Java Persistence Framework

Mybatis(sql mapper framework)
JPA+hibernate(ORM framework)

<< 목차 >>

1. Mybatis

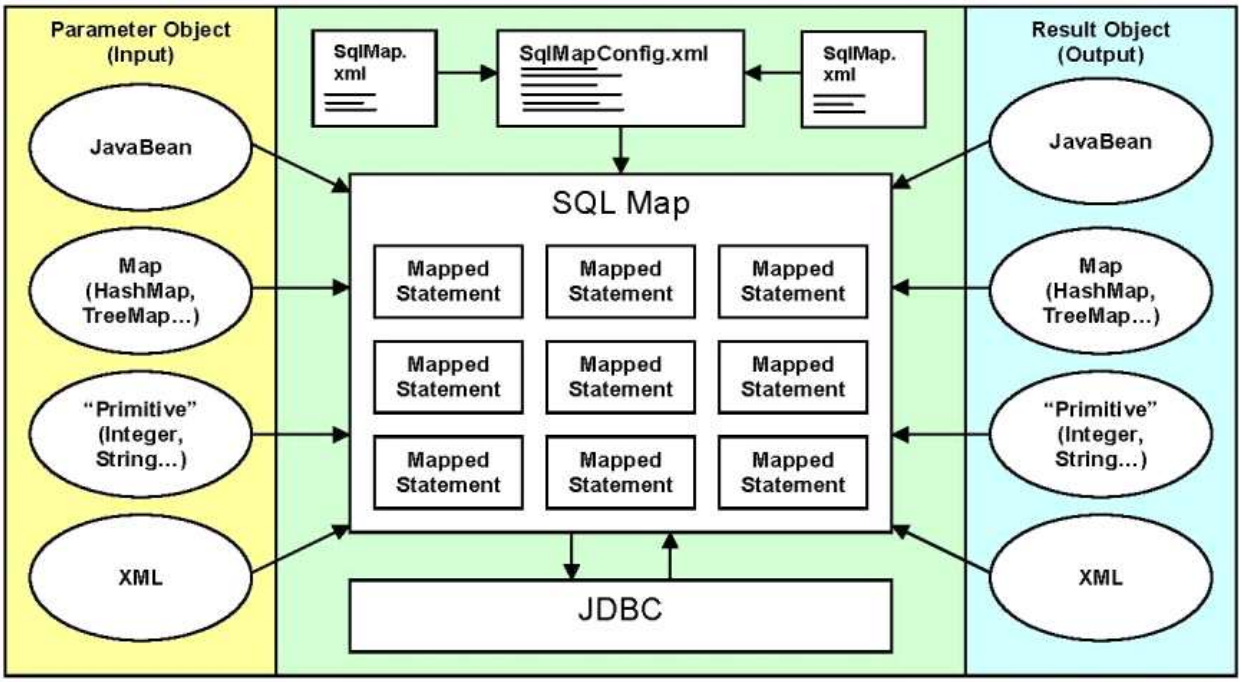
참고사이트 : https://blog.mybatis.org/

마이바티스는 자바 오브젝트와 SQL, 저장프로시저 그리고 몇 가지 고급 매핑을 지원하는 퍼시스턴스 프레임워크이다. 마이바티스는 JDBC로 처리하는 상당부분의 코드와 파라미터 설정 및 결과 매핑을 대신해 준다.

가. 특징

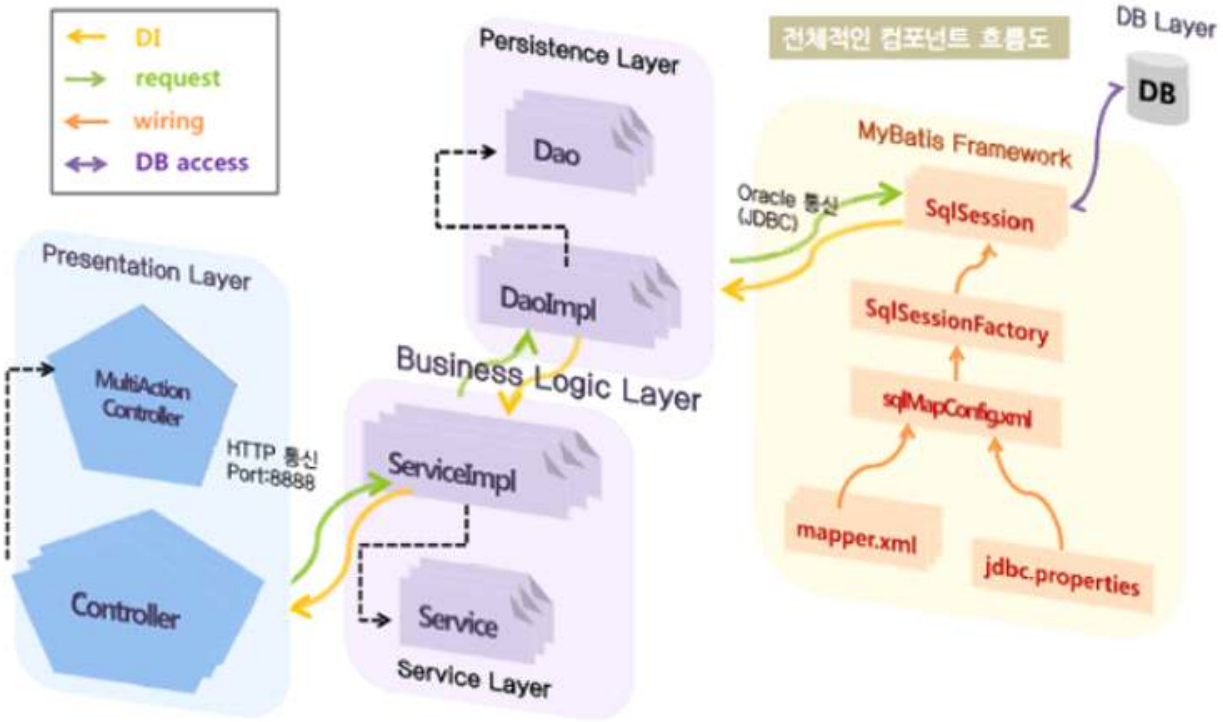
(1) SQL문을 이용하여 RDB에 접근, 데이터를 오브젝트(객체)화 시켜준다.
(2) SQL 명령어를 자바코드에서 분리하여 XML로 따로 관리
(3) JDBC의 모든 기능을 MyBatis가 대부분 제공하므로 한 두 줄의 자바 코드로 DB 연동을 처리. XML 파일에 저장된 SQL 명령어를 대신 실행하고 실행결과를 VO 같은 자바 객체에 자동으로 매핑까지 해 준다.

나. 개요



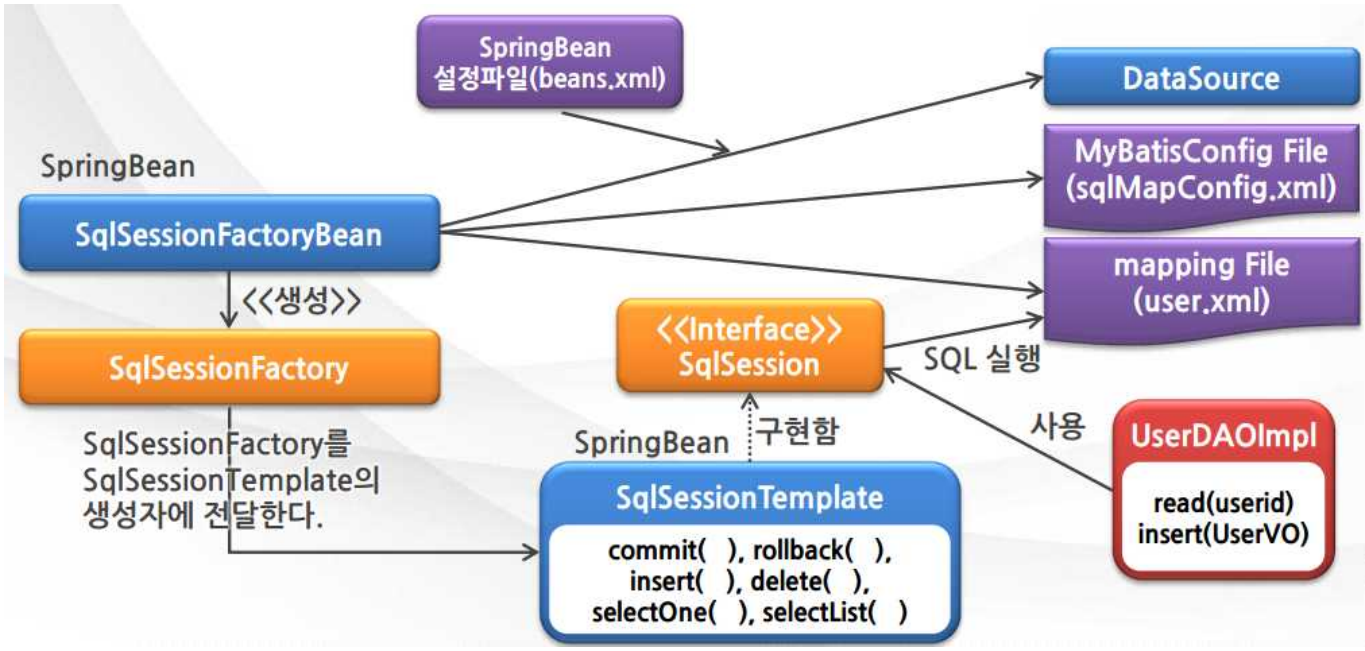
(1) sqlMapConfig
(2) sqlMap
(3) SqlSession

