**2주차 정리**

자바 ORM 표준 JPA 프로그래밍 – 기본편

장유현

**섹션 01. JPA 소개**

**SQL 중심적인 개발의 문제점**

: 데이터 베이스 관계형 DB에 저장해서 관리

- 무한 반복, 지루한 코드 생성

- 패러다임의 불일치 : 관계형 데이터 베이스에 저장할 시 개발자의 역량이 필요

\* 객체와 관계형 데이터 베이스의 차이

상속 관계 : 객체 상속 존재, 테이블(관계형 데이터 베이스)은 상속 대신 슈퍼타입 서브타입 관계

연관 관계 : 객체는 참조 사용, 테이블(관계형 데이터 베이스)은 외래 키 사용

객체 모델링

객체 그래프 탐색 : 객체는 자유롭게 객체 그래프를 탐색

* 객체를 자바 컬렉션에 저장하듯이 DB에 저장하는 방법 모색
* JPA

**JPA 소개**

\*ORM

- Object-relational mapping(객체 관계 매핑)

- 객체는 객체대로 설계

- 관계형 데이터 베이스는 관계형 데이터베이스대로 설계

- ORM 프레임워크가 중간에서 매핑

- 대중적인 언어에는 대부분 ORM 기술이 존재

특징

- 애플리케이션과 JDBC 사이에서 동작

- 동작 ; 저장

1. entity (멤버 객체) 분석

2. INSERT SQL 생성하고 JDBC API 사용해서 DB에 결과값 전송

3. 패러다임 불일치 해결(중요)

4. JDBC API를 통해 결과 반환 받고 적절하게 SELECT SQL 생성

5. ResultSet 매핑

6. 패러다임 불일치 해결(중요)

- JPA는 인터페이스의 모음

- JPA 2.1 표준 명세를 구현한 3가지 구현체 : 하이버네이트, EclipseLink, DataNucleus

JPA 주요 기능

- SQL 중심적인 개발에서 객체 중심으로 개발

- 생산성

저장: jpa.persist(member)

조회: Member member = jpa.find(memberld)

수정: member.setName(“변경할 이름”)

삭제: jpa.remove(member)

- 유지 보수 : 필드만 추가해서 사용하면서 SQL은 JPA가 모두 처리

- 패러다임의 불일치 해결

1. JPA와 상속(저장, 조회)

2. JPA 와 연관관계, 객체 그래프 탐색

3. 신뢰할 수 있는 엔티티, 계층

- 성능

1. 1차 캐시와 동일성 보장

- 같은 트랜잭션 안에서는 같은 엔티티를 반환 – 약간의 조회 성능 향상

- DB Isolation Level 이 Read Commit이어도 애플리케이션에서 Repeatable Read 보장

2. 트랜잭션을 지원하는 쓰기 지연(transactional write-behind) ; 버퍼링

- 트랜잭션을 커밋할 때까지 INSERT SQL 을 모음

- JDBC BATCH SQL 기능을 사용해서 한번에 SQL 전송

3. 지연 로딩(lazy loading) : 객체가 실제 사용될 때 전송

(<-> 즉시 로딩 : JOIN SQL로 한번에 연관된 객체까지 미리 조회)

- 데이터 접근 추상화와 벤더 독립성

**섹션02. JPA 시작** //실습이라 내용정리 생략

**섹션 03. 영속성 관리 – 내부 동작 방식**

**영속성 컨텍스트**

- 엔티티를 영구 저장하는 환경

- EntityManager.persist(entity)

- 엔티티 매니저를 통해서 영속성 컨텍스트에 접근

- 이점

1. 1차 캐시

2. 동일성(identity) 보장

: 1차 캐시로 반복 가능한 읽기(REPEATABLE READ) 등급의 트랜잭

션 격리 수준을 데이터베이스가 아닌 애플리케이션 차원에서 제공

3. 트랜잭션을 지원하는 쓰기 지연(transactional write-behind)

4. 변경 감지(dirty checking)

5. 지연 로딩(lazy loading)

\*엔티티의 생명주기

- 비영속성(new/transient) : 영속성 컨텍스트와 전혀 관계없는 새로운 상태

-영속(managed) : 영속성 컨텍스트에 관리되는 상태

- 준영속(detached) : 영속성 컨텍스트에 저장되었다가 분리된 상태

- 삭제(removed) : 삭제된 상태

**플러시**

- 영속성 컨텍스트의 변경 내용을 데이터 베이스에 반영

\* 플러시 발생

- 변경 감지

- 수정된 엔티티 쓰기 지연 SQL 저장소에 등록

- 쓰기 지연 SQL 저장소의 쿼리를 데이터 베이스에 전송(등록, 수정, 삭제 쿼리)

\* 방법

- em.flush() 사용: 직접 호출

- 트랜잭션 커밋시 플러시 자동 호출

- JPQL 쿼리 실행시 플러시 자동 호출

\*플러시 모드 옵션

em.setFlushMode(FlushModeType.COMMIT)

-FlushModeType.AUTO : 커밋이나 쿼리를 실행할 때 플러시(기본값)

-FlushModeType.COMMIT : 커밋할 때만 플러시

- 영속성 컨텍스트를 비우지 않음

- 영속성 컨텍스트의 변경내용을 데이터베이스에 동기화

- 트랜잭션이라는 작업 단위가 중요 -> 커밋 직전에만 동기화

**준영속 상태**

-영속 -> 준영속

- 영속 상태의 엔티티가 영속성 컨텍스트에서 분리(detached)

- 영속성 컨텍스트가 제공하는 기능을 사용 못함

\* 준영속 상태로 만드는 방법

- em.detach(entity) : 특정 엔티티만 준영속 상태로 전환

- em.clear() : 영속성 컨텍스트를 완전히 초기화

- em.close() : 영속성 컨텍스트를 종료