ver 3.1

MyBatis 3

http://www.jobtc.kr
http://www.jhta.co.kr
hipwg@naver.com(박원기)
2018.07

1.mybatis 개요

1.1. MyBatis란

- 모든 MyBatis 애플리케이션은 SqlSessionFactory 인스턴스를 사용한다.
- 개발자가 지정한 SQL, 저장 프로시저 및 고급 매핑을 지원하는 퍼시스턴스[1] 프레임 워크[2] 이다.
- 데이터베이스 레코드에 원시타입과 Map 인터페이스, POJO[3]를 설정하고 매핑하기 위해 XML과 애노테이션을 사용할 수 있다.
- POJO를 통해 비즈니스 처리를 하게하고 SQL문장은 XML 문서로 분리해서 사용한다.
- 2.x버전과 3.x버전과의 호완성은 없다.
- http://mybatis.github.io/mybatis-3/ko/index.html 에서 한글로 직역된(?) 사용자 메뉴얼을 참조할 수 있다.

1.2. 라이브러리 다운로드

https://code.google.com/p/mybatis/ 또는 blog.mybatis.org 에 접속하여 압축된 라이브러리를 다운로드 받아 압축을 해제한다.

- mybatis-X.jar 파일을 작업하고자 하는 프로젝트의 WEB-INF/lib 경로에 복사하는 것으로 설치 작업이 완료된다.(X는 버전번호)

1.3. SqlSessionFactory 빌드하기

SqlSessionFactory는 DataBase을 연결하기 위한 연결 정보와 SQL, Transaction등과 같은 관련된 정보를 갖고 있는 요소이다. 자바빈이나 서블릿에서 JDBC를 연결하기 위한 클래스를 정의한 후 해당 클래스를 생성하여 Connection을 가져오는 역활과 매우 유사하다.

1.4. 환경 설정

SqlSessionFactory를 빌드하기 위한 방법에는 XML로 빌드 하는 방법과 자바 코드로 빌드 하는 방법이 있지만 본 문서에서는 XML을 사용한 빌드 방법만을 기술하겠다.

작성되는 파일은 WEB-INF/classes/ 또는 src/ 안에 존재해야 하며 빌드 정보의 구조는 아래와 같다. <configuration/>안에 <environments/>와 <mappers/> 요소가 존재한다.

<environments/>안에는 데이터베이스 연결 정보와 같은 내용을 기술하고 <mappers/> 안에는 sql 문장들이 기술되어 있는 xml 파일들의 경로와 파일명을 기술한다.

```
<?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">
```

1 Persistent 란 영속적인, 지속적인 이란 의미가 있으며 이는 메모리에 저장되기 보다는 하드 디스크상에 저장된 어떤 환경 및 데이터를 의미한다.
2 프레임 워크 : 정해진 절차와 방법들을 사용하여 일관성있게 프로그램을 개발할 수 있도록 도와주는 툴.
3 POJO : Plane Old Java Object의 약자로 순수 자바로 만들어진 객체를 의미함.

```
<configuration>
   <environments default="development">
         <!--db 연결을 위한 일반적인 정보들 -->
   </environments>

   <mappers>
         <!-- sql 문장들이 들어 있는 mapper 파일들 -->
   </mappers>
</configuration>
```

1.5. environments 설정

1.5.1. 프로퍼티 파일 사용하지 않고 설정

[실예]

WEB-INF/classes/config.xml

```
1.  <?xml version="1.0" encoding="UTF-8" ?>
2.  <!DOCTYPE configuration
3.   PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
4.   "http://mybatis.org/dtd/mybatis-3-config.dtd">
5.
6.  <configuration>
7.   <environments default="development">
8.    <environment id="development">
9.     <transactionManager type="JDBC"/>
10.    <dataSource type="POOLED">
11.     <property name="driver" value="oracle.jdbc.driver.OracleDriver"/>
12.     <property name="url" value="jdbc:oracle:thin:@127.0.0.1:1521:xe"/>
13.     <property name="username" value="hr"/>
14.     <property name="password" value="HR"/>
15.    </dataSource>
16.   </environment>
17.  </environments>
18.
19.  <mappers>
20.   <mapper resource="member.xml"/>
21.   <mapper resource="board/board.xml"/>
22.  </mappers>
23.
24. </configuration>
```

