



JSP 2.3 & Servlet 3.1

MYBATIS

- 김근형 -

ORM(Object-Relational Mapping)

▶ ORM 이란?

- ▶ ORM은 데이터베이스와 객체 지향 프로그래밍 언어 간의 호환되지 않는 데이터를 변환하는 프로그래밍 기법
- ▶ 객체 관계 매핑이라고도 한다.
- ▶ 단순히 표현하면 객체와 관계와의 설정이라 할 수 있다.
- ▶ ORM의 개념이 나온 이유는 코딩의 반복적인 부분을 줄일 수 있고 SQL의 의존적인 코딩에서 벗어나 생산적인 코딩이 가능하며 유지보수가 편리하기 때문이다.

ORM(Object-Relational Mapping)

▶ ORM 프레임워크의 장점

- ▶ 객체 지향적인 코드로 인해 더 직관적이고 비즈니스 로직에 집중할 수 있게 도와준다.
 - ▶ 선언문, 할당, 종료 같은 부수적인 코드가 없거나 급격히 줄어든다.
 - ▶ 각종 객체에 대한 코드를 별도로 작성하기 때문에 코드의 가독성을 올려준다.
 - ▶ SQL의 절차적이고 순차적인 접근이 아닌 객체지향적인 접근으로 인해 생산성이 증가한다.
- ▶ 재사용 및 유지보수의 편리성이 증가한다.
 - ▶ ORM은 독립적으로 작성 되어있고, 해당 객체들을 재활용 할 수 있다.
 - ▶ 매핑정보가 명확하여 ERD를 보는 것에 대한 의존도를 낮출 수 있다.
- ▶ DBMS에 대한 종속성이 줄어든다.
 - ▶ DBMS를 교체하는 거대한 작업에도 비교적 적은 리스크와 시간이 소요된다.

ORM(Object-Relational Mapping)

- ▶ ORM 프레임워크의 주의할 점
 - ▶ 설계를 매우 신중히 해야한다.
 - ▶ 프로젝트의 복잡성이 커질 경우 난이도 또한 올라갈 수 있다.
 - ▶ 잘못 구현한 경우에 속도 저하 및 심각할 경우 일관성이 무너진다.
 - ▶ 일부 자주 사용되는 대량 쿼리는 속도를 위해 SP를 쓰는 등 별도의 튜닝이 필요하다.
 - ▶ 프로시저가 많은 시스템에서는 ORM의 객체 지향적 장점을 활용하기 힘들다.

MyBatis

▶ MyBatis

- ▶ MyBatis 란 객체지향 언어인 Java 와 SQL Based 인 관계형 데이터베이스(RDBMS) 사이의 데이터를 다루는 방식의 괴리를 해결하기 위해 만들어진 Persistence Framework 의 일종이다.
- ▶ Java 에서 DB와의 Connection 을 위해 제공하는 JDBC 를 Wrapping 한 구조로 되어 있으며, 기존의 JDBC 에 비해 많은 장점들을 갖고 있다.
- ▶ 무엇보다 사용이 간단하고, 60% 정도의 생산성 향상이 있다고 한다.
- ▶ 또한 JDBC 사용시 매번 Query Statement 를 생성하는 구문을 작성해야 했던 것과 다르게 쿼리의 재사용과, 코드와의 분리가 좀 더 수월 해졌기 때문에 유지 보수 측면에서도 이점을 갖는다.

Mybatis

▶ 스코프와 생명주기

▶ SqlSessionFactoryBuilder

- ▶ 이 클래스는 인스턴스화되어 사용되고 던져질 수 있다.
- ▶ SqlSessionFactory 를 생성한 후 유지할 필요는 없다.
- ▶ 그러므로 SqlSessionFactoryBuilder 인스턴스의 가장 좋은 스코프는 메소드 스코프(예를들면 메소드 지역변수)이다.
- ▶ 여러개의 SqlSessionFactory 인스턴스를 빌드하기 위해 SqlSessionFactoryBuilder를 재사용할 수도 있지만 유지하지 않는 것이 가장 좋다.

Mybatis

▶ 스코프와 생명주기

▶ SqlSessionFactory

- ▶ 한번 만든 뒤 SqlSessionFactory는 애플리케이션을 실행하는 동안 존재해야만 한다.
- ▶ 그래서 삭제하거나 재 생성할 필요가 없다.
- ▶ 애플리케이션이 실행되는 동안 여러 차례 SqlSessionFactory 를 다시 빌드하지 않는 것이 가장 좋은 형태이다.
- ▶ 그러므로 SqlSessionFactory 의 가장 좋은 스코프는 애플리케이션 스코프이다.
- ▶ 애플리케이션 스코프로 유지하기 위한 다양한 방법이 존재한다. 가장 간단한 방법은 싱글톤 패턴이나 static 싱글톤 패턴을 사용하는 것이다.

