# Spring에 JPA 적용하기


- description :
- author : 도봉산핵주먹
- email : hylee@repia.com
- lastupdate : 2020-12-16


## JPA 사용해야 되는 이유

- 안녕히 계세요 SQL?
- 자바 개발자의 필수 기술 (트렌드에 뒤쳐지지 말자)
  - 안녕히 계세요 마이바티스?
- 객체지향과 관계형DB의 패러다임이 불일치 해결
  - 객체 상속 관계 - Table 슈퍼타입 서브타입 관계
  - 설계는 가능하지만 추가는 어찌 어찌 개발 하겠지만 조회는 답이 없음

## JPA 어려운 이유

- JPA 사용 방법을 알아야 한다. - 진입 장벽이 있음
- 객체와 테이블 설계 매핑이 쉽지 않다.(성능 고려)
- JPA 내부 동작 방식을 제대로 알아야 한다.

## JPA 설계

- 테이블에 맞게 객체를 설계 or 설계에 맞게 테이블 설계

## JPA 요점

- 객체와 관계형 데이터베이스 매핑 (설계 관점, 정적)
- 영속성 컨텍스트 (운영 관점)
  - 장점
    - 1차 캐시 기능 (트랜잭션 내에서)
    - 영속 엔티티의 동일성 보장 (동일한 트랜잭션 내에서)
- 트랙잭션을 지원하는 쓰기 지연
  - 쓰기 지연 SQL 저장소가 있음
  - 개발자에게 최적화의 여지를 둠
- 변경 감지 (Dirty Checking)
- 엔티티 삭제
- 플러시
  - 영속성 컨텍스를 비우지 않으며, 데이터베이스와 동기화라고 보면 됨
- 준영속 상태

- 지연 로딩(Lazy Loading)

# 매핑

```
@Entity : JPA가 관리함을 의미
  public, protected 생성자, 기본 생성자 필수
  @Table(name = "데이터베이스 테이블명")

실행 시점에 테이블 자동 생성 가능
  create, create-drop, update, validate, auto, none
 개발 초기 단계는 create, update
 테스트 서버는 update, validate
 스테이징, 운영 서버 validate, none
  @Column의 옵션은 JPA 로직과는 상관이 없고, 생성시 ddl에 영향을 줌
  updatable = false, 업데이트시에 업데이트 하지 않음
  nullable = false
  unique = true, 생성시 이름이

  sql은 date=날짜, time=시간, tiemstamp만 있음

  GeneratorType.IDENTIFY
  persist()하면 쿼리가 실행됨
  GeneratorType.SEQUENCE
 트랜잭션을 해야해야 쿼리가 실행됨

 실전 예제
```

# 연관관계 매핑

```
'객체지향 설계의 목표는 자율적인 객체들의 협력 공동체를 만드는 것이다.'
- 조영호(객체지향의 사실과 오해), 오브젝트

JPA 포인터 역할
양방향 연관관계의 주인

객체와 테이블의 차이를 이해해야 됨
테이블은 외래키만으로 양방향이 모두 적용됨(조인...)
객체는 그렇지 않음

양방향, 단방향... 양방향이 좋은가? 기본적으로 단방향으로 하는게 좋다.
설계시에 단방향 매핑만으로 이미 연관관계 매핑 완료하자
개발시에 꼭 필요하면 추가 하자.
```

- 연관관계의 주인을 정한다는 것은 외래 키 관리자를 선택하는 것이다. (외래키 등록 수정 가능), 하지만 가짜 매핑에도 넣어주자(양방향 매핑의 경우)
- 외래키가 있는 곳을 주인으로 정하자.
    - 만일 외래키가 없는 곳에서 주인으로 정하고 업데이트를 했는데, 업데이트를 하지 않은 곳의 테이블이 수정이 될 것이며, 이해하기(분석하기) 복잡함
- 주인은 mappedBy 속성을 사용하지 않는다.

- 주인이 아니면 mappedBy 속성을 사용해서 속성의 값으로 연관관계의 주인을 지정해야 한다.