[코드 설명]

8행 : transactionManager의 두가지 타입

- JDBC : 커밋과 롤백을 간단하게 처리한다. 트랜잭션의 스코프를 관리하기 위해 dataSource 로부터 커넥션을 가져온다.
- MANAGED : 커밋이나 롤백을 실행하지 않는다.

9행 : dataSource의 3가지 타입

- UNPOOLED : 매 요청에 대해 커넥션을 열고 닫는다.
- POOLED : DataSource 에 풀링이 적용된 JDBC 커넥션을 사용한다.
- JNDI : 컨테이너 설정에 따라 설정이 변경되며 JNDI 컨텍스트를 참조한다.

19~22god : Mappers

SQL문장들이 기술되어 있는 xml 파일들을 지정한다.

- 20행 : 컨텐츠에 해당하는 sql문장이 저장되어 있는 파일. 경로를 설정하지 않았기 때문에 WEB-INF/classes 폴더 안에 파일이 있어야 함.
- 21행 : 경로가 지정되어 있기 때문에 src/board/안에 파일이 있어야 함.

1.5.2. 프로퍼티 파일 사용하여 설정

config.xml 파일을 작성할 때 Database 연결과 관련한 정보를 별도의 프로퍼티파일을 작성하여 사용해 보자.

step 1. 외부 프로터피 파일을 작성한다.

config.properties

```
driver = oracle.jdbc.driver.OracleDriver
url = jdbc:oracle:thin:@127.0.0.1:1521:xe
username = hr
password = HR
```

step 2. 외부 프로퍼티 파일 정보를 <properties/>를 사용하여 정의한다.

step 3.

프로퍼티로 적용할 부분을 ${프로퍼티명}으로 수정한다.

WEB-INF/classes/config.xml

```
1.  <?xml version="1.0" encoding="UTF-8" ?>
2.  <!DOCTYPE configuration
3.   PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
4.   "http://mybatis.org/dtd/mybatis-3-config.dtd">
5.
6.  <configuration>
7.
8.    <properties resource="config.properties"/>
9.
10.
11.   <environments default="development">
12.    <environment id="development">
13.     <transactionManager type="JDBC"/>
14.     <dataSource type="POOLED">
15.      <property name="driver" value="${driver}"/>
16.      <property name="url" value="${url}"/>
17.      <property name="username" value="${username}"/>
18.      <property name="password" value="${password}"/>
19.     </dataSource>
20.    </environment>
21.   </environments>
22.
23.   <mappers>
24.    <mapper resource="member.xml"></mapper>
25.    <mapper resource="board/board.xml"></mapper>
26.   </mappers>
27.
28. </configuration>
```

1.6. Mapper 경로 지정 방법

mybatis는 자동으로 리소스를 찾는 방법을 제공하지 않는다.

클래스 패스에 상대적으로 리소스를 지정할 수 도 있고 , url을 통해서 지정할 수 도 있다.

기본 클래스 패스 위치 : WEB-INF/classes or /src

```
<mappers>
   <!--WEB-INF/classes안에있는경우-->
   <mapperresource="member.xml"/>
   <!--src/board안에있는경우-->
   <mapperresource="board/board.xml"/>
   <!--특정패키지안에있는경우-->
   <mapperresource="kr/jobtc/myResource/geustbook.xml"/>
   <!--특정디렉토리안에있는경우-->
   <mapperurl="file:///a/b/c/member.xml"/>
</mappers>
```

1.7. SqlSessionFactory 주요 내용

- 모든 MyBatis Application은 SqlSessionFactory 인스턴스를 사용한다.
- 각 인스턴스마다 하나의 SqlSessionFactory만을 사용할 수 있기 때문에 만약 여러 개의 DB를 연결하려고 하는 경우 DB마다 각각의 SqlSessionFactory가 필요할 수 있다.
- 아래의 두가지 방법을 사용하여 SqlSessionFactory를 빌드정보를 정의 한다.