Mybatis

▶ 스코프와 생명주기

▶ SqlSession

- ▶ 각각의 스레드는 자체적으로 SqlSession 인스턴스를 가져야 한다.
- ▶ SqlSession 인스턴스는 공유되지 않고 스레드에 안전하지도 않다.
- ▶ 그러므로 가장 좋은 스코프는 요청 또는 메소드 스코프이다.
- ▶ SqlSession 을 static 필드나 클래스의 인스턴스 필드로 지정해서는 안된다. 그리고 서블릿 프레임워크의 HttpSession 과 같은 관리 스코프에 뒀어도 안된다.
- ▶ 어떠한 종류의 웹 프레임워크를 사용한다면 HTTP 요청과 유사한 스코프에 두는 것으로 고려해야 한다. 달리 말해서 HTTP 요청을 받을때마다 만들고 응답을 리턴할때마다 SqlSession 을 닫을 수 있다.
- ▶ SqlSession 을 닫는 것은 중요하다. 언제나 finally 블록에서 닫아야만 한다

Mybatis

▶ 스코프와 생명주기

▶ Mapper 인스턴스

- ▶ Mapper는 매핑된 구문을 바인딩 하기 위해 만들어야 할 인터페이스이다.
- ▶ mapper 인터페이스의 인스턴스는 SqlSession 에서 생성한다.
- ▶ 그래서 mapper 인스턴스의 가장 좋은 스코프는 SqlSession 과 동일하다.
- ▶ mapper 인스턴스의 가장 좋은 스코프는 메소드 스코프이다.
- ▶ 사용할 메소드가 호출되면 생성되고 끝난다. 명시적으로 닫을 필요는 없다.

MyBatis

▶ MyBatis

▶ <http://www.mybatis.org/mybatis-3/ko/index.html>

mybatis



Last Published: 07 4월 2019 | Version: 3.5.1

CORE

소개

시작하기

매퍼 설정

Mapper XML 파일

동적 SQL

자바 API

SQL Builder 클래스

로깅

프로젝트 문서화

프로젝트 정보

프로젝트 보고서



소개

마이바티스는 무엇인가?

마이바티스는 개발자가 지정한 SQL, 저장프로시저 그리고 몇가지 고급 매핑을 지원하는 퍼시스턴스 프레임워크이다. 마이바티스는 JDBC로 처리하는 상당부분의 코드와 파라미터 설정 및 결과 매핑을 대신해준다. 마이바티스는 데이터베이스 레코드에 원시타입과 Map 인터페이스 그리고 자바 POJO를 설정해서 매핑하기 위해 XML과 애노테이션을 사용할 수 있다.

이 문서가 더 좋아지도록 도와주세요...

이 문서가 일부 내용을 자세히 다루지 않거나 특정 기능에 대해 설명하지 않을 경우 마이바티스를 공부하는 가장 좋은 방법은 사용자 스스로 그 부분에 대한 문서를 작성하는 것이다

이 문서는 xdoc포맷으로 되어 있으며 [프로젝트 깃허브](#)에서 볼 수 있다. 저장소를 포크하고 수정한 뒤 Pull request요청을 보내면 됩니다.

그러면 이 문서의 가장 멋진 저자가 되고 다른 많은 사람들이 이 문서를 읽을 것이다

번역에 대하여

번역자 : 이동국(fromm0@gmail.com, <http://hdg.pe.kr>, <https://www.facebook.com/dongguk.lee.3>)

다음의 몇가지 언어로 번역된 마이바티스 문서를 읽을 수 있다.

영어

스페인어

일본어

한국어

중국어번체

당신의 모국어로 작성된 마이바티스 문서를 읽고 싶으신가요? 그렇다면 모국어로 작성된 문서를 패치로 첨부한 이슈를 등록해달라

MyBatis

▶ mybatis.jar 다운


- ▶ <https://github.com/mybatis/mybatis-3/releases>

Latest release

mybatis-3.5.1

0f8dc7f

mybatis-3.5.1

 harawata released this 8 days ago · 5 commits to master since this release

Bug fixes:

- `keyProperty` specified with parameter name could cause `ExecutorException`. #1485
- False positive error 'Ambiguous collection type ...'. #1472
- `EnumTypeHandler` is not used when the enum has methods. #1489
- Auto-mapping fails in a result map referenced from a constructor arg with `columnPrefix`. #1496
- Constructor auto-mapping could fail when `columnPrefix` is specified in the parent result map. #1495
- `LocalTimeTypeHandler` loses fractional seconds part. #1478
- `LocalDateTypeHandler` and `LocalDateTimeTypeHandler` could return unexpected value. #1478

Enhancements:

MyBatis

▶ 프로젝트 세팅

- ▼ Chapter17
 - > Deployment Descriptor: Chapter17
 - > JAX-WS Web Services
 - ▼ Java Resources
 - ▼ src
 - ▼ com.mvc.conf
 - ▼ SqlSessionManager.java
 - > SqlSessionManager
 - configuration.xml
 - ▼ com.mvc.dao
 - member.xml
 - > Libraries
 - > JavaScript Resources
 - > build
 - ▼ WebContent
 - > META-INF
 - > WEB-INF
 - mybatisExample1.jsp