- mappedBy = 매핑되어진, 주인이 아님, 가짜 매핑, 여기에만 값을 넣으면 적용이 안됨

- 연관관계의 주인만이 데이터베이스의 연관관계와 매핑되고 외래 키를 관리할 수 있다.
- 주인이 아닌 곳은 읽기만 할 수 있다.
- 데이터베이스 테이블의 다대일, 일대다 관계에서는 대부분 다 쪽이 외래 키를 가진다. (주인- 진짜 매핑)
- 자동차와 자동차 바퀴
    - 자동차가 중요하지만, 주인은 자동차 바퀴 (1:n)
    - 비즈니스 로직의 중요성으로 주인을 정하지 말고, 외래 키의 위치로 정하자!!

# 양방향 연관관계 주의

- 순수 객체 상태를 고려해서 항상 양쪽에 값을 설정하자.
- 연관관계 편의 메소드를 생성하자
    - 양방향중에 한군데만 하자
    - 
```
    // 양쪽값
public void addMember(MemberHello member) {
    member.setTeam(this);
    members.add(member);
}

    // 양쪽 값
public void changeTeam(Team team) {
    this.team = team;
    team.getMembers().add(this);     // 연관관계 편의 메소드
}
```

- 양방향 매핑시에 무한 루프를 조심하자
    - 예 toString(), lombok, JSON 생성 라이브러리
    - 양방향 호출이 됨
    - lombok로 toString을 사용하지 말자. 컨트롤러에서는 Entity를 반환하지 말자(Api 스펙 변경 문제(⇒ DTO 사용해야 됨), 무한 루프 문제)

# 다양한 연관단계 매핑

- 고려사항
    - 다중성
        - 다대일(@ManyToOne)
            - 다쪽에 외래키가 있고, 주인임, 멤버 - 팀에서 멤버가 주인임, @ManyToOne
            - 연관관계의 주인임 (속성에 mappedBy가 없음)
        - 일대다(@OneToMany)
            - 일이 연관관계의 주인, 엔티티가 관리하는 외래 키가 다른 테이블에 있음 - 업데이트문 실행
            - 실무에서는 거의 없지만, 가능하기는 함, 구지 한다면 다대일 양방향으로 하고 주인을 외래키가 있는 곳으로 지정
            - @JoinColumn을 사용해야 됨. 그렇지 않으면 중간 테이블(조인 테이블)이 생김, 성능, 운영상 관리 포인트 증가
            - 읽기 전용 필드를 사용해서 양방향처럼 사용하는 방법(@JoinColumn(name="", insertable=false, updatable=false)

        - 일대일(@OneToOne)
            - 외래키에 유니크 제약조건이 있어야 됨
            - @JoinColumn을 넣자(디폴트값 지저분함)
            - 양방향도 가능(mappedBy(name = “”))
            - 대상 테이블에 외래 키 단방향은 지원 안됨
            - 자기테이블의 값은 자기가 관리하자
            - 설계시 외래키의 위치는 향후 장기적인 관점에서 판단하자
            - 프록시 기능의 한계로 지연 로딩으로 설정해도 항상 즉시 로딩됨
        - 다대다(@ManyToMany)
            - 관계형 데이터베이스는 정규화된 테이블 2개로 표현을 할 수 없음
            - 연결 테이블을 추가해서 일대다, 다대일 관계로 풀어내야 함
            - 객체는 컬렉션을 사용해서 객체 2개로 다대다 관계 가능
            - 중간 테이블에 필요한 추가 정보를 넣을 수 없음
            - 실무에서는 사용하지 말자!!
            - 실무에서는 연결 테이블을 엔티티로 승격하고 @ManyToMany → @OneToMany, @ManyToOne로 변경
            - 단순한 다대다는 존재하지 않음
            - 조인테이블이라도 PK를 가지자, 될 수 있으면 의미없는 Pk를 쓰자, 향후 유연성이 있음
    - 단방향, 양방향
        - 테이블
            - 외래키하나로 양쪽 조인 가능
            - 방향 개념이 없음
        - 객체
            - 참조용 필드가 있는 쪽으로만 참조 가능
            - 한쪽만 참조하면 단방향
            - 양쪽이 서로 참조하면 양방향 → 사실은 단방향이 두개임