1.7.1. SqlSessionFactory 생성

환경 설정부분에서 생성된 config.xml파일을 읽어들여 SqlSessionFactory를 생성해야 한다. mybatis를 사용하기 위해서는 Database별로 SqlSessionFactory를 생성해야 한다.

SqlSessionFactory의 대부분은 싱글톤 형식으로 사용하기 때문에 static형으로 만들어 가져다 사용한다.

src/begin/BoardFactory.java

```
1.  package begin;
2.
3.  import java.io.Reader;
4.
5.  import org.apache.ibatis.io.Resources;
6.  import org.apache.ibatis.session.SqlSessionFactory;
7.  import org.apache.ibatis.session.SqlSessionFactoryBuilder;
```

```
8.
9.  public class BoardFactory {
10.
11.   // 드라이버를 로딩, Connection정보를 갖고 있음.
12.   private static SqlSessionFactory factory;
13.
14.   static{
15.      try{
16.         Reader reader=Resources.getResourceAsReader("config.xml");
17.         factory = new SqlSessionFactoryBuilder().build(reader);
18.
19.      }catch(Exception ex){
20.         ex.printStackTrace();
21.      }
22.   }
23.
24.   public static SqlSessionFactory getFactory(){
25.      return factory;
26.   }
27.
28. }
```

1.7.2. SqlSessionFactory 생성 테스트

정의된 config.xml 과 BoardFactory에 의해서 정상적으로 SqlSessionFactory가 생성되어 필요한 곳에서 사용될 수 있는지 간단한 jsp 코드를 사용하여 테스트해 보자.

WebContent/begin/factory_test.jsp

```
1.  <%@page import="begin.BeginFactory"%>
2.  <%@page import="org.apache.ibatis.session.SqlSession"%>
3.  <%@ page language="java" contentType="text/html; charset=UTF-8"
4.     pageEncoding="UTF-8"%>
5.  <!DOCTYPE html>
6.  <html>
7.  <head>
8.  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
9.  <title>Insert title here</title>
10. </head>
11. <body>
12. <%
13. // Factory 생성 테스트
14. // properties로 연결된 config 테스트
15.
16. SqlSession sqlSession = null;
17.
18. try{
19.   sqlSession = BeginFactory.getFactory().openSession();
20.
21.   if(sqlSession == null){
22.     out.print("SqlSession 생성중 오류 발생...");
23.   }else{
24.     out.print("SqlSession이 정상적으로 생성되었습니다...");
25.   }
26.   sqlSession.close();
27. }catch(Exception ex){
28.   out.print(ex.toString());
29. }finally{
```

```
30.
31. }
32. %>
33. </body>
34. </html>
```

2.SqlMap

SqlMapper(<mappers/>)에 의해서 지정된 클래스 패스에 Sql 문장을 xml 유형으로 작성한다. 이는 아래와 같은 특징을 갖고 있다.

- SQL 을 작성하는데 집중하도록 만들어졌다.
- JDBC 코드에 비해 최대 95% 이상 코드수가 감소하기도 한다.
- SQL 문장중 ${var} 또는 #{var}를 기술하는데 이는 PrepareStatement 문장의 역활을 하게 한다.
  - ${var} ☾ 양옆에 작은 따옴표를 붙여주지 않는다.(잠재적인 SQL 주입 공격에 노출될 수 있음)
  - #{var} ☾ 양옆에 작은 따옴표를 붙여준다.
  - 예) id값이 park 인 경우
    - where id=${id} ☾ where id=park
    - where id=#{id} ☾ where id='park'
- insert, update, delete시 mybatis의 세션을 commit하고 닫아 주어야 한다.
- parameterType을 자바의 기본형을 사용할 때는 임의의 변수값으로 매핑되지만, 동적 SQL 문장등에서는 오류가 발생한다. 따라서 매핑될 변수명은 '_parameter'를 사용하는 것이 안전하다.

