1. Mybatis 란?

자바에선 데이터베이스 프로그래밍을 하기 위해 JDBC(자바에서 제공하는 데이터베이스 프로그래밍 API)를 제공하고, JDBC는 관계형 데이터 베이스를 사용하기 위해 다양한 API를 제공한다.

다양한 관계형 데이터베이스를 지원하기 위해 JDBC는 세부적인 작업이 가능하게 작업별로 각각의 메소드를 호출하게된다. 이러한 사항들은 다수의 메소드를 호출하고 관련된 객체를 해제 해야하는 단점이 있다. Mybatis는 JDBC보다 좀더 편하게 사용하기 위해 개발 되었다.

Mybatis는 객체지향 어플리케이션에서 관계형 데이터베이스를 쉽게 사용할 수 있도록 도와주는 데이터 맵핑 프레임워크로서 Mybatis의 장점은 다음과 같다.

SQL 및 프로시져구문의 독립

복잡한 JDBC코드를 걷어내며 깔끔한 소스코드를 유지할 수 있다.

수동적인 파라미터 설정과 쿼리 결과에 대한 맵핑 구문을 제거할 수 있다.

Mybatis는 별도의 XML 문서에 맵핑된 프로시져와 SQL 구문을 연동하여 데이터베이스와 연동할 수 있도록 도와주어 데이터베이스 개발에 집중할 수 있도록 돕는다. 결과적으로 복잡한 JDBC 연동 코드나 트랜잭션 코드를 간소화시킬 수 있도록 도와주며 이는 결과적으로 소스코드의 유지보수를 용이하게 돕는다. 같은 맥락이지만 ResultSet과 같이 결과값을 맵핑하는 객체 또한 자동화시켜주어 많은 라인의 소스코드를 줄일 수 있다.

그밖에도 데이터베이스 개발이 Java와 분리된다는 것은 프로젝트 협업시에도 많은 이점을 제공해 줄 수 있다.

MyBatis의 특징

- 간단하다 : 간단한 퍼시스턴스 프레임워크
- 생산성 : 62%정도 줄어드는 코드 , 간단한 설정
- 성능 : 구조적강점(데이터 접근 속도를 높여주는 Join 매핑)
  여러가지 방식의 데이터를 가져오기 전략 (가져오기 미루기 , SQL 줄이기 기법)
- 관심사의 분리 : 설계를 향상 (유지보수성)
  리소스를 관리하여 계층화를 지원(Connection,PreparedStetement,ResultSet)
- 작업의 분배 : 팀을 세분화하는 것을 도움
- SQL문이 애플리케이션 소스 코드로부터 완전 분리
- 이식성 : 어떤 프로그래밍 언어로도 구현가능 (자바,C#,.NET,RUBY)
- 오픈소스이며 무료이다.

2. Mybatis 설치

Mybatis 를 사용하기 위해 mybatis-x.x.x.jar 파일을 클래스패스에 두어야 한다. Dynamic Web Project 의 경우에는 이 파일을 다운로드하여 WEB-INF/lib 폴더에 저장한다.

https://github.com/mybatis/mybatis-3/releases 사이트에 방문하여

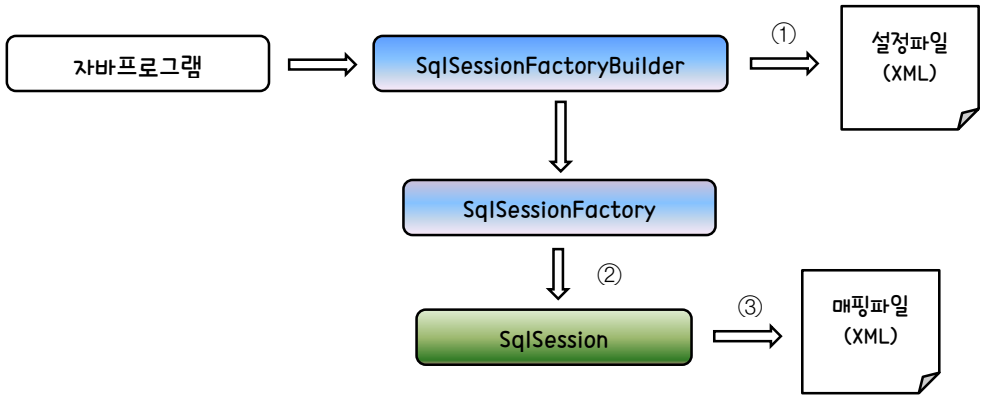
mybatis-3.4.1.zip 을 다운로드하고 압축을 푼 후에 mybatis-3.4.1.jar 파일을 WEB-INF/lib 폴더에 저장한다.