MyBatis

- ▶ 프로젝트 세팅
 - ▶ configuration.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="false" />
    <setting name="useGeneratedKeys" value="true" />
    <setting name="defaultExecutorType" value="REUSE" />
  </settings>

  <environments default="development">
    <environment id="development">
      <transactionManager type="JDBC" />
      <dataSource type="POOLED">
        <property name="driver" value="org.mariadb.jdbc.Driver" />
        <property name="url" value="jdbc:mariadb://localhost:3306/java" />
        <property name="username" value="root" />
        <property name="password" value="1234" />
      </dataSource>
    </environment>
  </environments>

  <mappers>
    <mapper resource="com/mvc/dao/member.xml" />
  </mappers>
</configuration>
```

MyBatis

▶ 프로젝트 세팅

▶ member.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="member">
  <select id="getCount" resultType="int">
    SELECT
      count(*)
    FROM
      member
  </select>
</mapper>
```

MyBatis

- ▶ 프로젝트 세팅
 - ▶ SqlSessionManager.java

```
public class SqlSessionManager {  
    public static SqlSessionFactory sqlSession;  
  
    static {  
        String resource = "com/mvc/conf/configuration.xml";  
        Reader reader;  
  
        try {  
            reader = Resources.getResourceAsReader( resource );  
            sqlSession = new SqlSessionFactoryBuilder().build( reader );  
        } catch (IOException e) {  
            e.printStackTrace();  
        }  
    }  
  
    public static SqlSessionFactory getSqlSession() {  
        return sqlSession;  
    }  
}
```


MyBatis

- ▶ 프로젝트 세팅
 - ▶ mybatisExample1.jsp

```
<%@page import="com.mvc.conf.SqlSessionManager"%>
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@page import="org.apache.ibatis.session.SqlSessionFactory"%>
<%@page import="org.apache.ibatis.session.SqlSession"%>
<%@page import="java.util.*"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
<%
    SqlSessionFactory sqlSessionFactory = SqlSessionManager.getSqlSession();
    SqlSession sqlSession = sqlSessionFactory.openSession();

    int cnt = 0;

    try{
        // 단일 값 가져오기
        cnt = sqlSession.selectOne("member.getCount") ;
    }catch(Exception e){
        e.printStackTrace() ;
    }
%>

<p>단일 값 : <%=cnt %></p>
</body>
</html>
```

MyBatis – config

▶ Settings

- ▶ 런타임시 마이바티스의 행위를 조정하기 위한 중요한 값들
- ▶ <http://www.mybatis.org/mybatis-3/ko/configuration.html>

```
<settings>  
  <setting name="cacheEnabled" value="false" />  
  <setting name="useGeneratedKeys" value="true" />  
  <setting name="defaultExecutorType" value="REUSE" />  
</settings>
```

MyBatis - config

▶ typeAlias

- ▶ 타입 별칭은 자바 타입에 대한 짧은 이름이다. 오직 XML 설정에서만 사용되며, 타이핑을 줄이기 위해 존재한다.

```
<typeAliases>
  <typeAlias alias="mvo" type="com.mvc.vo.MemberVO"/>
</typeAliases>

<select id="firstMember" resultType="mvo">
  SELECT * FROM member WHERE id = 100
</select>
```

MyBatis - config

▶ typeAlias

- ▶ typeAlias를 패키지로 설정하고 bean의 어노테이션을 통해 설정할 수도 있다.

```
<typeAliases>
  <!-- <typeAlias alias="mvo" type="com.mvc.vo.MemberVO"/> -->
  <package name="com.mvc.vo"/>
</typeAliases>
```

```
@Alias("mvo")
public class MemberVO {
    int id;
    String password;
    String name;
    int age;
    String gender;
    String email;
```

MyBatis - config

▶ typeAlias

- ▶ typeAlias는 기존 타입 중 많이 사용하는 타입을 별칭으로 지정하여 사용할 수 있도록 제공해 주고 있다.
- ▶ <http://www.mybatis.org/mybatis-3/ko/configuration.html>

별칭	매핑된 타입
_byte	byte
_long	long
_short	short
_int	int
_integer	int
_double	double

MyBatis - config

▶ Mapper

- ▶ SQL 구문을 정의한 xml을 정의하는 설정 정보
- ▶ 설정 정보는 크게 네 가지로 설정할 수 있다.

```
<!-- 클래스패스의 상대경로의 리소스 사용 -->
<mappers>
  <mapper resource="org/mybatis/builder/AuthorMapper.xml"/>
  <mapper resource="org/mybatis/builder/BlogMapper.xml"/>
  <mapper resource="org/mybatis/builder/PostMapper.xml"/>
</mappers>
```

```
<!-- 절대경로의 url을 사용 -->
<mappers>
  <mapper url="file:///var/mappers/AuthorMapper.xml"/>
  <mapper url="file:///var/mappers/BlogMapper.xml"/>
  <mapper url="file:///var/mappers/PostMapper.xml"/>
</mappers>
```

```
<!-- 매퍼 인터페이스를 사용 -->
<mappers>
  <mapper class="org.mybatis.builder.AuthorMapper"/>
  <mapper class="org.mybatis.builder.BlogMapper"/>
  <mapper class="org.mybatis.builder.PostMapper"/>
</mappers>
```

```
<!-- 매퍼로 패키지내 모든 인터페이스를 등록 -->
<mappers>
  <package name="org.mybatis.builder"/>
</mappers>
```