[전체구조]

```
<?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="board">
  <resultMap type="class full name" id="id">
    <result property='property' column='column'/>
    ...
  </resultMap>


  <select id='id' parameterType="type" resultType='type|id'>
    select 문장
  </select>

  <update id="id" parameterType="type">
    update 문장
  </update>

  <insert id='type' parameterType="type">
    insert 문장
  </insert>

  <delete id='id' parameterType="type">
    delete 문장
  </delete>
</mapper>
```

2.1. mapper

sql문장들이 기술될 블럭을 지정한다.

namespace : 다른 mapper들과 구분될 수 있도록 논리적 영역을 구분하기 위해 사용된다. 자바코드에서 호출시 마치 객체명이나 패키지명처럼 "."을 사용하여 구분한다.

<mapper namespace='board'> <select id='list' /> </mapper>	sqlSession.selectList("board.list")

2.2. resultMap

mybatis의 select절에 의해서 반환될 데이터를 구조화 시킬 수 있다.

type : 자바 클래스명을 패키지명과 함께 지정한다.

id : 고유한 이름을 지정한다. <select/>와 같은 곳에서 resultType의 type의 속성값으로 사용된다.

<resultMap type='board.BoardVo' id='board_map'> ... </resultMap>	<select id='list' resultType = 'board_map' />

2.3. select

sql 문장중 select 절을 사용하여 특정한 값을 가져올 때 사용된다. 하나의 <select/>엘리먼트로 정의되지만 실제로 자바 코드에서 호출될 때는 반환되는 데이터의 건수에 따라 selectList, selectOne과 같이 서로 다른 메서드를 사용하여 호출된다.

- 기본구조

```
<select id='id' resultType='type' parameterType='type' />
```

2.3.1. resultType

select 된 결과를 어떤 유형으로 반환할 지를 지정한다.

- 기본형
  - 매핑되는 변수명엔 임의의 변수명을 사용해도 되지만 '_parameter'를 사용하는 편이 안전하다.
- 객체형
- resultMap형

어떤 유형의 resultType을 사용하더라도 selectList()에 의해서 호출되면 반환값을 List구조로 반환된다.

- resultType이 기본형인 경우

<select id='list' resultType='String'> select 절 </select>	두건 이상인 경우 List<String> list = (ArrayList)sqlSession.selectList("board.list"); 한건인 경우 String str = (BoardVo)sqlSession.selectOne("board.list");

- resultType이 객체형인 경우

<select id='list' resultType='board.BoardVo'> select 절 </select>	두건 이상인 경우 List<Boardvo> list = (ArrayList)sqlSession.selectList("board.list"); 한건인 경우 BoadVo vo = (BoardVo)sqlSession.selectOne("board.list");

- resultType이 resultMap인 경우

<select id='list' resultType='board_map'> select 절 </select>	두건 이상인 경우 List<BoardVo> list = (ArrayList)sqlSession.selectList("board.list"); 한건인 경우 BoardVo vo = (BoardVo)sqlSession.selectOne("board.list");

- resultType이 boolean형인 경우

리턴값이 boolean형인 경우는 select문의 최종 결과를 0 또는 1로 반환해야 한다. 작성자와 암호를 비교하여 board테이블에서 해당 자료가 존재하는지 아닌지를 판단하는 간단한 코드를 살펴보자.

```
<select id='login' resultType='boolean' parameterType='bean.BoardVo'>
  select case when count(*) > 0
    then 1
    else 0
    end result
  from board where worker=#{worker} and pwd = #{pwd}
</select>
```

```
boolean b = (boolean)sqlSession.selectOne("board.login", vo);
```

- #{mid}와 #{pwd}는 baen.BoardVo.getWorker(), bean.BoardVo.getPwd()를 각각 의미한다.