- 연관관계의 주인 ← 양방향일 경우
    - 테이블은 외래키가 하나임, 객체는 참조가 2군데임
    - 객체에서 외래키를 관리할 곳을 정해야됨
    - 외래키를 관리하는 곳이 연관관계의 주인
    - 주인의 반대편은 단순 조회만 함

# 매핑 추가 정보

```
관계형 데이터베이스는 상속관계가 없다

객체는 상속관계는 데이터 베이스에서는 조인, 단일 테이블, 각각테이블로 구현이 가능함
기본 전략은 단일 테이블

@MappedSuperclass - 자주 애용하자
테이블과 관계가 없고 단순히 엔티티가 공통으로 사용하는 매핑 정보를 모으는 역할
abstract(추상 클래스)로 사용

주로 등록일, 수정일, 등록자, 수정자 같은 전체 엔티티에서 공통으로 적용하는 정보를 모을 때 사용

@Entity 클래서는 엔티티나 @MappedSuperclass로 지정한 클래스만 상속 가능
```

# 적용파일 목록

- context-hibernate.xml
- context-jpa-respository.xml

## context-hibernate.xml

> Tip

아래 코드에서 3곳만 고치면 된다.

`<property name="dataSource" ref="" />`

- context-datasource.xml 에 JPA로 사용할 DB정보의 bean ID

`<property name="packagesToScan" value="" >`

- values는 패키지 정보를 입력

`<prop key="hibernate.dialect">`

- JPA가 적용되는 DB 정보와 맞게 입력
- 정보는 아래표 참고

```
<?xml version="1.0" encoding="utf-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:context="http://www.springframework.org/schema/context"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-4.0.xsd
  http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-4.0.xsd">

    <!-- 컨테이너가 관리하는 EntityManager 생성, @PersistenceContext와 함께 사용 -->
    <bean id="entityManagerFactory"
class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">
        <property name="dataSource" ref="egov.dataSource" />
        <!-- 어노테이션 매핑정보 스캔 -->
        <property name="packagesToScan" value="com" />
        <!-- 구현체별 자체 기능을 표준화 -->
        <property name="jpaVendorAdapter">
            <bean
class="org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter">
                <property name="showSql" value="true" />
                <property name="generateDdl" value="true" />
            </bean>
        </property>
```

```
        <!-- persistence.xml 설정정보와 함께 사용가능 -->
        <property name="jpaProperties">
            <props>
                <!-- <prop
key="hibernate.ejb.naming_strategy">org.hibernate.cfg.EJB3NamingStrategy</pr
op> -->
                <prop
key="hibernate.ejb.naming_strategy">org.hibernate.cfg.ImprovedNamingStrategy
</prop>
                <!-- hibernate 5 버전에서의 세팅 (4 버전은 위에껄 사용)-
             <prop key="hibernate.naming.implicit-
strategy">org.hibernate.boot.model.naming.ImplicitNamingStrategyLegacyJpaImp
l</prop>
             <prop key="hibernate.naming.physical-
strategy">org.hibernate.boot.model.naming.PhysicalNamingStrategyStandardImpl
</prop>
             -->
                <prop key="hibernate.hbm2ddl.auto">validate</prop>
                <prop
key="hibernate.dialect">org.hibernate.dialect.Oracle10gDialect</prop>
                <prop key="hibernate.show_sql">true</prop>
                <prop key="hibernate.format_sql">true</prop>
                <prop key="hibernate.use_sql_comments">true</prop>
                <prop key="hibernate.jdbc.batch_size">5</prop>
            </props>
        </property>
    </bean>

</beans>
```

