Chap15. 고급 주제와 성능 최적화

-꼬막조림

알아볼 내용

- 예외 처리
- 엔티티 비교
- 프록시 심화 주제
- 성능 최적화

예외 처리

JPA 표준 예외

트랜잭션 롤백을 표시하는 예외

심각한 예외.

복구해선 안 됨.

예외 발생 후 강제로 커밋하려고 해도 커밋이 되지 않음.

트랜잭션 롤백을 표시하지 않는 예외

심각하지 않은 예외.

커밋 할지 롤백할지 결정 가능.

예외 처리

트랜잭션 롤백을 표시하는 예외

EntityExistsException
- EntityManager.persist(..) 호출 시 이미 같은 엔티티가 있으면 발생.

EntityNotFoundException
- EntityManager.getReference(..)를 호출했는데 실제 사용 시 엔티티가 존재하지 않을 때 발생.

OptimisticLockException
PessimisticLockException 등

트랜잭션 롤백을 표시하지 않는 예외

NoResultException
- 결과가 없을 때.

NonUniqueResultException
- 결과가 하나를 초과 할 때.

LockTimeoutException
QueryTimeoutException 등

예외 처리

서비스 계층에서 데이터 접근 계층의 구현 기술에 직접 의존하는 것은 좋은 설계가 아님.

예외도 마찬가지인데, 서비스 계층에서 JPA 예외를 직접 다루는건 좋지 못한 설계임.

스프링 프레임워크는 데이터 접근 계층에 대한 예외를 추상화해서 제공함.

예외 처리

JPA 예외	->	스프링 변환 예외
PersistenceException	->	JpaSystemException
NoResultException	->	EmptyResultDataAccessException
NonUniqueResultException	->	IncorrectResultSizeDataAccessException
LockTimeoutException	->	CannotAcquireLockException
	...	

예외 처리

appConfig.xml

```
<beans>
        <bean class=”org.springframework
                .dao.annotation.PersistenceExc
        eptionTranslationPostProcessor” />
</beans>
```

@Repository
NoResultExceptionTestRepository.java

```
public Member findMember() {
        return em.createQuery(“select m
        from …, …).getSingleResult();
}
```

인터셉터 동작

NoResultException 발생

결과값이 1개
초과일때 에러 발생

EmptyResultDataAccessException 으로 변환

엔티티 비교(영속성 컨텍스트가 같을 때)

MemberServiceTest.java

```
@Transactional
public class MemberServiceTest {
        @Test
        public void 테스트() {
                Member mem1 = new Member("홍길동");
                Long savedId = service.join(mem1);
                Member mem2 = repo.findOne(savedId);
                assertTrue( mem1 == mem2 );
        }
}
```

영속성 컨텍스트 내부에 있는
1차 캐시는 찾으려는 엔티티가
존재하면 항상 같은 엔티티를 반환함.
(엔티티의 값뿐만 아니라 주소값이
같음)

==연산자, equals() 비교 가능.

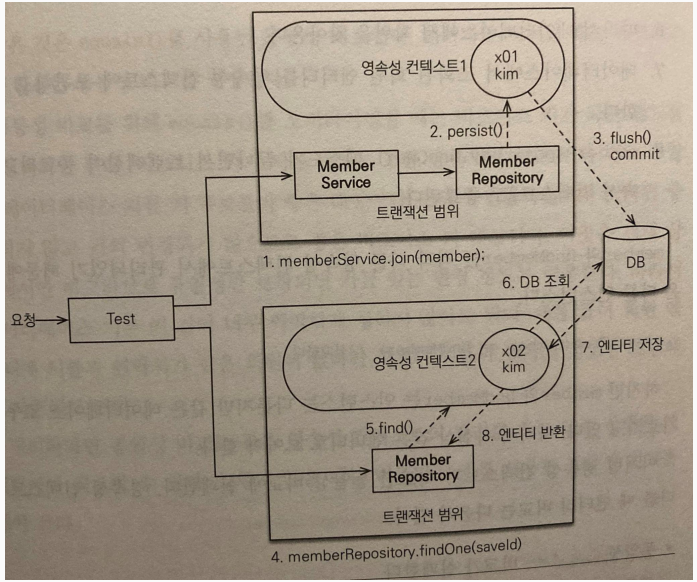
엔티티 비교(영속성 컨텍스트가 다를 때)