- board.login 에서 board는 <mapper namespace='board'/>에서와 같이 namespace값이다.(이하 공통)

2.3.2. parameterType

<select/> 실행시 외부에서 전달되는 데이터 타입을 의미한다. 기본형, 객체형을 가질 수 있다. 사용자 정의 객체인 경우는 패키지명과 함께 클래스명을 기술하면 된다.

- 기본형인 경우

```
<select id='getPwd' resultType='String' parameterType='String'>
   select pwd from member where mid=#{mid}
</select>
```

```
String pwd = sqlSession.selectOne("board.getPwd", "hong");
```

#{mid}는 임의의 값으로 지정할 수 있다.

- 객체형인 경우

```
<select id='getPwd' resultType='String' parameterType='board.BoardVo'>
   select subject from member where worker=#{worker}  and pwd = #{pwd}
</select>
```

```
BoardVo vo = ...
String pwd = sqlSession.selectOne("board.getPwd", vo);
```

#{worker}, #{pwd} 는 BoardVo의 getWorker(), getPwd()를 호출한다.

2.4. insert

데이터를 Database에 입력할 때 사용되는 요소이다. parameterType은 대부분 자바 VO객체형이다. 반환값은 insert된 결과행값을 갖는다. 또한 sqlSession을 다 사용한뒤 반드시 commit해 주어야 한다.

- 기본구조

```
<insert id='id' parameterType='type' />
```

- 간단예

```
<insert id='insert' parameterType='board.BoardVo'>
```

```
    insert into board( worker, subject) values(#{worker}, #{subject} )
</insert>
```

```
BoardVo vo = ...
int r = sqlSession.insert("board.insert", vo);
if(r>0){
    sqlSession.commit();
}
sqlSession.close();
```

- #{worker}, #{subject}은 board.BoardVo객체의 getWorker(), getSubject()이 된다.
- sqlSession.insert()의 반환값은 insert된 행의 갯수이다.

2.5. update

데이터를 수정하기 위한 요소이다.

- 기본구조

```
<update id='id' parameterType='type' />
```

- 간단예

```
<update id='update' parameterType='board.BoardVo'>
    update board set subject = #{subject} where worker = #{worker}
</update>
```

```
BoardVo vo = ...
int r = sqlSession.update("board.update", vo);
if(r>0){
    sqlSession.commit();
}
sqlSession.close();
```

- #{subject}, #{worker}는 board.BoardVo에 있는 getSubject(), getWorker()를 의미한다.
- sqlSession.update()의 반환값은 update된 행의 갯수이다.

2.6. delete

데이터를 삭제하기 위한 요소이다.

- 기본구조

```
<delete id='id' parameterType='type' />
```

- 간단예

```
<delete id='delete' parameterType='int'>
    delete from board where serial = ${serial}
</delete>
```

```
int serial = 100;
int r = sqlSession.delete("board.delete", serial);
if(r>0){
    sqlSession.commit();
}
sqlSession.close();
```

- ${serial}은 어떤 변수명을 사용해도 상관없다.
- sqlSession.delete()의 반환값은 삭제된 행수이다.

3.동적SQL

sqlMap에서 상황에 따른 보다 동적인 sql문장을 만들기 위해 사용되는 명령어들이다.

대표적인 종류

- if
- choose
- where
- set
- trim
- foreach

3.1. if

sql문장 작성시 조건을 명시할 수 있다.

■ 기본구조

```
<if test="조건">
  조건이 참인경우
</if>
```

- 사용예

날짜 항목이 입력되었을 때만 검색조건에 추가 되도록 sql을 만들어 보자.

```
<select id='if' parameterType='board.BoardVo'   resultMap = 'String'>
      select * from board
      <if test='month != null'>
          where to_char(mdate, 'mm') = #{month}
      </if>
</select>
```

3.2. choose

switch 문이나 다중 if문과 그 기능이 유사하다.