다. MyBatis3 주요 컴포넌트



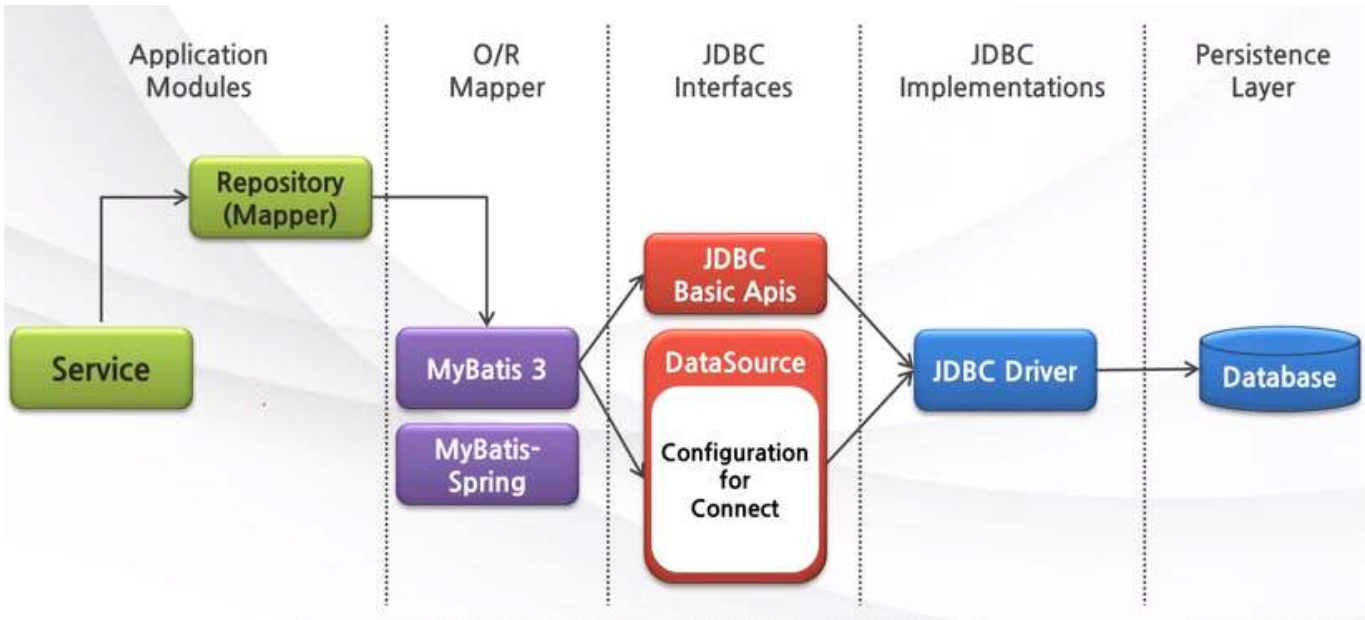
MyBatis 설정파일 (SqlMapConfig.xml)	▪ 데이터소스, Mapping 파일의 경로, 쓰레드 관리와 같은 설정을 제공. ▪ <settings> 요소는 XML파일을 빌드하는 SqlMapClient 인스턴스를 위해 다양한 옵션과 최적화를 설정하도록 한다. ▪ typeAlias 요소는 긴 전체 경로를 포함한 클래스명을 참조하기 위한 짧은 이름을 명시하도록 한다..
SqlSessionFactoryBean	▪ MyBatis 설정 파일을 바탕으로 SqlSessionFactory를 생성한다. ▪ SqlSessionFactoryBean을 Bean 등록할 때 DataSource 정보와 MyBatis Config 파일정보, Mapping 파일의 정보를 함께 설정한다.
SqlSessionFactory	▪ SqlSession을 생성한다.
SqlSession	▪ 핵심적인 역할을 하는 클래스로서 SQL 실행이나 트랜잭션 관리를 실행한다. ▪ SqlSession 오브젝트는 Thread-Safe 하지 않으므로 thread마다 필요에 따라 생성한다.
mapper 파일	▪ SQL문과 OR Mapping을 설정한다.

라. MyBatis-Spring 주요 컴포넌트



SqlSessionTemplate	▪ 핵심적인 역할을 하는 클래스로서 SQL 실행이나 트랜잭션 관리를 실행한다. ▪ SqlSession 인터페이스를 구현하며, Thread-safe하다. ▪ Spring Bean으로 등록해야 한다.

마. MyBatis와 MyBatis-Spring을 사용한 DB 액세스 Architecture



바. Mybatis – spring 연동 설정

(1) 라이브러리 설치 - pom.xml 설정

- spring-jdbc : Spring 에서 지원하는 JDBC
- commons-dbcp2 : 커넥션풀을 담당하는 Apache Commons DBCP
- mybatis : myBatis 코어 라이브러리
- mybatis-spring : Spring 에서 연동을 지원하는 myBatis

```
<!-- spring-jdbc -->
<dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-jdbc</artifactId>
        <version>${org.springframework-version}</version>
</dependency>
<!-- commons-dbcp -->
<dependency>
        <groupId>org.apache.commons</groupId>
        <artifactId>commons-dbcp2</artifactId>
        <version>2.7.0</version>
</dependency>

<!-- Mybatis -->
<dependency>
        <groupId>org.mybatis</groupId>
        <artifactId>mybatis</artifactId>
        <version>3.5.4</version>
</dependency>

<!-- Mybatis-spring -->
<dependency>
        <groupId>org.mybatis</groupId>
        <artifactId>mybatis-spring</artifactId>
        <version>2.0.4</version>
</dependency>
```

▣ 오라클 드라이버 설치 (optional)

```
<!-- 오라클 JDBC 드라이버 -->
<dependency>
    <groupId>com.oracle.ojdbc</groupId>
    <artifactId>ojdbc8</artifactId>
    <version>19.3.0.0</version>
</dependency>
```

- 로컬 시스템에 있는 드라이버를 지정하는 경우 : scpoe와 systemPath를 이용하여 지정함.

```
  <dependency>
        <groupId>ojdbc</groupId>
        <artifactId>ojdbc</artifactId>
        <version>6</version>
        <scope>system</scope>
        <systemPath>${basedir}/WebContent/WEB-INF/lib/ojdbc6.jar</systemPath>
   </dependency>
```

(2) jdbc.properties 설정

ORACLE 접속 정보 설정 프로퍼티를 src/main/resource 경로 아래 생성한다. RDBMS 드라이브, 접속경로, 계정, 암호 순으로 항목을 입력한다.

```
jdbc.driver = oracle.jdbc.OracleDriver
jdbc.url = jdbc:oracle:thin:@localhost:1521:xe
jdbc.username = spring
jdbc.password = spring
```

(3) datasource-context.xml

src/main/resource 경로에 config/spring/context 폴더 생성 후 파일 생성

```
<bean id="jdbcProp" class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
  <property name="location" value="classpath:jdbc.properties" />
</bean>

<bean id="dataSource" class="org.apache.commons.dbcp2.BasicDataSource" destroy-method="close">
  <property name="driverClassName" value="${jdbc.driver}" />
  <property name="url" value="${jdbc.url}" />
  <property name="username" value="${jdbc.username}" />
  <property name="password" value="${jdbc.password}" />
</bean>
```

(4) mybatis-context.xml

```
<!-- SqlSession setup for MyBatis Database Layer Spring과 Mybatis 연동 설정 -->
<bean id="sqlSession" class="org.mybatis.spring.SqlSessionFactoryBean">
    <property name="dataSource" ref="dataSource" />
    <property name="configLocation" value="classpath:sql-map-config-spring.xml" />
    <property name="mapperLocations" value="classpath:/mappings/*.xml" />
</bean>

<!-- SqlSessionTemplate  -->
<bean class="org.mybatis.spring.SqlSessionTemplate">
    <constructor-arg ref="sqlSession"></constructor-arg>
</bean>

<!-- MapperConfigurer setup for MyBatis Database Layer
     with @MapperScan("boardMapper") in BoardMapper Interface -->
<bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
    <property name="basePackage" value="com.dbal.app.**.impl" />
</bean>
```

(가) sqlSessionFactoryBean

SqlSessionFactory를 생성하기 위한 클래스입니다. 이 빈을 사용해서 Spring은 SqlSessionFactory 객체를 한 번만 생성하게 됩니다. 그리고 MyBatis를 사용할 때마다 SqlSessionFactory를 이용해서 MyBatis 객체를 매번 생성하게 됩니다.

dataSource와 configLocation 두 가지만 설정하더라도 잘 동작하며 추가적으로 mapperLocation 지정가능하다. mapperLocations 노드에서 스캔하려는 맵퍼의 경로 범위를 설정하면 경로와 일치하는 맵퍼를 모두 호출하게 된다.

(나) sqlSessionTemplate

sqlSessionTemplate은 MyBatis의 SqlSession과 같은 역할을 담당하지만 트랜잭션을 처리하는 방법에서 약간의 차이점이 있다. MyBatis의 SqlSession은 트랜잭션 처리를 위해 commit/rollback 메소드를 명시적으로 호출해야 하지만, SqlSessionTemplate은 Spring이 트랜잭션을 대신 처리하게 구조화돼 있기 때문에 commit/rollback 메소드를 호출할 수 없게 되어있다.