MyBatis - Mapper

▶ Mapper

- ▶ 실제 작동되는 SQL 구문을 만들기 위해서 Mapper를 사용한다.
- ▶ Mapper에 사용되는 주요 태그는 아래와 같다.
 - ▶ resultMap - 데이터베이스 결과데이터를 객체에 로드하는 방법을 정의하는 엘리먼트
 - ▶ sql - 다른 구문에서 재사용하기 위한 SQL 조각
 - ▶ insert - 매핑된 INSERT 구문.
 - ▶ update - 매핑된 UPDATE 구문.
 - ▶ delete - 매핑된 DELETE 구문.
 - ▶ select - 매핑된 SELECT 구문.

MyBatis - Mapper

- ▶ Mapper <SELECT>
 - ▶ Select 관련 쿼리를 넣는 태그
 - ▶ 가장 중요한 태그로서 주요 속성들은 다음과 같다.

속성	설명
id	구문을 찾기 위해 사용될 수 있는 네임스페이스내 유일한 구분자
parameterType	구문에 전달될 파라미터의 패키지 경로를 포함한 전체 클래스명이나 별칭
resultType	이 구문에 의해 리턴되는 기대타입의 패키지 경로를 포함한 전체 클래스명이나 별칭.
resultMap	외부 resultMap 의 참조명

MyBatis - Mapper

- ▶ Mapper <SELECT>

- ▶ parameterType, resultType 사용법

```
<select id="getName" parameterType="String" resultType="String">
    SELECT NAME FROM member WHERE email = #{email}
</select>
<select id="getMember" parameterType="String" resultType="map">
    SELECT * FROM member WHERE email = #{email}
</select>
<select id="getList" resultType="map">
    SELECT * FROM member
</select>
```

MyBatis - Mapper

▶ Mapper <SELECT>

- ▶ parameterType, resultType 사용법
- ▶ selectExample1.jsp

```
<body>
<%
    SqlSessionFactory sqlSessionFactory = SqlSessionManager.getSqlSession();
    SqlSession sqlSession = sqlSessionFactory.openSession();

    String name = "";
    Map map = new HashMap();
    List<Map> list = null;

    try{
        // 단일 값 가져오기
        name = sqlSession.selectOne("member.getName", "aaa@aaa.com") ;
        // 다중 값 가져오기
        map = sqlSession.selectOne("member.getMember", "aaa@aaa.com") ;
        // 리스트로 받아오기
        list = sqlSession.selectList("member.getList") ;
    }catch(Exception e){
        e.printStackTrace() ;
    }

    %>
    <p>이름 : <%=name%> </p>
    <p>세부내용 : <%=map.toString() %></p>
    <p>리스트 :<br>
    <%
        for(Map m:list){
    %>
        <%=m.toString() %><br>
    <%
        }
    %>
    </p>
</body>
```

MyBatis - Mapper

- ▶ Mapper <ResultMap>

- ▶ 데이터를 가져올 때 작성되는 JDBC코드를 대부분 줄여주는 역할을 담당한다.
- ▶ resultMap은 간단한 구문에서는 매핑이 필요하지 않고 복잡한 구문에서 관계를 서술하기 위해 필요하다.
- ▶ com.mvc.vo.MemberMapper.java

```
public class MemberMapper {  
    int userId;  
    String userPass;  
    String userName;  
    int userAge;  
    String userGender;  
    String userEmail;  
}
```

MyBatis - Mapper

- ▶ Mapper <ResultMap>

- ▶ member.xml

```
<resultMap type="com.mvc.vo.MemberMapper" id="memberMap">
  <result property="userId" column="id"/>
  <result property="userPass" column="password"/>
  <result property="userName" column="name"/>
  <result property="userAge" column="age"/>
  <result property="userGender" column="gender"/>
  <result property="userEmail" column="email"/>
</resultMap>

<select id="getListByResultMap" resultMap="memberMap">
  SELECT * FROM member
</select>
```

MyBatis - Mapper

- ▶ Mapper <ResultMap>
- ▶ selectExample2.jsp

```
<body>
<%
    SqlSessionFactory sqlSessionFactory = SqlSessionManager.getSqlSession();
    SqlSession sqlSession = sqlSessionFactory.openSession();

    MemberMapper mapper = null;
    List<MemberMapper> list = null;

    try{
        // 리스트로 받아오기1
        list = sqlSession.selectList("member.getListByResultMap") ;

    }catch(Exception e){
        e.printStackTrace() ;
    }
%>
<p>
리스트 : <br>
<%
    for(MemberMapper m : list){
        %>
        <%=m.toString()%><br>
        <%
    }
%>
</p>
</body>
```

MyBatis - Mapper

- ▶ Mapper <ResultMap>

- ▶ 만약 vo의 멤버변수 이름과 컬럼 이름이 같다면 굳이 resultMap을 설정할 필요가 없다.