■ 기본구조

```
<choose>
   <when test='조건1'>
       실행문1
   </when>
   <when test='조건2'>
       실행문2
   </when>
    ...
```

```
    <otherwise>
        만족하는 조건이 없는 경우
    </otherwise>

</choose>
```

■ 사용예

날짜 또는 작성자가 입력된 경우만 where 조건을 만들어 보자.

```
<select id='choose' parameterType='board.BoardVo' resultMap = 'boardMap'>
   select * from board
   <choose>
      <when test="month != null">
         where to_char(mdate, 'mm') = #{month}
      </when>
      <when test="worker != null">
         where worker = #{worker}
      </when>
   </choose>
</select>
```

3.3. where

where 절을 보다 동적으로 만들어 주는 요소이다.

[특징]

- <where/>안에서 만족하는 조건이 없다면 where절을 생략해 준다.
- 만약 <where/>안에서 sql 문장이 and나 or로시작되면 and나 or를 제거해 준다.

■ 기존 구조

```
<where>
    내용
</where>
```

■ 사용예

```
<select id='where' parameterType="bean.DynamicVo" resultMap='dynamicMap'>
   select * from board
   <where>
      <if test="worker != null"> <!-- bean.DynamicVo.getWorker()에 값이 있다면 -->
         worker = #{worker}
      </if>

      <if test="month != null">
         and to_char(mdate, 'mm') = #{month}
      </if>
   </where>
</select>
```

- worker에 값이 있다면 자동으로

where worker=#{worder}

라는 문장이 추가된다. 그러나 만약 worker에 값이 없고 month에 값이 있다면 and는 사라지고

where to_char(mdate, 'mm') = #{month}

가 추가된다. 그러나 worker와 month에 모두 값이 있다면

where worker=#{worker} and to_char(mdate, 'mm') = #{month}

가 추가된다.

3.4. set

update절의 set절을 동적으로 만들어 주는 요소이다. 용법은 <where/>와 동일하다.

[특징]

- <set/>안에 요소가 존재할 경우만 set절을 만들어 준다.
- 불필요한 ','를 삭제해 준다.

■ 기본구조

```
<set>
...
</set>
```

■ 사용예

```
<update id='set' parameterType="bean.DynamicVo">
   update member
   <set>
      <choose>
         <when test="pwd != null">
            pwd = #{pwd},
         </when>
         <when test="pwd = null or pwd = ''">
            pwd = 'manager',
         </when>
      </choose>
   </set>

   where mid = #{worker}
</update>
```

- pwd(암호)가 입력되면 입력된 값으로 현재 유저(worker)의 암호를 변경해 주고 암호가 입력되지 않으면 기본값으로 'manager'로 암호로 설정해 준다.

pwd = #{pwd},

와 같이 불필요한 ','는 자동으로 삭제된다.

3.5. trim

<where/>나 <set/> 요소보다 다양하게 특정요소를 추가하거나 제거할 수 있는 요소이다.

■ 기본구조

```
<trim prefix = '앞에 붙일단어' suffixOverrides='제거단어| 제거단어' prefixOverrides='제거단어 | 제거단어'>
  ...
</trim>
```

- prefix : where, set과 같이 기술된 내용을 추가한다.
- suffixOverrides : 뒤에 붙은 ',' 와 같이 불필요한 내용을 제거한다.
- prefixOverrides : 앞에 붙은 or, and 와 같은 불필요한 내용을 제거한다.