MemberServiceTest.java

```
public class MemberServiceTest {
        @Test
        public void 테스트() {
                Member mem1 = new Member("홍길동");
                Long savedId = service.join(mem1);
                Member mem2 = repo.findOne(savedId);
                assertTrue( mem1 == mem2 ); // 실패
        }
}
```

== 연산자는 사용불가.

eqauls()를 구현 후 사용.(기본키와 같은 유니크한 값으로 비교)



프록시 심화주제

객체 비교(==)

```
// 1. 프록시 객체 조회
refMem = em.getReference(Member.class, “mem1”);
// 2. 원본 객체 조회
findMem = em.find(Member.class, “mem1”);
Assert.assertTrue(refMem == findMem);
```

refMem, findMem 모두 프록시 객체임
class = Member_$$_jvst843_0

```
// 1. 원본 객체 조회
findMem = em.find(Member.class, “mem1”);
// 2. 프록시 객체 조회
refMem = em.getReference(Member.class, “mem1”);
Assert.assertTrue(refMem == findMem);
```

refMem, findMem 모두 원본 객체임
class = Member

프록시 심화주제

객체 클래스 비교(==)

```
// 프록시 객체 조회
refMem = em.getReference(Member.class, “mem1”);

Assert.assertFalse(Member.class ==refMem.getClass());

Assert.assertTrue(refMem instanceof Member);
```

프록시 객체와 원본 객체의 엔티티 타입비교 하려면 instanceof 연산자 사용

프록시 심화주제

객체&프록시 간 동등비교 비교(equals())

```
// 1. 프록시 객체 조회
refMem = em.getReference(Member.class, “mem1”);
em.flush();  em.clear(); // 영속성 컨텍스트 초기화
// 2. 원본 객체 조회
findMem = em.find(Member.class, “mem1”);

Assert.assertFalse( findMem.eqauls(refMem) );
```

equals()로 동등비교 실패.
equals 메소드의
refMem.name == findMem.name은
“mem1” == “mem1” 으로 true 일것 같지만
refMem은 프록시 객체여서
refMem.name의 값은 null임.
refMem.getName()==findMem.getName()
으로 수정 해주면 됨.

성능 최적화

읽기 전용 쿼리의 성능 최적화

영속성 컨텍스트를 사용하여 엔티티를 쓰게되면 1차 캐시를 사용하여 많은 메모리가 필요한데 단순 조회용 데이터는 영속성 컨텍스트를 이용하지 않는것이 좋음.

1. 스칼라 타입으로 조회
   스칼라타입으로 조회하게 되면 영속성 컨텍스트가 결과를 관리하지 않음.
   select o.id, o.name, o.price from Order o

2. 읽기 전용 쿼리 힌트
   힌트를 사용하면 이후에 엔티티를 수정해도 DB에 반영되지 않음.
   TypedQuery<Order> query = em.createQuery("select o from Order o", Order.class);
   query.setHint("org.hibernamte.readOnly", true);

성능 최적화

읽기 전용 쿼리의 성능 최적화

3. 읽기 전용 트랜잭션 사용
   @Transactional(readonly = true)

4. 트랜잭션 밖에서 읽기
   트랜잭션 없이 엔티티를 조회.
   @Transactional(propagation = Propagation.NOT_SUPPORTED)

성능 최적화

SQL 쿼리 힌트 사용

JPA는 데이터베이스 SQL 힌트 기능을 제공하지 않음.
하이버네이트를 직접 사용해서 SQL 쿼리 힌트를 사용.

```
Session session = em.unwrap(Session.class); // 하이버네이트 직접 사용

List<Member> list = session.createQuery("select m from Member m")
                           .addQueryHint("FULL (MEMBER)")
                           .list();
```

성능 최적화

트랜잭션을 지원하는 쓰기 지연과 성능 최적화

네트워크 호출 한 번은 단순한 메소드를 수만 번 호출하는 것보다 더 큰 비용이 듬.
JDBC가 제공하는 SQL 배치 기능을 사용하면 SQL을 모아 데이터베이스에 한번에 보낼 수 있음.
<property name="hibernate.jdbc.batch_size" value="50"/> <!-- 50건씩 모아서 실행 →

※ em.persist()는 호출즉시 Insert문 실행 됨. 배치를 이용한 쓰기지연 불가.
※ 배치처리 할 쿼리 중간에 다른 처리가 들어가면 배치 처리불가.
ex)em.persist(new Member());
em.persist(new Member()); // 여기서 배치 실행 1
em.persist(new Child()); // 여기서 배치 실행 2
em.persist(new Member()); // 여기서 배치 실행 3

감사합니다.