to be deleted doesn't exist, delete is a NOOP
    if existing == null {
        return
    }

    em.remove(em.contains(entity) ? entity : em.merge(entity))
}
```

스프링 데이터 JPA 영속성 컨텍스트

- 즉, 해당 클래스 內 메서드들은 각각 트랜잭션 범위이며 영속성 컨텍스트 범위이다.
- 각 메서드들을 동일한 영속성 컨텍스트에서 사용하고자 한다면,

  메서드들을 호출하는 서비스 단 메서드를 @Transactional로 묶어주면 된다.

```java
@Override
@Transactional
public void saveRecommendation(Long userId, LinkProduct product) {

    LinkRecommendation linkRecommendation = linkRecommendationRepository.findOneByUserIdAndProduct(userId, product);

    if(linkRecommendation == null) {
        linkRecommendation = new LinkRecommendation(userId, product,  recommended: true);
    } else {
        linkRecommendation.updateRecommended(!linkRecommendation.recommended);
    }

    linkRecommendationRepository.save(linkRecommendation);
}
```

- 그렇게 되면 하위 Repository 메서드들은 서비스 단 트랜잭션을 위임 받아,

  동일 트랜잭션 내에 수행되므로 동일한 영속성 컨텍스트를 사용하게 된다.

감사합니다!