■ 사용예(select)

```
<select id='trim_select' parameterType='bean.DynamicVo' resultMap = 'dynamicMap'>
   select * from board
   <trim prefix="where" prefixOverrides="or | and">
      <if test="worker != null">
         or worker like '%${worker}%'
      </if>
   </trim>
</select>
```

- <if/> 조건이 맞으면 자동으로 prefix='where'에서 지정된 where가 추가되고 <if/>안에 있는 or는 불필요하기 때문에 삭제되어 아래와 같은 sql이 동적으로 만들어 진다.

select * from board where worker like '%${worker}%'

■ 사용예(update)

```
<update id='trim_update' parameterType='bean.DynamicVo' >
   <if test="worker != null">
      update board
      <trim prefix="set" suffixOverrides=",">
         mdate = sysdate,
      </trim>
      where worker = #{worker}
   </if>
</update>
```

- worker에 값이 존재하면

update board set mdate=sysdate where worker=#{worker}

문장이 만들어 진다.

3.6. foreach

배열이나 Collection구조의 데이터가 parameterType으로 전달되었을 때 이를 순회하여 항목을 하나씩 가져올 수 있는 기능이다. mytatis가 보여주는 가장 강력한 기능중 하나이다.

■ 기본구조

```
<foreach collection='collection' item='item' index='index'  open='open' close='close' separator='separator' />
```

- collection : 자바의 List나 Map 또는 배열 구조의 요소
- item : collection에 정의된 요소를 하나씩 대입 받는 변수. map구조인 경우 value값.
- index : 대입 받은 색인값. map구조인 경우 key값.
- open : 반복처리를 시작하기전 추가될 처음 문자열.
- close : 반복처리를 마치고 추가될 마지막 문자열.
- separator : 반복처리시 item 사이에 끼워질 구분 문자열.

■ 사용예(배열 또는 List 구조인 경우)

```
<select id='foreach_list' parameterType='bean.DynamicVo' resultMap = 'dynamicMap'>

   select * from board

   <where>
      <if test='worker != null'>
         worker in
         <foreach collection='workerList' item='data' index='index'
                  open="(" close=")" separator=",">
            #{data}
         </foreach>
      </if>
   </where>
</select>
```

Java Code

```
SqlSession s = BoardFactory.getFactory().openSession();
DynamicVo vo = new DynamicVo();
List<String> workerList = new ArrayList<String>();
workerList.add("manager");
workerList.add("guest");

vo.setWorkerList(workerList);

List<DynamicVo> v = s.selectList("dynamic.foreach_list", vo);
```

프로그램이 실행되면

select * from board where worker in ('manager', 'guest')

가 실행된다.

Map 구조인 경우엔 전달된 값 자체가 Map 형식일 때와 Map형식의 구조를 하나의 필드를 갖는 객체를 전달 받았을 때로 나누어 생각해 보자.

- 사용예(파라메터 자체가 Map 구조인 경우)

```
<select id='foreach_map' parameterType='Map' resultMap='dynamicMap'>
      select * from board
   <where>
      <if test='worker != null'>
         worker = #{worker}
      </if>
      <if test='month != null'>
         and to_char(mdate, 'mm') = #{month}
      </if>
   </where>
</select>
```

#{worker}, #{month}가 Map의 Key값이되고 해당 Value값을 가져온다.

```
Map<String, Object> map = new HashMap<String, Object>();
map.put("month", “01”);
map.put("worker", “hong”);

List<DynamicVo> v = s.selectList("dynamic.foreach_map", map);
```

만들어진 sql :

select * from board where worker=’hong’ and to_char(mdate, ‘mm’) = ‘01’

- 사용예(Map 구조가 다른 Object의 필드가 될때

```
<select id='foreach_map2' parameterType='Map' resultMap='dynamicMap'>
   select * from board
   <where>
      <if test='worker != null'>
         worker = #{worker}
      </if>
      <if test='list != null'>
          and
         <foreach collection='list' index='key' item='value'
               separator="or" open="("  close=")">
             to_char(mdate, 'mm') = #{value}
         </foreach>
      </if>
   </where>

</select>
```

- #{worker}, #{list}는 Map 구조가 갖고 있는 key값이다.