## hibernate.dialect 정보

| | |
|---|---|
| DB2 | org.hibernate.dialect.DB2Dialect |
| DB2 AS/400 | org.hibernate.dialect.DB2400Dialect |
| DB2 OS390 | org.hibernate.dialect.DB2390Dialect |
| PostgreSQL | org.hibernate.dialect.PostgreSQLDialect |
| MySQL5 | org.hibernate.dialect.MySQL5Dialect |
| MySQL5 with InnoDB | org.hibernate.dialect.MySQL5InnoDBDialect |
| MySQL with MyISAM | org.hibernate.dialect.MySQLMyISAMDialect |
| Oracle (any version) | org.hibernate.dialect.OracleDialect |
| Oracle 9i | org.hibernate.dialect.Oracle9iDialect |
| Oracle 10g | org.hibernate.dialect.Oracle10gDialect |
| Oracle 11g | org.hibernate.dialect.Oracle10gDialect |
| Sybase | org.hibernate.dialect.SybaseASE15Dialect |
| Sybase Anywhere | org.hibernate.dialect.SybaseAnywhereDialect |
| Microsoft SQL Server 2000 | org.hibernate.dialect.SQLServerDialect |

| | |
|---|---|
| Microsoft SQL Server 2005 | org.hibernate.dialect.SQLServer2005Dialect |
| Microsoft SQL Server 2008 | org.hibernate.dialect.SQLServer2008Dialect |
| SAP DB | org.hibernate.dialect.SAPDBDialect |
| Informix | org.hibernate.dialect.InformixDialect |
| HypersonicSQL | org.hibernate.dialect.HSQLDialect |
| H2 Database | org.hibernate.dialect.H2Dialect |
| Ingres | org.hibernate.dialect.IngresDialect |
| Progress | org.hibernate.dialect.ProgressDialect |
| Mckoi SQL | org.hibernate.dialect.MckoiDialect |
| Interbase | org.hibernate.dialect.InterbaseDialect |
| Pointbase | org.hibernate.dialect.PointbaseDialect |
| FrontBase | org.hibernate.dialect.FrontbaseDialect |
| Firebird | org.hibernate.dialect.FirebirdDialect |

## context-jpa-respository.xml

> Tip

아래 코드에서 한개만 고치면 된다.

`<jpa:repositories base-package="">`

- base-package="" 는 패키지 정보 입력

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:jpa="http://www.springframework.org/schema/data/jpa"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-4.0.xsd
  http://www.springframework.org/schema/data/jpa
http://www.springframework.org/schema/data/jpa/spring-jpa.xsd">

    <jpa:repositories base-package="com"></jpa:repositories>

</beans>
```

## POM.xml

```
    <properties>
        <egovframework.jpa.version>3.7.0</egovframework.jpa.version>
        <hibernate.version>4.3.11.Final</hibernate.version>
    </properties>
    <dependencies>
        <dependency>
```

```
            <groupId>egovframework.rte</groupId>
            <artifactId>egovframework.rte.psl.data.jpa</artifactId>
            <version>${egovframework.jpa.version}</version>
        </dependency>

        <!-- Hibernate -->
        <dependency>
            <groupId>org.hibernate</groupId>
            <artifactId>hibernate-core</artifactId>
            <version>${hibernate.version}</version>
        </dependency>

        <dependency>
            <groupId>org.hibernate</groupId>
            <artifactId>hibernate-validator</artifactId>
            <version>5.4.2.Final</version>
        </dependency>
        <dependency>
            <groupId>org.springframework.data</groupId>
            <artifactId>spring-data-jpa</artifactId>
            <version>1.11.10.RELEASE</version>
            <exclusions>
                <exclusion>
                    <groupId>org.slf4j</groupId>
                    <artifactId>slf4j-api</artifactId>
                </exclusion>
                <exclusion>
                    <groupId>org.slf4j</groupId>
                    <artifactId>jcl-over-slf4j</artifactId>
                </exclusion>
            </exclusions>
        </dependency>
    </dependencies>
```

## Builder 사용하기

- [SpringBoot + JPA] Lombok @Builder 빌더패턴 적용기

도봉산핵주먹, spring, jpa, 주레피

From:
http://125.132.25.164/dokuwiki/ - **문제를 잘 정의하는 것은 문제를 절반 해결한 것이다. - 2023.12**

Permanent link:
**http://125.132.25.164/dokuwiki/doku.php?id=wiki:spring:jpa&rev=1656597747**

Last update: **2022/06/30 23:02**