(다) mapperScannerConfigurer

매퍼 인터페이스를 자동으로 검색하여 빈으로 등록한다. 매퍼 인터페이스의 패키지 중 가장 상위 패키지를 지정해주면 그 하위에 있는 매퍼 인터페이스를 모두 등록합니다. 여기서 자동 검색 대상이 되는 인터페이스는 메소드를 한 개 이상 갖는 인터페이스만이 대상이며, 클래스는 대상에서 제외된다.

(5) sql-map-config.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE configuration PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
    "http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>
    <settings>
        <setting name="cacheEnabled"                value="true"/>
        <setting name="lazyLoadingEnabled"          value="false"/>
        <setting name="multipleResultSetsEnabled"   value="true"/>
        <setting name="useColumnLabel"              value="true"/>
        <setting name="useGeneratedKeys"            value="false"/>
        <setting name="defaultExecutorType"         value="SIMPLE"/>
        <setting name="defaultStatementTimeout"     value="25000"/>
        <!-- 오라클 컬럼이 first_name 를 VO firstName 처럼 카멜케이스로 변환  -->
        <setting name="mapUnderscoreToCamelCase" value="true"/>
    </settings>

   <!-- Alias 설정 -->
    <typeAliases>
        <typeAlias alias="board" type="com.springbook.biz.board.BoardVO"/>
    </typeAliases>

    <typeHandlers>
      <!--  java.sql.Timestamp 를 java.util.Date 형으로 반환 -->
      <typeHandler javaType="java.sql.Timestamp"  handler="org.apache.ibatis.type.DateTypeHandler"/>
      <typeHandler javaType="java.sql.Date"       handler="org.apache.ibatis.type.DateTypeHandler"/>
    </typeHandlers>
</configuration>
```

TypeHandler : 파라미터를 PreparedStatement에 세팅하거나 ResultSet으로부터 값을 추적할 때마다 타입 핸들러는 알맞은 자바 타입의 값을 추적하기 위해 사용

(6) web.xml

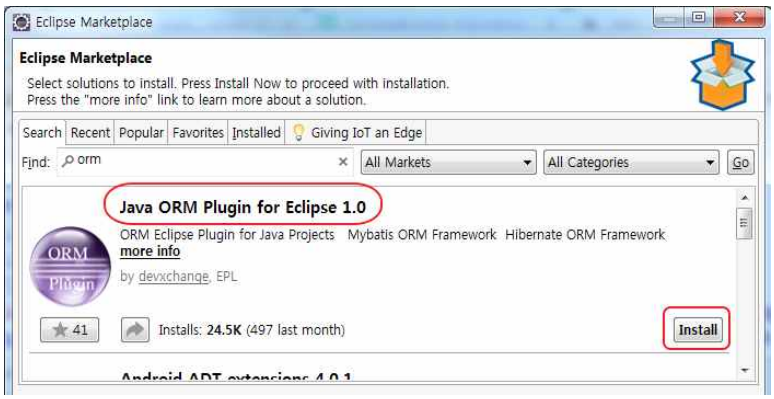
```
<context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>classpath:/config/*-context.xml</param-value>
</context-param>

<listener>
        <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>
```

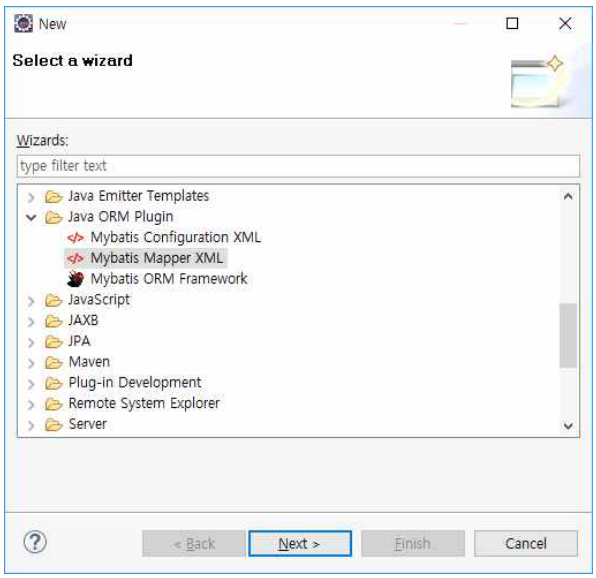
(7) mapper 파일 작성

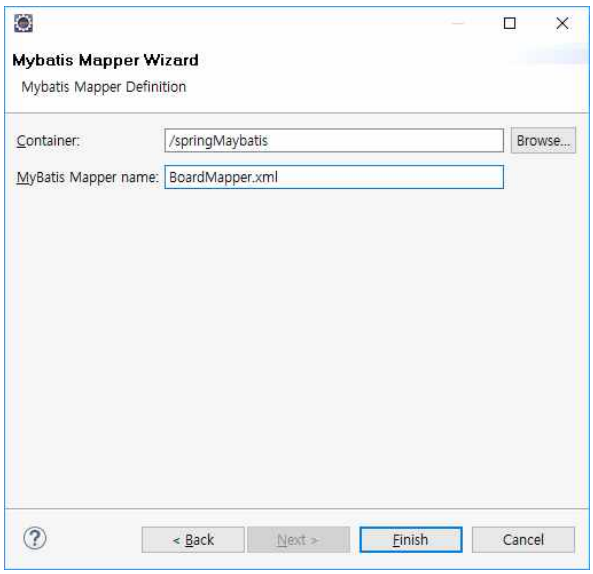
(가) java ORM plugin plugin설치

1) MaBatis와 관련된 복잡한 XML 설정 파일들을 자동으로 만들고 관리할 수 있다.
2) egovframework을 사용하면 플러그인이 설치되어 있음.(없으면 설치)
3) eclipse -> market place에서 orm 검색하여 설치



(나) 매퍼파일 생성 : File 메뉴 -> new -> other -> Mybatis Mapper XML
컨테이너로 현재 실행할 프로젝트 지정하고 파일명을 입력






```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
 "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.dbal.app.emp.map.EmpMapper">

    <select id="getEmp" resultType="com.dbal.app.emp.map.EmpVO"
                        parameterType="com.dbal.app.emp.map.EmpVO">
        select *
        from employees
        where employees_id = #{employees_id}
    </select>

    <select id="getEmpList" resultType="com.dbal.app.emp.map.EmpVO">
        select *
        from employees
        order by first_name
    </select>

    <insert id="empInsert" parameterType="com.dbal.app.emp.map.EmpVO">
        insert into employees(employee_id,
                              first_name,
                              last_name,
                              email,
                              hire_date,
                              job_id )
                      values ( #{employee_id},
                              #{first_name},
                              #{last_name},
                              #{email},
                              #{hire_date},
                              #{job_id} )
    </insert>
</mapper>
```

(8) Mapper interface 작성

```
package com.dbal.app.board.mapper.BoardMapper;

public interface EmpMapper {

        public EmpVO getEmp(EmpVO empVO);
        public List<EmpVO> getEmpList();
        public void empInsert(EmpVO empVO);

}
```

EmpMapper.java

```
1  package com.dbal.app.emp.mapper;
2
3  import java.util.List;
8
9  public interface EmpMapper {
10
11     public EmpVO getEmp(EmpVO empVO);
12     public List<EmpVO> getEmpList();
13     public void empInsert(EmpVO empVO);
14 }
```

emp_mapper.xml

```
4  <mapper namespace="com.dbal.app.emp.mapper.EmpMapper">
5
6      <select          id="getEmp"
7                  resultType="com.dbal.app.emp.map.EmpVO"
8              parameterType="com.dbal.app.emp.map.EmpVO">
9          select *
10         from employees
11         where employees_id = #{employees_id}
12     </select>
13
14     <select id="getEmpList" resultType="com.dbal.app.emp.map.EmpVO">
```

(9) DAO 클래스 작성

Mapper 인터페이스 대신에 DAO 클래스를 구현할 수도 있다.

```
package com.dbal.app.emp.mapper;

import java.util.List;
import org.mybatis.spring.SqlSessionTemplate;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Repository;

@Repository
public class EmpDAO {

        @Autowired private SqlSessionTemplate mybatis;

        public EmpVO getEmp(EmpVO empVO) {
                return mybatis.selectOne("com.dbal.app.emp.map.EmpMapper.getEmp",empVO );
        }

        public List<EmpVO> getEmpList() {
                return mybatis.selectList("com.dbal.app.emp.map.EmpMapper.getEmpList");
        }
        public void empInsert(EmpVO empVO) {
                mybatis.insert("com.dbal.app.emp.map.EmpMapper.empInsert", empVO);
        }
}
```

(10) junit을 이용한 단위 테스트

(가) pom.xml junit 버전 4.12로 지정

(나) build path에 junit 추가

(다) 테스트 클래스 작성

```
package com.dbal.app.emp;

import java.util.List;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

import com.dbal.app.emp.mapper.EmpMapper;
import com.dbal.app.emp.mapper.EmpVO;

@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations = "classpath:/config/*-context.xml")

public class EmpMapperClient {

	@Autowired EmpMapper mapper;

	@Test
	public void empListTest() {
		List<EmpVO> list = mapper.getEmpList();
		System.out.println(list);
	}
}
```

사. mapper xml

1. 기본 자료형 비교

연산자	자바	비고
eq	==	
neq	!=	
gt	>	
gte	>=	
not	!	
lt	<	사용안됨
lte	<=	사용안됨
and	&&	사용안됨
or	||	사용안됨

1) 프로퍼티 속성 값과 기준 값이 동일한 경우.

