▼ JPA 소개

SQL 중심적인 개발의 문제점

객체를 관계형 db 에 관리해야 하는 생성

- 무한 반복, 지루한 코드 sql 문
- 객체 vs 관계형 데이터베이스 : 객체를 관계형 데이터베이스에 저장하려면 sql 변환
  - 차이 : 상속, 연관관계, 데이터타입, 데이터 식별방법
- 계층형 아키텍처 진정한 의미의 계층 분할이 어렵다

JPA Java Persistence API

자바 진영의 ORM 기술 표준

JPA 는 애플리케시연과 JDBC 사이에서 동작

JPA 는 표준명세

jpa 는 인터페이스의 모음

jpa2.1 표준 명세를 구현한 3가지 구현체

하이버네이트, EclipseLink, DataBucleus

왜 JPA 를 사용?

- sql 중심적인 개발에서 객체 중심으로 개발
- 생산성
  - 저장: jpa.persist(member)
  - 조회: Member member = jpa.find(memberId)
  - 수정: member.setName("변경 이름")
  - 삭제: jpa.remove(member)
- 유지보수 : 필드만 추가하면 됨, sql 은 jpa가 처리
- 패러다임의 불일치 해결(상속, 연관관계, 객체 그래프 탐색, 비교)
- 성능
- 데이터 접근 추상화와 벤더 독립성
- 표준

▼ JPA 시작

./h2.sh

http://localhost:8082/login.do?jsessionid=7dad18a78f25668d9048e5529f1d07d7

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>jpa-basic</groupId>
    <artifactId>ex1-hello-jpa</artifactId>
    <version>1.0.0</version>
    <dependencies>
        <!-- JPA 하이버네이트 -->
        <dependency>
            <groupId>org.hibernate</groupId>
```

```
            <artifactId>hibernate-entitymanager</artifactId>
            <version>5.3.10.Final</version>
        </dependency>
        <!-- H2 데이터베이스 -->
        <dependency>
            <groupId>com.h2database</groupId>
            <artifactId>h2</artifactId>
            <version>1.4.199</version>
        </dependency>
        <!-- 자바 11 이면 추가 -->
        <dependency>
            <groupId>javax.xml.bind</groupId>
            <artifactId>jaxb-api</artifactId>
            <version>2.3.0</version>
        </dependency>
    </dependencies>
</project>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.2"
             xmlns="http://xmlns.jcp.org/xml/ns/persistence" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence http://xmlns.jcp.org/xml/ns/persistence/persistence_2_2.xs
    <persistence-unit name="hello">
        <properties>
            <!-- 필수 속성 -->
            <property name="javax.persistence.jdbc.driver" value="org.h2.Driver"/>
            <property name="javax.persistence.jdbc.user" value="sa"/>
            <property name="javax.persistence.jdbc.password" value=""/>
            <property name="javax.persistence.jdbc.url" value="jdbc:h2:tcp://localhost/~/test"/>
            <property name="hibernate.dialect" value="org.hibernate.dialect.H2Dialect"/>
            <!-- 옵션 -->
            <property name="hibernate.show_sql" value="true"/>
            <property name="hibernate.format_sql" value="true"/>
            <property name="hibernate.use_sql_comments" value="true"/>
            <!--<property name="hibernate.hbm2ddl.auto" value="create" />-->
        </properties>
    </persistence-unit>
</persistence>
```

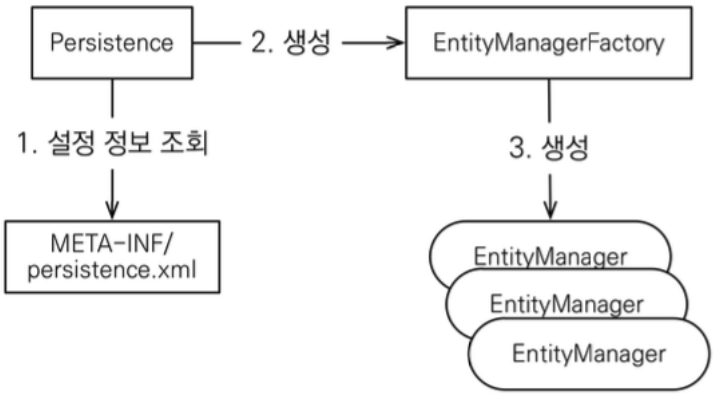
데이터베이스 방언

hibernate.dialect 속성에 지정

H2 : org.hibernate.dialect.H2Dialect
Oracle 10g : org.hibernate.dialect.Oracle10gDialect
MySQL : org.hibernate.dialect.MySQL5InnoDBDialect

하이버네이트는 40가지 이상의 데이터베이스 방언 지원

- jpa 구동방식



create table Member(

id bigint not null,

name varchar(255),

primary key(id)

);

```
package hellojpa;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.EntityTransaction;
import javax.persistence.Persistence;

// 저장의 경우 JPA 정석 코드

public class JpaMain {

    public static void main(String[] args) {
        // 엔티티 매니저 팩토리 생성, 파라미터로 넘어가는 값은 persistence.xml에서 persistence-unit의 name 값을 넘겨주면 된다.
        // emf의 경우 애플리케이션 로딩 시점에 딱 하나만 만들면 된다.
        EntityManagerFactory emf = Persistence.createEntityManagerFactory("hello");
        // 엔티티 매니저 생성
        EntityManager em = emf.createEntityManager();