XML 파일에서 SqlSessionFactory 빌드하기

모든 Mybatis 애플리케이션은 SqlSessionFactory 객체를 사용한다. SqlSessionFactory 객체는 SqlSessionFactoryBuilder 를 사용하여 만들수 있다. SqlSessionFactoryBuilder 는 XML 설정파일에서 SqlSessionFactory 객체를 빌드할 수 있다.

XML 파일에서 SqlSessionFactory 객체를 빌드하는 것은 매우 간단한다. 설정을 위해 클래스패스 자원을 사용하는 것을 추천하나 파일 경로나 file:// URL 로부터 만들어진 InputStream 객체를 사용할 수도 있다. Mybatis 는 서 자원 로드를 처리하는데 도움을 주는 Resources 라는 유틸성 클래스를 제공한다.

```
String resource = "resource/mybatis-config.xml";
InputStream inputStream = Resources.getResourceAsStream(resource);
SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(inputStream);
SqlSession session = sqlSessionFactory.openSession();
```



① SqlSessionFactoryBuilder 는 데이터베이스 접속 정보 등이 기재된 Mybatis 설정 파일을 읽어서 SqlSessionFactory 객체를 생성한다

② SqlSessionFactory 는 세션을 오픈하면서 SqlSession 객체를 생성한다.

③ 필요한 SQL 문 수행을 요청할 때마다 SqlSession 은 지시된 SQL 을 매핑 파일에서 찾아서 수행시킨다.

[Mybatis 설정파일]

```
<configuration>
  <environments default="development">
    <environment id="development">
      <transactionManager type="JDBC"/>
      <dataSource type="POOLED">
        <property name="driver" value="oracle.jdbc.driver.OracleDriver"/>
        <property name="url" value="jdbc:oracle:thin:@localhost:1521:xe"/>
        <property name="username" value="jdbctest"/>
        <property name="password" value="jdbctest"/>
      </dataSource>
    </environment>
  </environments>
  <mappers>
    <mapper resource="resource/VisitorMapper.xml"/>
  </mappers>
</configuration>
```

[Mybatis 매핑파일]

```
<?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="resource.VisitorMapper">
        <select id="selectVisitor"   resultType="model.vo.VisitorVO">
        select name, to_char(writedate,'yyyy"년"mm"월"dd"일"') writedate, memo from visitor
        </select>
         <select id="selectVisitor1"   resultType="model.vo.VisitorVO">
        select name, to_char(writedate,'yyyy"년"mm"월"dd"일"') writedate, memo
                                from visitor where id <![CDATA[<]]> 5
        </select>
        <insert id="insertVisitor"   parameterType="model.vo.VisitorVO">
        insert into visitor (name, writedate, memo) values (#{name},sysdate, #{memo})
        </insert>
        <select id="searchVisitor" parameterType="string" resultType= "model.vo.VisitorVO">
        select name, to_char(writedate,'yyyy"년"mm"월"dd"일"') writedate, memo
                                from visitor where memo like '%'||#{key}||'%'
        </select>
</mapper>
```

```
<?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="resource.MeetingMapper">
  <select id="selectMeeting"   resultType="model.vo.MeetingVO">
    select id, name, title, to_char(meetingdate,'yyyy"년"mm"월"dd"일" HH24"시"mi"분"')
      meetingDate from Meeting
  </select>
  <select id="searchMeeting"   parameterType="string" resultType="model.vo.MeetingVO">
    select id, name, title, to_char(meetingdate,'yyyy"년"mm"월"dd"일" HH24"시"mi"분"')
    meetingDate from meeting where title like '%'||#{key}||'%'
  </select>
  <insert id="insertMeeting"   parameterType="model.vo.MeetingVO">
    <selectKey resultType="int" keyProperty="id" order="BEFORE">
      select meeting_seq.nextval from dual
    </selectKey>
    insert into meeting (id, name, title, meetingdate) values (#{id}, #{name}, #{title},
          to_date(#{meetingDate}, 'yyyy-mm-dd"T"hh24:mi'))
  </insert>
  <delete id="deleteMeeting"   parameterType="_int"   >
    delete from meeting where id = #{id}
  </delete>
  <update id="updateMeeting"   parameterType="model.vo.MeetingVO"   >
    update meeting set
                name = #{name}, title = #{title},
                meetingdate = to_date(#{meetingDate}, 'yyyy-mm-dd"T"hh24:mi' )
    where id = #{id}
  </update>
</mapper>
```