```
<if test="value == 1"></if>
 <if test="value eq 1"></if>
```

2. 참조 자료형 비교

1) 프로퍼티 속성 값이 널인 경우.

```
<if test="value == null"></if>
<if test="value eq null"></if>
```

2) 프로퍼티 속성 값이 널 또는 빈 문자인 경우.

```
<if test="value == null or value == ''">/if>
<if test="value eq null or value eq ''"></if>
```

3. 문자열 비교

1) 프로퍼티 속성 값이 기준 한 자리 문자열과 같은 경우.

```
<if test="value == 'a'.toString()">
<if test="value eq 'a'.toString()">
<if test='value == "a"'>
```

2) 프로퍼티 속성 값이 기준 한 자리 이상 문자열과 같은 경우.

```
<if test="value == 'abc'">
<if test="value eq 'abc'">
<if test='value == "abc"'>
```

3. 자바메서드 사용

1) 비교

```
<if test='!paraName1.equals("all")'>
```

2) 대소문자

```
<if test='paraName1 !=null  and paraName1.equalsIgnoreCase("test")'>
```

아. 트랜잭션

(1) 트랜잭션이란

두 개 이상의 쿼리를 한 작업으로 실행해야 할 때 사용. 여러 쿼리를 논리적으로 하나의 작업으로 묶어준다. 트랜잭션으로 묶인 쿼리 중 하나라도 실패하면 전체 쿼리를 실패로 간주하여 롤백처리하고 모두 성공하면 커밋을 한다.

```
public void insert(EmpVO vo) {
    try {
        //1. connect
        conn = ds.getConnection();

        //트랜잭션 범위 시작
        conn.setAutoCommit(false);

        //2. statement
        String sql = "INSERT INTO EMPLOYEES  ";
        PreparedStatement pstmt = conn.prepareStatement(sql);
        pstmt.executeUpdate();

        sql = "INSERT INTO MEMBER  ";
        pstmt = conn.prepareStatement(sql);
        int r = pstmt.executeUpdate();

        //커밋 : 트랜잭션 범위 종료
        conn.commit();

    } catch(Exception e){
        if(conn != null)
            //롤백 : 트랜잭션 범위 종료
            try { conn.rollback(); } catch (SQLException e1) {  }
    } finally {
        if(conn != null)
            try { conn.close(); } catch (SQLException e1) {  }
    }
}
```

(2) @Transactional annotation 을 이용한 트랜잭션 처리

(가) transaction-context.xml 설정

```
<!-- TransactionManager bean 등록 -->
<bean id="txManager"
    class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
    <property name="dataSource" ref="dataSource" />
</bean>

<!-- @Transactional 어노테이션 처리 -->
<tx:annotation-driven transaction-manager="transactionManager" />
```

(나) Transaction 처리하고 싶은 메소드 위에 @Transactional 어노테이션을 지정.

- @Transactional 은 public 메소드에서만 정상 작동한다.
- @Transactional 을 달아놓은 메소드가 동일한 클래스 내의 다른 메소드에 의해 호출된다면 트랜잭션이 정상 작동하지 않는다.
- 트랜잭션도 공통 기능 중 하나로 내부적으로 AOP를 사용하고 프록시를 통해서 이루어진다.

(다) @Transaction의 주요 속성

속성	설명	주요 값
value		transaction manager bean 이름
propagation	트랜잭션 전파타입	REQUIRED: 트랜잭션 필요. 진행 중인 트랜잭션이 존재하지 않으면 생성 MANDATORY: 진행 중인 트랜잭션이 없으면 익셉션이 발생 REQUIRES_NEW: 항상 새로운 트랜잭션을 시작
isolation	트랜잭션 격리레벨	DEFAULT: 기본설정을 사용 READ_UNCOMMITTED: 다른 트랜잭션이 커밋하지 않은 데이터를 읽을 수 있다. READ_COMMITTED: 다른 트랜잭션이 커밋한 데이터를 읽을 수 있다. REPEATABLE_READ SERIALIZABLE: 동일한 데이터에 대해서 동시에 두 개 이상의 트랜잭션을 수행할 수 없다.
timeout	제한시간	

(3) AOP 설정인 경우

```
<!-- TransactionManager bean 등록 -->
<bean id="txManager"
    class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
    <property name="dataSource" ref="dataSource" />
</bean>

<!-- TransactionManager  적용범위 -->
<tx:advice id="txAdvice" transaction-manager="txManager">
    <tx:attributes>
        <tx:method name="get*" read-only="true" />
        <tx:method name="insert" rollback-for="Exception" />
        <tx:method name="*" />
    </tx:attributes>
</tx:advice>

<!-- TransactionManager  AOP 설정 -->
<aop:config>
    <aop:pointcut id="allPointcut"
        expression="execution(* com.yedam..*Impl.*(..))" />
    <aop:advisor pointcut-ref="allPointcut"
        advice-ref="txAdvice" />
</aop:config>
```

read-only = true : insert / update / delete 쿼리를 내부적으로 실행시 exception 을 뱉는다.

* roll-backfor = Exception : Exception (Exception 밑의 모든 Exception 포함) 이 발생할 경우 rollback 처리. (no rollback for 옵션으로 특정 Exception이 발생시 rollback 처리를 하지 않게 처리 가능)

propagation : 트랜잭션의 전파속성으로써 메소드 내에서 다른 메소드를 사용할 때 하나의 트랜잭션으로 묶을지, 별도의 트랜잭션으로 분류할지 등 과 같은 설정을 지정하는 옵션. default 값이 required 로 알고 있어 위와 같이 선언할 필요는 없는 걸로 알고 있다.. 여러가지 설정이 있는데 나중에 정리하는걸로.

2. jdbc 로그 설정

가. pom.xml 라이브러리 설정 추가

```
        <!-- log jdbc -->
        <dependency>
                <groupId>org.lazyluke</groupId>
                <artifactId>log4jdbc-remix</artifactId>
                <version>0.2.7</version>
        </dependency>
```

나. log4j.xml

```
    <logger name="jdbc.sqltiming" additivity="false">
        <level value="DEBUG"/>
        <appender-ref ref="console"/>
    </logger>
     <logger name="jdbc.sqlonly" additivity="false">
        <level value="DEBUG"/>
        <appender-ref ref="console"/>
    </logger>
<!--      <logger name="jdbc.resultset" additivity="false">
        <level value="DEBUG"/>
        <appender-ref ref="console"/>
    </logger> -->
    <logger name="jdbc.resultsettable" additivity="false">
        <level value="DEBUG"/>
        <appender-ref ref="console"/>
    </logger>
```

- jdbc.sqlonly : sql 구문이 정렬되어 보기 좋게 출력. PreparedStatement일 경우 관련된 argument 값으로 대체된 SQL문이 보여진다.
- jdbc.sqltiming : sql 문과 실행시간 출력
- jdbc.resultset : ResultSet을 포함한 모든 JDBC 호출 정보를 로그로 남기므로 매우 방대한 양의 로그가 생성된다.
- jdbc.resultsettable : 실행 결과를 테이블형식으로 확인

다. context-datasource.xml

기존 datasource 빈의 id를 "dataSourceSpied" 로 변경하고
아래의 id 가 "dataSource" 빈을 등록하여 설정한다.

```
    <bean id="jdbcProp"
            class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
            <property name="location" value="classpath:global.properties" />
    </bean>

    <bean id="dataSourceSpied"
            class="org.apache.commons.dbcp.BasicDataSource" destroy-method="close">
            <property name="driverClassName" value="${jdbc.driver}" />
            <property name="url" value="${jdbc.url}" />
            <property name="username" value="${jdbc.username}" />
            <property name="password" value="${jdbc.password}" />
    </bean>

    <bean id="dataSource"
            class="net.sf.log4jdbc.Log4jdbcProxyDataSource">
            <constructor-arg ref="dataSourceSpied" />
            <property name="logFormatter">
                    <bean class="net.sf.log4jdbc.tools.Log4JdbcCustomFormatter">
                            <property name="loggingType" value="MULTI_LINE" />
                            <property name="sqlPrefix" value="SQL:::" />
                    </bean>
            </property>
    </bean>
```

라. 로그확인

```
INFO : jdbc.sqlonly - SQL:::SELECT *
                      FROM BOARD
                      WHERE 1 = 1
                      ORDER BY SEQ DESC

DEBUG: jdbc.sqltiming -  org.apache.ibatis.executor.statement.PreparedStatementHandler.query(PreparedStatementHandler.java:62)
1. SELECT * FROM BOARD WHERE 1 = 1 ORDER BY SEQ DESC  {executed in 18 msec}

DEBUG: jdbc.resultset - 1. ResultSet.new ResultSet returned

INFO : jdbc.resultsettable - |----|----|--------------------|-----------|--------|------|-----------------|--------------|
INFO : jdbc.resultsettable - |SEQ |CNT |CONTENT             |REGDATE    |TITLE   |WRITER |ORIGINALFILENAME |UPLOADFILENAME |
INFO : jdbc.resultsettable - |----|----|--------------------|-----------|--------|------|-----------------|--------------|
INFO : jdbc.resultsettable - |22  |0   |22                  |[null]     |22      |22     |[null]           |[null]         |
```

3. 게시판 만들기

가. 개발환경

JAVA 1.8
Apache Tomcat 8.x
eclipse 2020-06 (4.16.0)
Oracle 11g
Spring 5.2.7
Maven 3.6.0
myBatis 3.5.4
jQuery 2.2.3
loombok

나. 테이블 생성