- ▶ MemberVO.java

```
public class MemberVO {  
    int id;  
    String password;  
    String name;  
    int age;  
    String gender;  
    String email;  
}
```


MyBatis - Mapper

- ▶ Mapper <ResultMap>

- ▶ mapper.xml

```
<select id="getListByVO" resultType="com.mvc.vo.MemberVO">  
    SELECT * FROM member  
</select>
```

MyBatis - Mapper

▶ Mapper <ResultMap>

▶ selectExample2.jsp(1)

```
SqlSessionFactory sqlSessionFactory = SqlSessionManager.getSqlSession();
SqlSession sqlSession = sqlSessionFactory.openSession();

MemberMapper mapper = null;
List<MemberMapper> list1 = null;
List<MemberVO> list2 = null;
try{
    // 리스트로 받아오기1
    list1 = sqlSession.selectList("member.getListByResultMap") ;
    // 리스트로 받아오기2
    list2 = sqlSession.selectList("member.getListByVO") ;
}catch(Exception e){
    e.printStackTrace() ;
}
```

MyBatis - Mapper

- ▶ Mapper <ResultMap>
- ▶ selectExample2.jsp(2)

```
<p>
리스트 : <br>
<%
    for(MemberMapper m : list1){
        %>
        <%=m.toString()%><br>
        <%
    }
%>
</p>
<p>
리스트 : <br>
<%
    for(MemberVO m : list2){
        %>
        <%=m.toString()%><br>
        <%
    }
%>
</p>
```

MyBatis - Mapper

- ▶ Mapper <Insert>

- ▶ member.xml

```
<insert id="setMember" parameterType="mvo">  
    INSERT INTO MEMBER(ID,PASSWORD,NAME,AGE,GENDER,EMAIL)  
    VALUES("#{id}","#{password}","#{name}","#{age}","#{gender}","#{email}")  
</insert>
```

MyBatis - Mapper

- ▶ Mapper <Insert>
 - ▶ insertExample1.jsp

```
SqlSessionFactory sqlSessionFactory = SqlSessionManager.getSqlSession();  
SqlSession sqlSession = sqlSessionFactory.openSession();
```

```
MemberVO vo = new MemberVO();  
vo.setId(105);  
vo.setName("임꺽정");  
vo.setPassword("9876");  
vo.setAge(41);  
vo.setGender("Y");  
vo.setEmail("lim@aaa.com");
```

```
int i = 0;
```

```
try{  
    //데이터 저장  
    i = sqlSession.insert("member.setMember", vo);
```

```
}catch(Exception e){  
    e.printStackTrace() ;  
}
```

```
if(i>0)sqlSession.commit();
```

MyBatis - Mapper

- ▶ Mapper <update>

- ▶ mapper.xml

```
<update id="updateMember" parameterType="mvo">  
    UPDATE MEMBER SET EMAIL = #{email} WHERE ID = #{id}  
</update>
```

MyBatis - Mapper

▶ Mapper <update>

▶ updateExample1.jsp

```
SqlSessionFactory sqlSessionFactory = SqlSessionManager.getSqlSession();
SqlSession sqlSession = sqlSessionFactory.openSession();

MemberVO vo = new MemberVO();
vo.setId(105);
vo.setEmail("jung@aaa.com");

int i = 0;

try{
    //데이터 업데이트
    i = sqlSession.update("member.updateMember", vo);
}catch(Exception e){
    e.printStackTrace() ;
}

if(i>0)sqlSession.commit();
```


MyBatis - Mapper

- ▶ Mapper <delete>

- ▶ mapper.xml

```
<delete id="deleteMember" parameterType="mvo">  
    DELETE FROM MEMBER WHERE ID = #{id}  
</delete>
```

MyBatis - Mapper

▶ Mapper <delete>

▶ deleteExample1.jsp

```
<%
    SqlSessionFactory sqlSessionFactory = SqlSessionManager.getSqlSession();
    SqlSession sqlSession = sqlSessionFactory.openSession();

    MemberVO vo = new MemberVO();
    vo.setId(105);

    int i = 0;

    try{
        //데이터 업데이트
        i = sqlSession.update("member.deleteMember", vo);
    }catch(Exception e){
        e.printStackTrace() ;
    }

    if(i>0)sqlSession.commit();
%>
```

MyBatis – insert key setting

▶ insert key setting

- ▶ 만약 <insert> 실행 시 테이블에 auto increment와 같은 설정이 담긴 키 를 가진 테이블 이라면 옵션을 통해 테이블에 키를 자동 갱신할 수 있다.
- ▶ 또한 insert 후에 키 값을 그대로 리턴 받을 수도 있다.