3. Mybatis 구현

SqlSession

HTTP 요청을 받을때마다 만들고 응답을 리턴할때마다 SqlSession 을 닫는다. 언제나 finally 블록에서 닫아야만 한다. 다음은 SqlSession 을 닫는 것을 확인하는 표준적인 형태다.

```
SqlSession session = sqlSessionFactory.openSession();
try {
        session.insert(...);
        session.update(...);
        session.delete(...);
} finally {
        session.close();
}
```

```
try (SqlSession session = sqlSessionFactory.openSession()) {
        session.insert(...);
        session.update(...);
        session.delete(...);
}
```

SqlSession 의 주요 메서드

```
T selectOne(String statement, Object parameter)
List<E> selectList(String statement, Object parameter)
Map<K,V> selectMap(String statement, Object parameter, String mapKey)
int insert(String statement, Object parameter)
int update(String statement, Object parameter)
int delete(String statement, Object parameter)
```

Mapper 파일 작성

- typeAliases

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

```
<typeAliases>
  <typeAlias alias="Author" type="domain.blog.Author"/>
  <typeAlias alias="Comment" type="domain.blog.Comment"/>
</typeAliases>
```

다음은 Mybatis 에 내장된 별칭으로서 대소문자를 구분한다.

별칭	매핑된 타입
_byte	byte
_long	long
_short	short
_int	int
_integer	int
_double	double
_float	float
_boolean	boolean
string	String
byte	Byte
long	Long
short	Short
int	Integer

integer	Integer
double	Double
float	Float
boolean	Boolean
date	Date
decimal	BigDecimal
bigdecimal	BigDecimal
object	Object
map	Map
hashmap	HashMap
list	List
arraylist	ArrayList
collection	Collection
iterator	Iterator

- SELECT 명령에 대한 Mapper 파일 작성 예

```
<select id="selectVisitor"  resultType="model.vo.VisitorVO">
   select * from visitor
</select>
<select id="selectVisitor1"  resultType="model.vo.VisitorVO">
   select * from visitor order by name
</select>
<select id="searchVisitor"  parameterType="java.lang.String" resultType="model.vo.VisitorVO">
   select * from visitor where memo like '%'||#{key}||'%'
</select>
```

```
<select id="selectNews"  parameterType="_int"  resultType="model.vo.NewsVO">
   select * from news where <![CDATA[ viewcount > 10 ]]>
</select>
<select id="selectNewsOne"  parameterType="_int" resultType="model.vo.NewsVO">
   select * from news where id = #{id}
</select>
<select id="selectNewsWriter"  parameterType="string"  resultType="model.vo.NewsVO">
   select * from news where writer = #{writer}
</select>
<select id="searchNews"  parameterType="java.lang.String" resultType="model.vo.NewsVO">
   select * from news where content like '%'||#{key}||'%'
</select>
```

- insert 명령에 대한 Mapper 파일 작성 예

```
[ 시퀀스 사용 : Oracle ]
<insert id="insertNews"  parameterType="model.vo.NewsVO">
        <selectKey resultType="_int" keyProperty="id" order="BEFORE">
                SELECT news_seq.nextval FROM dual
        </selectKey>
  insert into news (id, writer, title, content, writedate, viewcount)
  values (#{id},#{writer},#{title},#{content},#{writedate},#{viewcount})
</insert>
[ 자동 증가 컬럼 사용 : MySQL ]
<insert id="insertAuthor" useGeneratedKeys="true" keyProperty="id">
  insert into Author (username,password,email,bio)
  values (#{username},#{password},#{email},#{bio})
</insert>
```

- delete 명령에 대한 Mapper 파일 작성 예

```
<delete id="deleteVisitor"  parameterType="string"  >
  delete from visitor where name = #{name}
</delete>
```

```
<delete id="deleteNews" parameterType="_int”>
  delete from news where id = #{id}
</delete>
```

- update 명령에 대한 Mapper 파일 작성 예

```
<update id="updateNews"   parameterType="model.vo.NewsVO">
  update news set writer = #{writer}, title = #{title}, content = #{content} where id = #{id}
</update>
<update id="updateCount" parameterType="_int”>
  update news set viewcount = viewcount+1 where id =   #{id}
</update>
```

- DAO(자바 코드) 구현 예

[생성자 메서드]

```
try {
    String resource = "resource/mybatis-config.xml";
    InputStream inputStream = Resources.getResourceAsStream(resource);
    sqlSessionFactory = new SqlSessionFactoryBuilder().build(inputStream);
} catch (Exception e) {
    e.printStackTrace();
}
```