```
create table board (
  seq number primary key,
  title varchar2(100),
  wid varchar2(20),
  wdate date,
  contents varchar2(1000)
);

insert into board(seq, title) values (1, 'test1');
insert into board(seq, title) values (2, 'test2');
commit;
```

다. Board.java

```
package com.dbal.app.board;
@Data
public class Board {
	private String seq;
	private String title;
	private String wid;
	private String wdate;
	private String contents;

	@Builder
	public Board(String seq, String title, String wid, String wdate, String contents){
		this.seq = seq;
		this.title = title;
		this.wid = wid;
		this.wdate = wdate;
		this.contents = contents;
	}
}
```

라. BoardMapper.xml

```
<?xml version="1.0" encoding="UTF-8"
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.dbal.app.board.impl.BoardMapper">

    <select id="selectall" resultType="com.dbal.app.board.Board">
```

```
        SELECT  *
          FROM board
         ORDER BY seq DESC
    </select>

    <select id="selectone" resultType="com.dbal.app.board.Board">
        SELECT  *
          FROM board
         WHERE seq = #{seq}
    </select>

    <insert id="insert" parameterType="com.dbal.app.board.Board">
         INSERT INTO board (    seq,
                                wid,
                                title,
                                contents,
                                wdate )
                    VALUES (    (select nvl(max(seq),0)+1 from board),
                                #{wid},
                                #{title},
                                #{contents},
                                sysdate  )
    </insert>

    <update id="update">
         UPDATE board SET
                title = #{title},
                contents = #{contents}
          WHERE seq = #{seq}
    </update>

    <delete id="delete">
         DELETE FROM board
          WHERE seq = #{seq}
    </delete>
</mapper>
```

마. BoardMapper.java

Mapper interface를 사용하는 경우에 사용. 아니면 아래의 BoardDAO를 주로 사용함.

```
package com.company.board.mapper;

import java.util.List;
import org.springframework.stereotype.Repository;

@Repository
public interface BoardMapper {
    List<Board> selectall();
    Board selectone(int seq);
    void insert(Board board);
    void update(Board board);
    void delete(int seq);
}
```

바. BoardDAO

```
package com.company.board.mapper;

import java.util.List;

@Repository
public class BoardDAO {

    @Autowired
    private SqlSessionTemplate session;

     public List<Board> selectall(){
          return session.selectList("board.selectall", seq);
      }
    public Board selectone(int seq) {
          return session.selectOne("board.selectone", seq);
    }
    public void insert(Board board){
         session.insert("board.insert", board);
    }
    public void update(Board board){
         session.insert("board.update", board);
    }
    public void delete(int seq){
         session.insert("board.delete", board);
   }
}
```

사. BoardService.java

```
package com.company.board.service;

import java.util.List;

public interface BoardService {

    public List<Board> getSelectall() ;
    public Board getSelectone(int seq) ;
    public void insert(Board board) ;
    public void update(Board board) ;
    public void delete(int seq) ;

}
```

아. BoardServiceImpl.java

```
package com.company.board.impl;

import java.util.List;
import javax.annotation.Resource;
import org.springframework.stereotype.Service;

@Service
public class BoardServiceImpl implements BoardService {

    @Resource(name = "boardMapper")
    private BoardMapper boardMapper;

    public List<Board> getSelectall() {
        return this.boardMapper.selectall();
    }

    public Board getSelectone(int seq) {
        return this.boardMapper.selectone(seq);
    }
    public void insert(Board board) {
         this.boardMapper.insert(board);
    }
    public void update(Board board) {
         this.boardMapper.update(board);
    }
    public void delete(int seq) {
         this.boardMapper.delete(seq);
    }
}
```

자. 컨트롤러

BoardController .java

```
package com.company.board;
import java.util.List;
import java.util.Locale;
import javax.annotation.*;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.servlet.mvc.support.RedirectAttributes;

@Controller
public class BoardController {
```

```
        private static final Logger logger = LoggerFactory.getLogger(BoardController.class);

    // Resource 어노테이션을 이용하여 BoardService 선언.
    @Resource(name = "boardService")
    private BoardService boardService;

        @RequestMapping(value = "/board/list.do", method = RequestMethod.GET)
        public String list(Locale locale, Model model) {
                List<Board> list = this.boardService.getSelectall();
        model.addAttribute("list", list);
                return "/board/boardList";
        }

        // 게시판 상세보
    // PathVariable 어노테이션을 이용
    // bbs/1 -> seq = 1;
    // 일반 적으로 (@ReuqstParam(value = "board", required = false, defaultValue = "0"), int seq, Model model)
    @RequestMapping("/board/{seq}")
    public String select(@PathVariable int seq, Model model) {
        logger.info("board view seq = {}", seq);
        Board object = this.boardService.getSelectOne(seq);

        model.addAttribute("bean", object);
        return "/board/boardDetail";
    }

    // 게시판 쓰기
    @RequestMapping(value = "/board/insertForm.do", method = RequestMethod.GET)
    public String insertForm(@RequestParam(value="seq", defaultValue="0") int seq, Model model) {

        logger.info("board insert");

        if (seq > 0) {
            Board object = this.boardService.getSelectOne(seq);
            model.addAttribute("bean", object);
        }
        return "/board/insertForm";
    }

    @RequestMapping(value = "/board/insert.do", method = RequestMethod.POST)
    public String insert(@ModelAttribute("board") Board board, RedirectAttributes redirectAttributes) {
        String seq = board.getSeq();

        if (seq == null || seq.isEmpty()) {
            this.boardService.insert(board);
            redirectAttributes.addFlashAttribute("message", "추가되었습니다.");
            return "redirect:/board/list.do";
        } else {
            this.boardService.update(board);
            redirectAttributes.addFlashAttribute("message", "수정되었습니다.");
            return "redirect:/board/insertForm.do?seq=" + seq;
        }
    }
```

```
    @RequestMapping(value = "/board/delete.do", method = RequestMethod.POST)
    public String delete(@RequestParam(value = "seq", required = true) int seq) {
        this.boardService.delete(seq);
        return "redirect:/board/list.do";
    }
}
```

차. view page

webapp/WEB-INF/views/board/boardList.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
<h3>게시판</h3>
<a href="insertForm.do">글등록</a>
<c:forEach items="${list}" var="board">
        ${board.seq}  ${board.title} <br>
</c:forEach>
</body>
</html>
```

4. 프로시저 연동

가. jdbc 이용

- dao

```
    public boolean delete(int no) {
        try {
            conn = getConnection();
            CallableStatement cstmt = conn.prepareCall( "call emp_delete(?)");
            cstmt.setInt(1, no);
            cstmt.executeUpdate();
            return true;
        } catch(Exception e) {
            System.out.println("DB delete Error : " + e);
            return false;
        } finally {
            disconnect();
        }
    }
```

- prodecure emp_delete

```
CREATE OR REPLACE PROCEDURE EMP_DELETE (
 V_EMPNO IN NUMBER
)
IS

BEGIN
  DELETE FROM EMP
  WHERE EMPNO = V_EMPNO;
END
```

나. jdbc : 리턴값이 있는 경우

- dao

```
    public HashMap<String, Object> insertProc(Comments bean) throws Exception {
        CallableStatement cstmt = null ;
        int nextId = 0;
        HashMap<String, Object> map = new HashMap<String, Object>();
        try {
            connect();
            conn.setAutoCommit(false);

            cstmt = conn.prepareCall("{call COMMENTS_INS(?,?,?,?,?)}");
            cstmt.setString(1, bean.getName());
            cstmt.setString(2, bean.getContent());
            cstmt.setString(3, "1");
            cstmt.registerOutParameter(4, java.sql.Types.NUMERIC);
            cstmt.registerOutParameter(5, java.sql.Types.VARCHAR);
            cstmt.executeUpdate();
            conn.commit();

            nextId = cstmt.getInt(4);
            String out_msg = cstmt.getString(5);

            if (nextId != 0) {
                map.put("id", nextId);
                map.put("name", bean.getName());
                map.put(("content"), bean.getContent());
            } else {
                map.put("id", 0);
                map.put("msg", out_msg);
            }
        } catch (Throwable e) {
            try {
                conn.rollback();
            } catch (SQLException ex) {
            }
            map.put("id", 0);
            map.put("msg", e.getMessage());
```

```
        } finally {
            disconnect();
        }

        return map;
    }
```

- prodecure : COMMENTS_INS

```
create or replace PROCEDURE COMMENTS_INS
(
   P_CONTENT       IN   VARCHAR2
, P_NAME           IN   VARCHAR2
, P_BOARD_ID       IN   VARCHAR2
, P_ID            OUT  VARCHAR2
, OUT_MSG          OUT  VARCHAR2
)
/*---------------------------------------
프로시져명 : COMMENTS_INS
      설명: 댓글등록
    작성자: 홍길동
작성일자: 2017.xx.xx
---------------------------------------*/
IS
      V_ID NUMBER;

BEGIN

  -- 시퀀스조회
  BEGIN
       -- 시퀀스조회
       SELECT VALUE
         INTO V_ID
         FROM ID_REPOSITORY
        WHERE NAME='COMMENT';

         --시퀀스 증가
         V_ID := V_ID + 1;
        UPDATE ID_REPOSITORY
           SET VALUE = V_ID
         WHERE NAME='COMMENT';
  EXCEPTION
    WHEN NO_DATA_FOUND THEN
           V_ID := 1;
           INSERT INTO ID_REPOSITORY (NAME, VALUE) VALUES ( 'COMMENT', V_ID);
  END;

  -- 댓글 등록
  INSERT INTO COMMENTS (ID, NAME, CONTENT, BOARD_ID)
  VALUES (V_ID, P_NAME, P_CONTENT,P_BOARD_ID);

  P_ID := V_ID;
  OUT_MSG := '처리완료.';

EXCEPTION
  WHEN OTHERS THEN
    P_ID := 0;
    OUT_MSG := TO_CHAR(SQLCODE) || ' : ' || SQLERRM;

END COMMENTS_INS;
```