속성명	기능
useGeneratedKeys	(입력(insert, update)에만 적용) 데이터베이스에서 내부적으로 생성한 키 (예를들어 MySQL또는 SQL Server와 같은 RDBMS의 자동 증가 필드)를 받는 JDBC getGeneratedKeys메소드를 사용하도록 설정하다. 디폴트는 false 이다.
keyProperty	(입력(insert, update)에만 적용) getGeneratedKeys 메소드나 insert 구문의 selectKey 하위 엘리먼트에 의해 리턴된 키를 셋팅할 프로퍼티를 지정. 디폴트는 셋팅하지 않는 것이다. 여러개의 칼럼을 사용한다면 프로퍼티명에 콤마를 구분자로 나열할수 있다.

MyBatis – insert key setting

- ▶ insert key setting
 - ▶ MEMBER2.sql

```
create table java.MEMBER2 (  
  ID int AUTO_INCREMENT PRIMARY KEY,  
  PASSWORD varchar(20),  
  NAME varchar(15),  
  AGE int,  
  GENDER varchar(5),  
  EMAIL varchar(30),  
  unique key (ID)  
) engine=InnoDB character set = utf8;
```

MyBatis – insert key setting

- ▶ insert key setting

- ▶ member.xml

```
<insert id="setMember2" parameterType="mvo" useGeneratedKeys="true"
keyProperty="id">
    INSERT INTO MEMBER2(PASSWORD,NAME,AGE,GENDER,EMAIL)
    VALUES(#{password},#{name},#{age},#{gender},#{email})
</insert>
```

MyBatis – insert key setting

- ▶ insert key setting
- ▶ insertExample2.jsp

```
<%
    SqlSessionFactory sqlSessionFactory = SqlSessionManager.getSqlSession();
    SqlSession sqlSession = sqlSessionFactory.openSession();

    MemberVO vo = new MemberVO();
    vo.setName("임꺽정");
    vo.setPassword("9876");
    vo.setAge(41);
    vo.setGender("Y");
    vo.setEmail("lim@aaa.com");

    int i = 0;

    try{
        //데이터 저장
        i = sqlSession.insert("member.setMember2", vo);

    }catch(Exception e){
        e.printStackTrace();
    }

    if(i>0)sqlSession.commit();
%>
<p>입력한 Id 값 : <%=vo.getId()%></p>
```

MyBatis – insert key setting

▶ insert key setting

- ▶ 만약 autoincrement 세팅을 못하는 데이터베이스 (예를 들면 Oracle)에서는 안쪽에 `<selectKey>` 태그를 둔다.
- ▶ Key 태그의 옵션은 다음과 같다

속성	설명
keyProperty	selectKey구문의 결과가 셋팅될 대상 프로퍼티.
keyColumn	리턴되는 결과셋의 칼럼명은 프로퍼티에 일치한다. 여러개의 칼럼을 사용한다면 칼럼명의 목록은 콤마를 사용해서 구분한다.
resultType	결과값의 타입. 마이바티스는 이 기능을 제거할 수 있지만 추가하는게 문제가 되지는 않을것이다. 마이바티스는 String을 포함하여 키로 사용될 수 있는 간단한 타입을 허용한다.
order	BEFORE 또는 AFTER를 셋팅할 수 있다. BEFORE로 설정하면 키를 먼저 조회하고 그 값을 keyProperty 에 셋팅한 뒤 insert 구문을 실행한다. AFTER로 설정하면 insert 구문을 실행한 뒤 selectKey 구문을 실행한다. 오라클과 같은 데이터베이스에서는 insert 구문 내부에서 일관된 호출형태로 처리한다.
statementType	위 내용과 같다. 마이바티스는 Statement, PreparedStatement 그리고 CallableStatement을 매핑하기 위해 STATEMENT, PREPARED 그리고 CALLABLE 구문타입을 지원한다.

MyBatis – insert key setting

- ▶ insert key setting

- ▶ member.xml

```
<insert id="setMember3" parameterType="mvo">
  <selectKey keyProperty="id" resultType="int" order="BEFORE">
    select MAX(ID)+1 from MEMBER
  </selectKey>
  INSERT INTO MEMBER(ID,PASSWORD,NAME,AGE,GENDER,EMAIL)
  VALUES(#{id},#{password},#{name},#{age},#{gender},#{email})
</insert>
```

MyBatis – insert key setting

- ▶ insert key setting
 - ▶ insertExample3.jsp

```
<%
    SqlSessionFactory sqlSessionFactory = SqlSessionManager.getSqlSession();
    SqlSession sqlSession = sqlSessionFactory.openSession();

    MemberVO vo = new MemberVO();
    vo.setName("임꺽정");
    vo.setPassword("9876");
    vo.setAge(41);
    vo.setGender("Y");
    vo.setEmail("lim@aaa.com");

    int i = 0;

    try{
        //데이터 저장
        i = sqlSession.insert("member.setMember3", vo);

    }catch(Exception e){
        e.printStackTrace() ;
    }

    if(i>0)sqlSession.commit();
%>
<p>입력한 id : <%=vo.getId()%></p>
```

동적 SQL

- ▶ 동적 SQL
 - ▶ MyBatis 의 강력한 기능 중 하나
 - ▶ 상황에 따라 쿼리를 바꿔야 할 경우 사용한다.
 - ▶ 크게는 if, choose, where, set, foreach 가 있다.