- <foreach/> 문에서 collection=’list’ 는 모든 키값에 해당하는 값을 하나씩 item=’value’에 담아 반복문을 순행하게 된다.

```
Map<String, Object> map = new HashMap<String, Object>();
List<String> list = new ArrayList<String>();
list.add(“01”);
list.add(“02”);
list.add(“03”);
map.put(“worker”, “hong”);
map.put(“list”, list);

List<DynamicVo> v = s.selectList("dynamic.foreach_map2", map);
```

- mybatis 에 의한 sql은 아래와 같이 만들어질 것이다.

select * from board worker=’hong’ and (to_char(mdata, ‘mm’) = 01 or to_char(mdate, ‘mm’)=02
or to_char(mdate, ‘mm’) = 03

- Map 구조 사용시 key / value 값을 사용한 처리

```
<foreach collection="attFile" index="key" item="value">
  insert into boardattservlet(serial, pserial,
            attfile, oriattfile)
  values(seq_boardattservlet.nextval, seq_boardservlet.currval,
            #{key}, #{value})
</foreach>
```

위의 예는 첨부파일 작업시 실제로 저장되는 파일명은 attfile 로 Map의 키값으로 사용하였고, 원본 파일의 이름은 oriattfile 로 value값으로 저장되었다고 가정하였다.

```
String attfile = “저장된 파일명”;
String oriattfile = “실제 파일명”;
Map<String, String> attfile = new HashMap<String, String>();
attfile.put(attfile, oriattfile);
```

4. TIP

4.1. insert all 에서 시퀀스 증가시키기

일반적으로 insert all과 같은 쿼리를 사용하여 여러 건의 데이터를 한번에 입력시키려 할 때 오라클의 시퀀스는 증가 되지 않는다. 이 때 가장 편한 방법은 시퀀스를 증가시켜 값을 반환해 주는 함수를 만들어 사용하는 것이다.

step 1. 시퀀스 만들기

my_table에 사용할 시퀀스를 seq_my_table 명으로 생성하자.

```
create sequence seq_my_table;
```

step 2. 함수 만들기

오라클 PL-SQL을 사용하여 시퀀스를 증가시켜 반환하는 함수 nextSerial() 을 생성하자.

```
CREATE OR REPLACE function nextSerial
     return number
is
    ns number;
begin
    select seq_my_table.nextval into ns from dual;
    return ns;
end;
```

step 3. nextSerial() 사용하기

3개의 컬럼을 갖는 my_table에 저장한다고 가정하자.

```
insert all
    into my_table values(
        nextSerial() , 'a.png', 'b.png')

    into board2att values(
        nextSerial() , 'c.png', 'd.png')

select *  from dual;
```

4.2. delete cascade 를 사용한 두 데이블 데이터 동시 삭제

정규화 결과로 하나의 테이블이 두개 이상으로 나뉘어져 있을 때 메인 테이블의 데이터가 삭제될 때 FK로 연결된 서브 테이블의 데이터를 동시에 삭제하기 위해서 FK를 생성할 때

on delete cascade

옵션을 추가하여 FK를 생성한뒤 메인 테이블의 데이터를 삭제하면 서브 테이블의 데이터를 자동(?)

으로 삭제 시킬 수 있다.

step 1. cascade 옵션 추가

```
alter table sub add constraint sub_fk
foreign key(pSerial) references main(serial)
on delete cascade;
```

PK가 있는 주 테이블명을 main , FK를 설정하려는 테이블명을 sub로 하고 키값은 serial 과 pSerial 이라고 가정했다.

물론 이 때 main에 PK인 serial에 연결된 sub의 FK(pSerial) 사이에는 데이터의 무결정 제약이 없어야 한다. 즉, pSerial 값은 있는데 serial 값이 없는 경우는 FK가 생성되지 않는다.)

step 2. 데이터 삭제

```
delete from main where serial=100;
```

main 테이블의 serial=100인 데이터와 sub 테이블의 pSerial=100인 데이터들이 한꺼번에 삭제되는 것을 볼 수 있다.