다. mybatis에서 procedure 호출

(1) mapper파일

```
<parameterMap type="board" id="boardParam">
        <parameter property="title"    mode="IN"  jdbcType="VARCHAR" javaType="string"/>
        <parameter property="writer"   mode="IN"  jdbcType="VARCHAR" javaType="string"/>
        <parameter property="content"  mode="IN"  jdbcType="VARCHAR" javaType="string"/>
        <parameter property="seq"      mode="OUT" jdbcType="NUMERIC" javaType="int"/>
        <parameter property="out_msg"  mode="OUT" jdbcType="VARCHAR" javaType="string"/>
</parameterMap>

<insert id="insertBoardProc1" statementType="CALLABLE" parameterMap="boardParam">
        { call BOARD_INS_PROC(?,?,?,?,?) }
</insert>

<insert id="insertBoardProc2" statementType="CALLABLE" parameterType="board">
        { call BOARD_INS_PROC(
            #{title},
            #{writer},
            #{content, mode=IN, jdbcType=VARCHAR, javaType=string},
            #{seq,     mode=OUT, jdbcType=NUMERIC, javaType=java.math.BigDecimal},
            #{out_msg, mode=OUT, jdbcType=VARCHAR, javaType=string}
          )
        }
</insert>
```

(2) procedure

```
CREATE OR REPLACE PROCEDURE BOARD_INS_PROC
(
  P_TITLE     IN  VARCHAR2
, P_WRITER    IN  VARCHAR2
, P_CONTENT   IN  VARCHAR2
, P_SEQ       OUT NUMBER
, OUT_MSG     OUT VARCHAR2
)
/*----------------------------------------
프로시져명: BOARD_INS
     설명: 게시글등록
   작성자: 홍길동
 작성일자: 2017.xx.xx
----------------------------------------*/
IS
   V_SEQ NUMBER;

BEGIN
      -- 1. 시퀀스조회
      SELECT NVL(MAX(SEQ), 0) + 1
        INTO V_SEQ
        FROM BOARD;

    -- 2. 게시글 등록
    INSERT INTO BOARD(SEQ, TITLE, WRITER, CONTENT, REGDATE)
         VALUES( V_SEQ, P_TITLE, P_WRITER, P_CONTENT, SYSDATE);

    P_SEQ := V_SEQ;
    OUT_MSG := '처리완료.';

EXCEPTION
  WHEN OTHERS THEN
    P_SEQ := 0;
    OUT_MSG := TO_CHAR(SQLCODE) || ' : ' || SQLERRM;

END BOARD_INS_PROC;
```

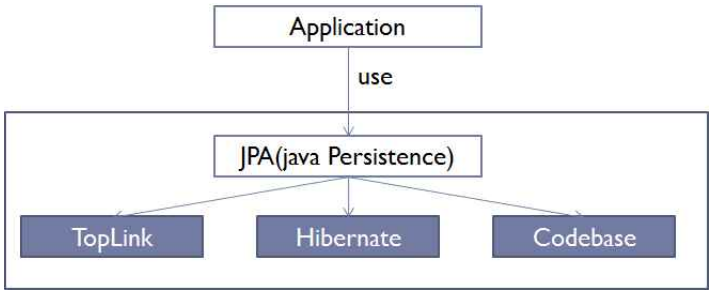
5. JPA

가. 개요

(1) JPA(Java Persistence API)

JPA는 자바 ORM 기술에 대한 API 표준 명세를 의미한다. JPA는 ORM을 사용하기 위한 인터페이스를 모아둔 것이며 JPA를 사용하기 위해서는 JPA를 구현한 Hibernate 및 EclipseLink, DataNucleus와 같은 ORM 프레임워크를 사용해야 한다.

JDBC가 특정 DBMS에 종속되지 않는 DB 연동 구현을 지원하는 것처럼 JPA API를 이용하면 소스 수정 없이 ORM 프레임워크를 교체할 수 있다.



(2) JPA 등장 배경

- 기존 SQL에 의존적이고 중심적인 개발 시 불편(유사한 CURD SQL 반복 작업)
- 쿼리가 변경되면 프로그램 소스 DTO 객체도 변경(DAO와 테이블의 강한 의존성)
- 데이터를 가져와서 객체지향적인 관계를 Mapping하는 일이 빈번(객체를 단순히 데이터 전달 목적으로 사용할 뿐, 객체 지향적이지 못함 (페러다임 불일치)

(3) 장점

- 객체지향적으로 데이터를 관리할 수 있기 때문에 비즈니스 로직에 집중
- 테이블 생성, 변경, 관리가 쉽다. 로직을 쿼리에 집중하기 보다는 객체 자체에 집중할 수 있다.
- query를 직접 작성하지 않고 메서드 호출만으로 query가 수행되다 보니, ORM을 사용하면 생산성이 매우 높다.
- 특정 밴더에 종속적이지 않다.(JPA는 추상화된 접근 계층을 제공하기 때문에 특정 벤더에 종속적이지 않다. DB 변경이 수월)

(4) 단점

- 그러나 query가 복잡해지면 ORM으로 표현하는데 한계가 있고, 성능이 raw query에 비해 느리다.
- 복잡한 통계 분석 쿼리를 메서드만으로 해결하는 것은 힘들다. 이것을 보완하기 위해 SQL과 유사한 기술인 JPQL을 지원한다.

(5) Hibernate

Boss에서 개발한 ORM 프레임워크이며 특정 클래스에 매핑되어야 하는 데이터베이스의 테이블에 대한 관계 정의가 되어 있는 XML 파일의 메타데이터 객체관계 매핑을 간단하게 수행시킨다. Hibernate를 사용하면 데이터베이스가 변경되더라도 SQL 스크립트를 수정하는 등의 작업을 할 필요가 없다. 애플리케이션에서 사용되는 데이터베이스를 변경시키고자 한다면 설정파일의 dialect 프로퍼티를 수정함으로서 쉽게 처리할 수 있다.

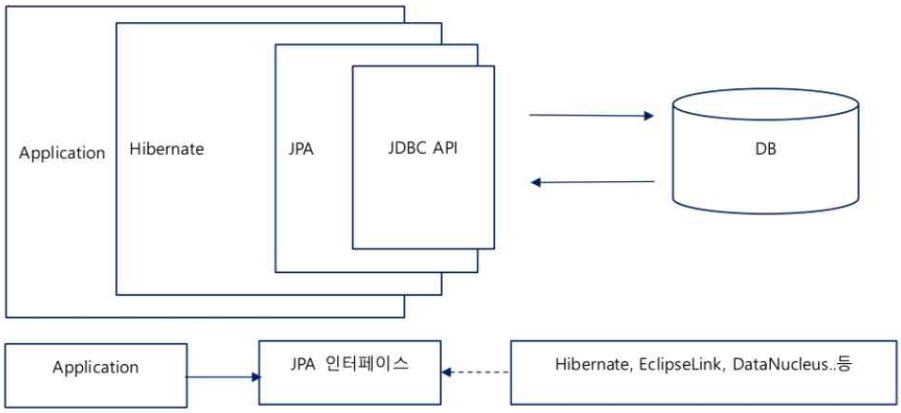


그림 11 JPA, Hibernate architecture

참고사이트: https://www.slideshare.net/visualkhh/hibernate-start

(6) Spring Data JPA

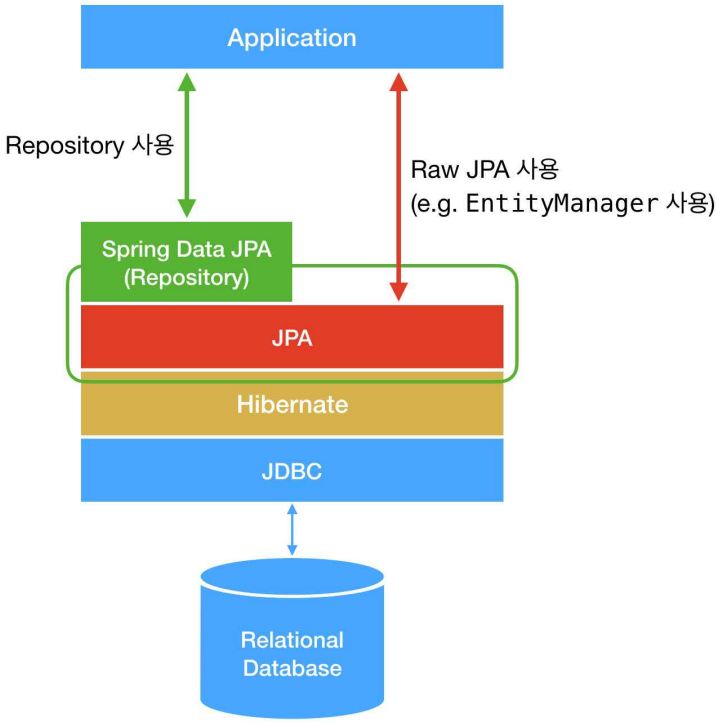
JPA는 기술명세이다. Java Persistence API의 약자로 자바 어플리케이션에서 관계형 데이터베이스를 사용하는 방식을 정의한 인터페이스이다. 자바 어플리케이션에서 관계형 데이터베이스를 어떻게 사용해야 하는지를 정의하는 한 방법일 뿐이다.