동적 SQL - if

▶ 동적 SQL - if

▶ if는 선택적으로 SQL 구문을 넣을 경우 사용한다.

▶ member.xml

```
<select id="getMemberList1" parameterType="mvo" resultType="mvo">
    SELECT * FROM member
    WHERE 1=1
    <if test="id != 0">
        AND id = #{id}
    </if>
    <if test="email != null">
        AND email like #{email}
    </if>
    <if test="gender != null">
        AND gender = #{gender}
    </if>
</select>
```

동적 SQL - if

▶ 동적 SQL - if

▶ selectListExample1.jsp

```
<%  
    SqlSessionFactory sqlSessionFactory = SqlSessionManager.getSqlSession();  
    SqlSession sqlSession = sqlSessionFactory.openSession();  
  
    MemberVO vo = new MemberVO();  
    List<MemberVO> list1 = null;  
    try{  
        // 리스트로 받아오기1  
        list1 = sqlSession.selectList("member.getMemberList1", vo);  
    }catch(Exception e){  
        e.printStackTrace() ;  
    }  
%>  
<p>  
리스트 : <br>  
<%  
    for(MemberVO m : list1){  
        %>  
        <%=m.toString()%><br>  
        <%  
    }  
%>  
</p>
```

동적 SQL - choose

▶ 동적 SQL – choose

- ▶ if, else if , else 문 처럼 실제 조건 중 다중 조건을 통한 쿼리를 완성시킬 경우 사용
- ▶ member.xml

```
<select id="getMemberList2" parameterType="mvo" resultType="mvo">
  SELECT * FROM member
  WHERE 1=1
  <choose>
    <when test="id != 0">
      AND id = #{id}
    </when>
    <when test="email != null">
      AND email like #{email}
    </when>
    <when test="gender != null">
      AND gender = #{gender}
    </when>
    <otherwise>
      AND 1 = 1
    </otherwise>
  </choose>
</select>
```

동적 SQL - choose

▶ 동적 SQL – choose

▶ selectListExample1.jsp

```
try{
    // 리스트로 받아오기1
    //list1 = sqlSession.selectList("member.getMemberList1", vo);
    // 리스트로 받아오기2
    list1 = sqlSession.selectList("member.getMemberList2", vo);
}catch(Exception e){
    e.printStackTrace() ;
}
```


동적 SQL - where

- ▶ 동적 SQL - where

- ▶ 동적으로 변하는 조건 커리에 대응하기 위해 만들어진 기능

- ▶ member.xml

```
<select id="getMemberList1" parameterType="mvo" resultType="mvo">
  SELECT * FROM member
  <!-- where 1=1 -->
  <where>
    <if test="id != 0">
      AND id = #{id}
    </if>
    <if test="email != null">
      AND email like #{email}
    </if>
    <if test="gender != null">
      AND gender = #{gender}
    </if>
  </where>
</select>
```

동적 SQL - Set

▶ 동적 SQL – set

- ▶ update에서 동적으로 변하는 update의 set 리스트들에 대응하기 위해 만들어진 컬럼
- ▶ member.xml

```
<update id="updateMember2" parameterType="mvo">
    UPDATE MEMBER
    <set>
        <if test="name != null">name=#{name},</if>
        <if test="password != null">password=#{password},</if>
        <if test="age != 0">age=#{age},</if>
        <if test="gender != null">gender=#{gender},</if>
        <if test="email != null">email=#{email}</if>
    </set>
    WHERE ID = #{id}
</update>
```

동적 SQL - Set

▶ 동적 SQL - set

▶ updateExample2.jsp

```
<%  
SqlSessionFactory sqlSessionFactory = SqlSessionManager.getSqlSession();  
SqlSession sqlSession = sqlSessionFactory.openSession();  
  
MemberVO vo = new MemberVO();  
vo.setId(104);  
vo.setEmail("kkuag@aaa.com");  
vo.setAge(50);  
  
int i = 0;  
  
try{  
    //데이터 업데이트  
    i = sqlSession.update("member.updateMember2", vo);  
}catch(Exception e){  
    e.printStackTrace();  
}  
  
if(i>0)sqlSession.commit();  
%>
```

동적 SQL - foreach

- ▶ 동적 SQL – foreach
 - ▶ 쿼리 작성 시 반복적인 구문을 작성해야 할 경우 쓰이는 태그
 - ▶ foreach는 아래와 같은 속성들을 가진다.
 - ▶ collection = 전달받은 인자. List나 Array 형태만 가능
 - ▶ item = 전달받은 인자값을 alias 명으로 대체
 - ▶ open = 해당 구문이 시작될때 삽입할 문자열
 - ▶ close = 해당 구문이 종료될때 삽입할 문자열
 - ▶ separator = 반복 되는 사이에 출력할 문자열
 - ▶ index=반복되는 구문 번호이다. 0부터 순차적으로 증가