        EntityTransaction tx = em.getTransaction();
        tx.begin();

        try {

            // 생성
            Member member = new Member();
            member.setId(1L);
            member.setName("helloA");
            em.persist(member); // 저장

            // 조회
            // 첫번째 파라미터 : 엔티티 클래스
            // 두번째 파라미터 : PK 값
            Member findMember = em.find(Member.class, 1L);

            // 삭제
            //em.remove(findMember);

            // 수정 - 저장안해도됨
            findMember.setName("HAN");

            tx.commit();
        } catch (Exception e) {
            tx.rollback();
        } finally {
            // 엔티티 매니저 종료
            em.close();
        }
        // 엔티티 매니저 팩토리 종료
        emf.close();
    }
}
```

```
package hellojpa;

import javax.persistence.Entity;
import javax.persistence.Id;

@Entity
public class Member {
    @Id
    private Long id;
    private String name;

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

주의

- **엔티티 매니저 팩토리**는 하나만 생성해서 애플리케이션 전체에서 공유
- **엔티티 매니저**는 쓰레드간에 공유X (사용하고 버려야 한다).
- **JPA의 모든 데이터 변경은 트랜잭션 안에서 실행**

jpql 로 상세조회됨 - 객체를 대상으로 쿼리 짜기

검색을 할때도 엔티티 객체를 대상으로 검색

▼ 영속성 컨텍스트

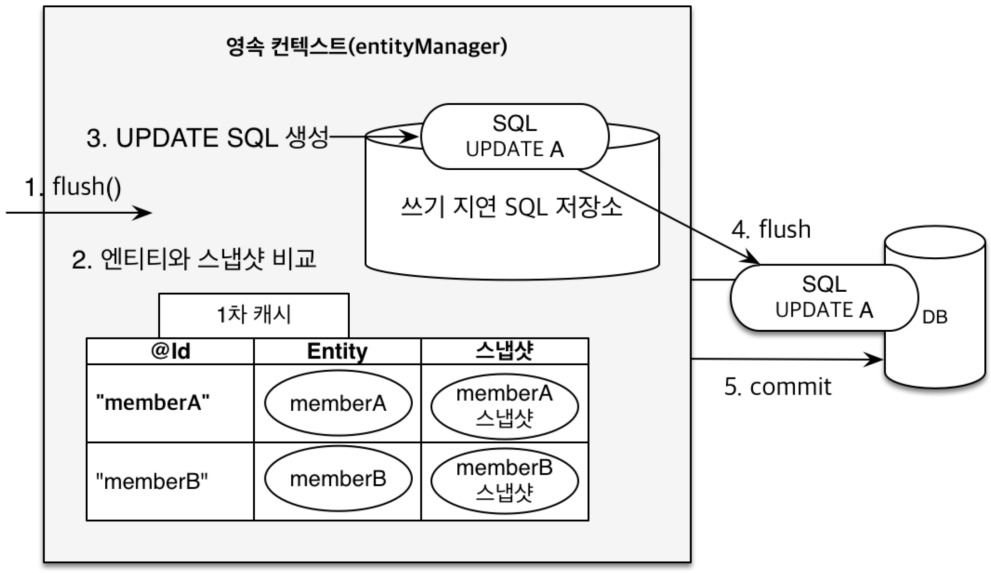
엔티티를 영구 저장하는 환경이라는 뜻

EntityManager.persist(entity);

영속성 컨텍스트는 논리적인 개념으로 눈에 보이지 않는다

엔티티 매니저를 통해 영속성 컨텍스트에 접근

- 엔티티의 생명주기
  - 비영속(new/transient)
    - 영속성 컨텍스트와 전혀 관계가 없는 새로운 상태
  - 영속(managed)
    - 영속성 컨텍스트에 관리되는 상태
  - 준영속(detached)
    - 영속성 컨텍스트에 저장되었다가 분리된 상태
  - 삭제(removed)
    - 삭제된 상태
- 영속성 컨텍스트의 이점
  - 1차캐시
    - 1차 캐시에서 찾음 → 없으면 db 조회 → 1차 캐시에 저장 → 반환
  - 동일성 보장 identity
    - == 비교 true 보장해준다
  - 트랜잭션을 지원하는 쓰기 지연 transactional write-behind
    - 엔티티 매니저는 데이터 변경시 트랜잭션을 시작해야 한다.
    - 커밋하는 순간 데이터베이스에 insert sql을 보낸다
  - 변경 감지 Dirty checking
    - update 이런거 없이 set 으로 변경되면 알아서 저장



        - 최초로 영속성 컨텍스트에 들어온 1차 캐시를 스냅샷 떠준다 → 변경이 있으면 커밋되는 시점에 flush 되면서 스냅샷과 엔티티를 변경하고 변경되면 sql 에 변경
    - 지연 로딩 lazy loading
- 플러시
    - 영속성 컨텍스트의 변경내용을 데이터베이스에 반영
    - 변경감지→수정된 엔티티쓰기 지연 sql저장소에 등록 → 쓰기 지연sql 저장소의 쿼리를 데이터베이스에 전송(등록, 수정,삭제 쿼리)
    - 플러시 하는 방법
        - em.flush() -직접 호출
        - 트랜잭션 커밋 - 플러시 자동 호출
        - JPQL쿼리 실행 -플러시 자동 호출
    - 영속성 컨텍스트를 비우지 않음
    - 영속성 컨텍스트의 변경내용을 데이터 베이스에 동기화
    - 트랜잭션이라는 작업 단위가 중요 → 커밋 직전에만 동기화 하면 됨
- 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