Hibernate는 JPA라는 명세의 구현체이다. 즉, 위에서 언급한 javax.persistence.EntityManager와 같은 인터페이스를 직접 구현한 라이브러리이다. JPA와 Hibernate는 마치 자바의 interface와 해당 interface를 구현한 class와 같은 관계이다.

Spring Data JPA는 Spring에서 제공하는 모듈 중 하나로, 개발자가 JPA를 더 쉽고 편하게 사용할 수 있도록 도와준다. 이는 JPA를 한 단계 추상화시킨 Repository라는 인터페이스를 제공함으로써 이루어진다. 사용자가 Repository 인터페이스에 정해진 규칙대로 메소드를 입력하면, Spring이 알아서 해당 메소드 이름에 적합한 쿼리를 날리는 구현체를 만들어서 Bean으로 등록해준다.

Spring Data JPA가 JPA를 추상화했다는 말은, Spring Data JPA의 Repository의 구현에서 JPA를 사용하고 있다는 것이다. 예를 들어, Repository 인터페이스의 기본 구현체인 SimpleJpaRepository의 코드를 보면 아래와 같이 내부적으로 EntityManager을 사용하고 있는 것을 볼 수 있다.

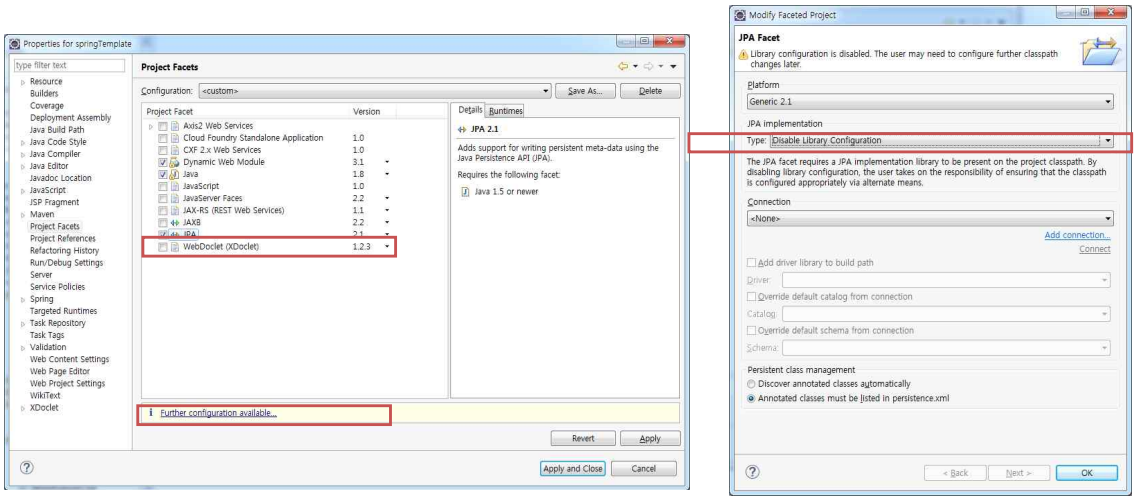
참고사이트: https://suhwan.dev/2019/02/24/jpa-vs-hibernate-vs-spring-data-jpa/



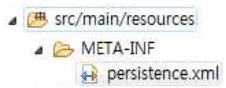
나. Spring과 JPA 연동

(1) JPA 프로젝트로 변환

(가) project facet에서 JPA 추가



(나) persistence.xml 파일 생성 확인



JAP는 persistence.xml 파일을 사용하여 설정 정보를 관리하며 META-INF 폴더에 있으면 별도의 설정 없이 JPA가 인식한다.

(다) 라이브러리 내려받기

```
        <!-- hibernate -->
        <dependency>
            <groupId>org.hibernate</groupId>
            <artifactId>hibernate-entitymanager</artifactId>
            <version>5.2.13.Final</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-orm</artifactId>
            <version>${org.springframework-version}</version>
        </dependency>
```

spring version 4.2 ==> hibernate 5.1

spring version 4.3 ==> hibernate 5.2

(2) 영속성 유닛(Persistence Unit) 설정

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.1"
    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_1.xsd">
    <persistence-unit name="JPAProject">
      <!-- Entity 클래스 등록  -->
      <class>com.company.app.jpa.BoardJPA</class>
      <properties>
       <!-- 필수 속성 -->
       <!-- 2) 하이버네이트 구현체 사용 -->
       <property name="javax.persistence.jdbc.driver" value="oracle.jdbc.driver.OracleDriver" />
       <property name="javax.persistence.jdbc.user" value="spring" />
       <property name="javax.persistence.jdbc.password" value="spring" />
       <property name="javax.persistence.jdbc.url" value="jdbc:oracle:thin:@localhost:1521:xe" />
       <property name="hibernate.dialect" value="org.hibernate.dialect.OracleDialect" />

       <!-- 옵션 -->
       <property name="hibernate.show_sql" value="true" />              <!-- SQL 보기 -->
       <property name="hibernate.format_sql" value="true" />            <!-- SQL 정렬해서 보기 -->
       <property name="hibernate.use_sql_comments" value="false" />     <!-- SQL 주석 보기 -->
       <property name="hibernate.id.new_generator_mappings" value="true" /><!-- JPA 표준에 맞게 새로운 키 생성
전략을 사용 -->
       <property name="hibernate.hbm2ddl.auto" value="create" />        <!-- DDL 자동 생성 -->
      </properties>
    </persistence-unit>
</persistence>
```

(가) 엔티티 클래스 등록

스프링 프레임워크에서는 엔티티 빈에는 @Entity 어노테이션을 지정하고 entityManagerFactory 빈 등록 시 packagesToScan 속성을 이용해 자동 등록되도록 지정할 수 있다.

(나) 속성

속성	설명
hibernate.dialect	사용할 데이터베이스 지정
hibernate.show_sql	생성된 SQL을 콘솔에 출력한다.
hibernate.format_sql	SQL을 출력할 때 일정한 포맷으로 보기 좋게 출력한다.
hibernate.use_sql_comments	SQL에 포함된 주석도 같이 출력한다.
hibernate.id.new_generator_mappings	새로운 키 생성 전략을 사용한다.
hibernate.hbm2ddl.auto	테이블 생성이나 수정 삭제 같은 DDL 구문을 자동으로 처리할 지를 지정한다.

(다) hibernate.dialect 속성

특정 DBMS에 최적화된 SQL을 제공하기 위해서 DBMS마다 다른 Dialect 클래스 지정한다. DBMS가 변경되는 경우 Dialect 클래스만 변경하면 SQL이 자동으로 변경되어 생성되므로 유지보수는 크게 향상된다.

(라) hibernate.hbm2ddl.auto 속성

속성값	설명
create	기존 데이터베이스를 drop 한 후 엔티티 클래스에 설정된 매핑 설정을 참조하여 새로운 테이블을 생성한다.(drop + create-only를 수행한 것과 같다) classpath에 import.sql이 있을 경우 import.sql을 구동해 준다.
create-drop	create기능과 같지만 애플리케이션이 종료되기 직전에 생성된 테이블을 삭제한다.(CREATE-DROP)
create-only	데이터베이스를 새로 생성
drop	데이터베이스를 drop
update	기존에 사용중인 테이블이 있으면 새 테이블을 생성하지 않고 재사용한다. 만약 엔티티 클래스의 매핑 설정이 변경되면 변경된 내용만 반영한다.(ALTER)
validate	테이블 스키마의 Validation기능만 제공
none	기본값이며 아무 일도 일어나지 않는다.

▣ hbm2ddl.auto가 update인 경우 테이블이 없으면 create table 실행하고 있는 경우는 alter table 를 실행

```
@Entity
@Table(name = "BOARD")
public class BoardDTO {
        ...
        private String img;
```

console 실행결과 확인

```
Hibernate:

    alter table BOARD
        add img varchar2(255 char)
```

```
@Entity
@Table(name = "MENU")
public class MenuDTO {
        @Id
        @GeneratedValue
        private Integer menu_no;
        private String menu_name;
        private String program_id;
        //setter/getter
        ...
}
```

console 실행결과 확인

```
Hibernate:

    create table MENU (
        menu_no number(10,0) not null,
        menu_name varchar2(255 char),
        program_id varchar2(255 char),
        primary key (menu_no)
    )
```

▣ hbm2ddl.auto가 create인 경우 drop table을 실행한 후 create table 실행

```
@Entity
@Table(name = "MENU")
public class MenuDTO {
        @Id
        @GeneratedValue
        private Integer menu_no;
        private String menu_name;
        private String program_id;