동적 SQL - foreach

- ▶ 동적 SQL – foreach

- ▶ member.xml

```
<select id="getMemberList3" parameterType="List"
  resultType="mvo">
  SELECT * FROM member
  WHERE email in
    <foreach item="item" index="index" collection="list" open="("
      separator="," close=")">
      #{item}
    </foreach>
</select>
```

동적 SQL - foreach

- ▶ 동적 SQL – foreach
- ▶ selectListExample2.jsp

```
<%  
    SqlSessionFactory sqlSessionFactory = SqlSessionManager.getSqlSession();  
    SqlSession sqlSession = sqlSessionFactory.openSession();  
  
    List<MemberVO> list1 = null;  
    List<String> emails = new ArrayList<String>();  
    emails.add("aaa@aaa.com");  
    emails.add("aaa@bbb.com");  
  
    try{  
        // 리스트로 받아오기1  
        list1 = sqlSession.selectList("member.getMemberList3", emails);  
    }catch(Exception e){  
        e.printStackTrace() ;  
    }  
%>  
<p>  
리스트 : <br>  
<%  
    for(MemberVO m : list1){  
        %>  
        <%=m.toString()%><br>  
        <%  
    }  
%>  
</p>
```

Interface를 이용한 Setting

▶ Interface를 이용한 세팅

- ▶ MyBatis는 두 가지 세팅 방법이 있으며 하나는 Mapper namespace 를 직접 입력하여 호출하는 방법과 Interface 클래스를 만들어서 매핑, 호출하는 방법이 있다.
- ▶ 지금까지 했던 방식은 Mapper namespace를 이용하여 호출하는 방법이며 이 방법은 설정이 간단하고 파일을 추가적으로 생성할 필요가 없으며 코딩이 줄어든다는 장점이 있다.
- ▶ 하지만 오타로 인한 에러 유발과 가독성의 떨어짐, SqlSession을 이용하여 조회할 시 조회 성격을 알아야 정확하게 사용할 수 있다.
 - ▶ 단일이면 selectOne, 리스트면 selectList

Interface를 이용한 Setting

▶ Interface를 이용한 세팅

- ▶ 인터페이스 세팅을 통한 방식은 파일이나 코딩의 생산량이 늘어난다.
- ▶ 하지만 Mapper를 열어보지 않아도 어떤 ID 값을 갖고 있는지 한눈에 볼 수 있으며 쿼리를 제외하고 호출하는 부분만 있기 때문에 한번에 볼 수 있다.
- ▶ 오타의 우려가 적어지며 이클립스 상에서 자동 오류 체크가 가능하고 인터페이스를 통해 Single Row인지 Multi Row인지 확인이 가능하다.
- ▶ mvc2 패턴 상에서 장기적으로 봤을 때 전자보단 후자가 훨씬 관리가 용이하다

Interface를 이용한 Setting

- ▶ Interface를 이용한 세팅
 - ▶ member2.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.mvc.mapper.MemberMapper">
  <select id="getCount" resultType="int">
    SELECT
    count(*)
    FROM
    member
  </select>
  <select id="getList" resultType="mvo">
    SELECT * FROM member
  </select>
</mapper>
```

Interface를 이용한 Setting

- ▶ Interface를 이용한 세팅

- ▶ com.mvc.mapper.MemberMapper.java

```
package com.mvc.mapper;  
  
import java.util.List;  
  
import com.mvc.vo.MemberVO;  
  
public interface MemberMapper {  
    public int getCount();  
  
    public List<MemberVO> getList();  
}
```

Interface를 이용한 Setting

- ▶ Interface를 이용한 세팅
 - ▶ configuration.xml

```
<mappers>  
    <mapper resource="com/mvc/dao/member.xml" />  
    <mapper resource="com/mvc/dao/member2.xml" />  
</mappers>
```

Interface를 이용한 Setting

- ▶ Interface를 이용한 세팅
- ▶ interfaceMapper.jsp

```
<%@page import="com.mvc.conf.SqlSessionManager"%>
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@page import="org.apache.ibatis.session.SqlSessionFactory"%>
<%@page import="org.apache.ibatis.session.SqlSession"%>
<%@page import="java.util.*"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
<%
    SqlSessionFactory sqlSessionFactory = SqlSessionManager.getSqlSession();
    SqlSession sqlSession = sqlSessionFactory.openSession();
    MemberMapper mapper = sqlSession.getMapper(MemberMapper.class);

    int i = 0;
    try{
        i = mapper.getCount();
    }catch(Exception e){
        e.printStackTrace() ;
    }
%>
<p>현재 데이터 갯수 : <%=i%></p>
</body>
</html>
```