        //setter/getter
        ...
}
```

console 실행결과 확인

```
Hibernate:

    drop table MENU cascade constraints

Hibernate:

    create table MENU (
        menu_no number(10,0) not null,
        menu_name varchar2(255 char),
        program_id varchar2(255 char),
        primary key (menu_no)
    )
```

(3) Entity 클래스 작성

엔티티의 클래스. 컬럼에 대응한 프로퍼티(필드와 엑세스 메서드)를 가지는 클래스

매핑 어노테이션을 생략하면 필드명을 그대로 칼럼명으로 매핑한다.

어노테이션	설 명
@Entity	@Entity 설정된 클래스를 엔티티 클래스라고 하며, @Entity 가 붙은 클래스는 테이블과 매핑된다.
@Table	엔티티와 관련된 테이블을 매핑한다. name 속성을 사용하여 BOARD 테이블과 매핑했는데 생략하면 클래스 이름이 테이블 이름과 매핑된다.
@Id	엔티티 클래스의 필수 어노테이션으로, 특정 변수를 테이블의 기본 키와 매핑한다. 예제에서는 seq 변수를 테이블의 SEQ 칼럼과 매핑했다. @Id가 없는 엔티티 클래스는 JPA가 처리하지 못한다.
@GeneratedValue	@Id가 선언된 필드에 기본 키를 자동으로 생성하여 할당할 때 사용한다. 다양한 옵션이 있지만 @GeneratedValue만 사용하면 데이터베이스에 따라서 자동으로 결정된다. H2는 시퀀스를 이용하여 처리한다.
@Temporal	날짜 타입의 변수에 선언하여 날짜 타입을 매핑할 때 사용한다. TemporalType의 DATE, TIME, TIMESTAMP 중 하나를 선택할 수 있다.
@Column	엔티티 클래스의 변수와 테이블의 컬럼을 매핑할 때 사용한다. 엔티티 클래스의 변수 이름과 컬럼 이름이 다를 때 사용하며, 생략하면 기본으로 변수 이름과 컬럼 이름을 동일하게 매핑한다.
@Transient	엔티티 클래스의 변수들 중에 테이블의 칼럼과 매핑되는 칼럼이 없거나 매핑에서 제외해야 하는 경우 사용한다. 객체에 임시로 어떤 값을 보관하고 싶을 때 사용할 수 있습니다.
@UniqueConstraint	unique key 생성
@Enumerated	자바의 enum 타입을 매핑할 때 사용

(가) @Entity

어노테이션을 클래스 선언 부분에 부여하여 객체를 테이블과 매핑 할 엔티티라고 JPA에게 알려주는 역할을 한다. (엔티티 매핑). @Entity가 붙은 클래스는 JPA가 관리하게 된다.

@Entity를 선언할 때 몇 가지 주의 사항이 있습니다.

- 기본 생성자는 꼭 존재해야 합니다.
- final class, inner class, enum , interface에는 사용할 수 없습니다.
- 필드에 final 을 사용하면 안됩니다.

(나) @Table

name : 매팽될 테이블 이름을 지정한다. (대소문자 구분없고 기본값은 엔티티의 이름)
catalog : 데이터베이스 카탈로그(catalog)를 지정한다.
schema : 데이터베이스 스키마를 지정한다.
uniqueConstraints : 결합 unique 제약조건을 지정하며, 여러 개의 컬럼이 결합되어 유일성을 보장하는 경우 사용한다.

(다) @Column

name : 컬럼 이름을 지정한다.(생략 시 프로퍼티명과 동일하게 매핑)
unique : unique 제약조건을 추가한다.(기본값 : false)
nullable : null 상태 허용 여부를 설정한다.(기본값 :false)
insertable : 입력 SQL 명령어를 자동으로 생성할 때 이 칼럼을 포함할 것인지를 지정한다.(default: true)
updatable : 수정 SQL 명령어를 자동으로 생성할 때 이 칼럼을 포함할 것인지를 지정한다.(default: true)
columnDefinition : 이 칼럼에 대한 DDL 문을 직접 설정한다.
length: 문자열 타입의 칼럼 길이를 지정한다. String 타입에만 적용(기본값: 255)
precision : 숫자 타입의 전체 자릿수를 지정한다.(기본값: 0)
scale: 숫자 타입의 소수점 자릿수를 지정한다.(기본값 : 0)

(라) @GeneratedValue

strategy : 자동 생성 유형을 지정한다.(GenerationType 지정)

generator : 이미 생성된 Generator 이름을 지정한다.

PK값 자동 생성 전략은 TABLE, SEQUENCE, IDENTITY, AUTO 네 가지가 있다.

1. TABLE: 키 생성 전용 테이블을 만들어서 sequence처럼 사용
2. SEQUENCE: 데이터베이스 시퀀스를 사용해서 기본 키를 할당(Oracle, DB2)
3. IDENTITY: 기본키 생성을 데이터베이스에 위임(MySQL, SQL Server, DB2)
4. AUTO: DB 종류에 따라 JPA가 선택(Oracle은 SEQUENCE, MySQL은 IDENTITY를 선택)

(마) @Temporal

날짜 타입(java.util.Date , java.util.Calendar)을 매핑 할 때 사용합니다.

자바에서는 Camel 표기법(regDate)을 사용하고, DB에서는 snake 표기법(reg_date)을 사용

속성

TemporalType.DATE	: 날짜, 데이터베이스 date 타입과 매핑 (ex) 2013–10–11
TemporalType.TIME	: 시간, 데이터베이스 time 타입과 매핑 (ex) 11:11:11
TemporalType.TIMESTAMP	: 날짜와 시간, 데이터베이스 timestamp(datetime) 타입과 매핑 (ex) 2013–10–11 11:11:11

(바) uniqueConstraints

```
@Entity
@Table(name = "MENU", uniqueConstraints= {@UniqueConstraint(columnNames=
{"menu_no","menu_name"})})
public class MenuDTO {
```

console 실행결과 확인

```
Hibernate:

    alter table MENU
       add constraint UKtb4vs24xojy12ks2ph3l8jkq5 unique (menu_no, menu_name)
```

(4) 트랜잭션 설정 수정

```
<!-- 트랜잭션 AOP -->
<!-- org.springframework.jdbc.datasource.DataSourceTransactionManager -->
<bean id="txManager" class="org.springframework.orm.jpa.JpaTransactionManager">
   <!-- <property name="dataSource" ref="dataSource"/> -->
   <property name="entityManagerFactory" ref="entityManagerFactory"></property>
</bean>
<tx:advice id="txAdvice" transaction-manager="txManager">
   <tx:attributes>
      <tx:method name="get*" read-only="true"/>
      <tx:method name="*"/>
   </tx:attributes>
</tx:advice>
<aop:config>
   <aop:pointcut expression="execution(* com.yedam.app..*Impl.*(..))"
         id="allpointcut"/>
   <aop:advisor advice-ref="txAdvice" pointcut-ref="allpointcut"/>
</aop:config>
```

(5) 스프링과 JPA 연동

```
    <!-- 스프링과 JPA 연동 설정 -->
    <bean id="jpaVendorAdapter"
class="org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter"></bean>

    <bean id="entityManagerFactory"
class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">
        <property name="dataSource" ref="dataSource"></property>
        <property name="jpaVendorAdapter" ref="jpaVendorAdapter"></property>
        <property name="packagesToScan" value="com.company.app"></property>
        <property name="jpaProperties">
          <props>
          <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">false</prop>
          <prop key="hibernate.id.new_generator_mappings">true</prop>
          <prop key="hibernate.hbm2ddl.auto">update</prop>
          </props>
        </property>
    </bean>
```

(6) DAO 클래스 작성

EntityManager를 통해서 엔티티를 취득하거나 갱신하면 내부에서 RDB에 엑세스가 일어난다.

EntityManager 객체가 제공하는 CRUD 기능의 메서드

메서드	설명
persist(Object entity)	엔티티를 영속한다.(INSERT)
merge(Object entity)	준영속 상태의 엔티티를 영속한다.(UPDATE)
remove(Object entity)	영속 상태의 엔티티를 제거한다.(DELETE)
find(Class<T> entityClass, Object promaryKey)	하나의 엔티티를 검색한다.(SELECT ONE)
createQuery(String qString, Class<T> resultClass)	JPQL에 해당하는 엔티티 목록을 검색한다.(SELECT LIST)

다. JPQL

JPQL의 탄생 배경은 JPA에서 제공하는 메서드 호출만으로 섬세한 쿼리 작성이 어렵다는 것에 있습니다. JPQL(Java Persistence Query Language)은 JPA를 구현한 프레임워크에서 사용하는 언어입니다. JPA 구현 프레임워크에서는 JPQL을 SQL로 변환해 데이터베이스에 질의하게 됩니다. JPQL은 테이블을 대상으로 쿼리하지 않고 객체를 고려해 쿼리합니다. 이 때문에 JPQL은 데이터베이스 테이블에 직접적인 의존 관계를 맺고 있지 않습니다. JPQL은 SQL과 비슷한 구조로 구성 됐습니다.

```
select b
  from guestbook
where b.writer = :writer
  order by r.no desc
```

관련사이트:

jpql: https://docs.oracle.com/html/E13946_01/ejb3_langref.html

QueryDSL: http://www.querydsl.com/static/querydsl/3.4.3/reference/ko-KR/html_single/

1. Criteria 쿼리 : JPQL을 편하게 작성하도록 도와주는 API. 빌더 클래스 모음.
2. 네이티브 SQL : JPA에서 JPQL대신 직접 SQL을 사용할 수 있다.
3. QueryDSL : Criteria 쿼리처럼 JPQL을 편하게 작성하도록 도와주는 빌더 클래스 모음. 비표준 오픈소스 프레임워크.
4. JDBC직접이용 : MyBatis같은 SQL 매퍼 프레임워크 사용. 필요하면 JDBC를 직접 사용가능함.