[insert() 메서드]

```
boolean flag = false;
SqlSession session = sqlSessionFactory.openSession(true);
try {
    String statement = "resource.VisitorMapper.insertVisitor";
    session.insert(statement, vo);
    flag = true;
} catch (Exception e) {
    e.printStackTrace();
} finally {
    session.close();
}
return flag;
```

[delete() 메서드]

```
int deleteNum = 0;
SqlSession session = sqlSessionFactory.openSession(true);
try {
    String statement = "resource.VisitorMapper.deleteVisitor";
    deleteNum = session.delete(statement, id);
} catch (Exception e) {
    e.printStackTrace();
} finally {
    session.close();
}
return deleteNum;
```

[list() 메서드]

```
public List<VisitorVO> list() {
    List<VisitorVO> list = null;
    SqlSession session = sqlSessionFactory.openSession();
    try {
      String statement =
      "resource.VisitorMapper.selectVisitor";
      list = session.selectList(statement);
    } catch (Exception e) {
      e.printStackTrace();
    } finally {
      session.close();
    }
    return list;
}
```

```
r";
return list;
```

[Advanced MyBatis]

(1) 수행되는 SQL 문의 로그 남기기

- MyBatis 설정파일의 <settings> 태그안에 다음 태그를 추가한다.

```
<setting name="logImpl" value="LOG4J2"/>
```

- Log4J 설정 파일에 다음 <Logger> 태그를 추가한다.

```
<Logger name="resource.MeetingMapper" level="DEBUG" additivity="false">
 <AppenderRef ref="xxxxx" />
</Logger>
```

(2) 매퍼파일 작성시 #{xxx} 와 ${xxx} 의 사용

#{xxx} → 'xxx'
${xxx} → xxx

#{xxx}을 사용하면 자동으로 타입에 맞춰서 단일 인용부호가 붙어서 처리되므로 SQL 문에 사용되는 동적 데이터 값 설정에 적당하다. 그런데 만일 테이블명 또는 컬럼명을 동적으로 설정하려는 경우엔 자동으로 부여되는 단일 인용부호가 필요없으므로 이 때는 ${xxx}를 사용한다.

예1)

```
select name, schoolname, addr from ${tablename} where ${colname} = #{colvalue}
```

예2)

```
select id, name, writedate, memo from visitor where memo like '%'||#{key}||'%'
select id, name, writedate, memo from visitor where memo like '%${key}%'
```

☆ 주의할 사항은 ${xxx}의 경우에는 xxx 가 만드시 전달되는 객체의 프로퍼티명이거나 HashMap 객체의키값어야 한다. #{xxx}의 경우엔 객체의 프로퍼티명이거나 HashMap 객체의키값에서 없으면 아규먼트값을 추출하여 설정하지만 ${xxx}는 다르다.

(3) SELECT 명령의 수행 결과를 HashMap 객체로 받아오기

```
Map<키타입, XxxxVO> map =  session.selectMap("매퍼의태그명", "키로사용할프로퍼티명");
```

(4) INSERT 할 데이터를 HashMap 객체로 주기

```
<insert id="태그명"  parameterType="hashmap">

HashMap<String, String> map = new HashMap<>();
map.put("name","·····");
map.put("writedate", "·····");
map.put("memo", "·····");
session.insert(statement, map);
```

(5) 동적 SQL 명령 작성

메퍼파일에 SQL 명령을 작성할 때 조건에 따른 동적 구성이 가능하다. <if>, <choose>, <where>, <foreach> 태그 등을 사용할 수 있다.

```
<select id=" "
parameterType="YyyVO"
        resultType="XxxVO">
  SELECT * FROM 테이블명
  WHERE state = 'ACTIVE'
  <if test="title != null">
    AND title like #{title}
  </if>
</select>
```

```
<select id=" " resultType="XxxVO">
  SELECT * FROM 테이블명 WHERE state = 'ACTIVE'
  <choose>
    <when test="title != null">
      AND title like #{title}
    </when>
    <when test="author != null and content != null">
      AND author_name like #{author}
    </when>
    <otherwise>
      AND featured = 1
    </otherwise>
  </choose>
</select>
```

```
<select id="findActiveXxxVOLike"
     resultType="XxxVO">
  SELECT * FROM 테이블명
  WHERE
  <if test="state != null">
    state = #{state}
  </if>
  <if test="title != null">
    AND title like #{title}
  </if>
</select>
```

```
<select id="findActiveXxxVOLike" resultType="XxxVO">
  SELECT * FROM 테이블명
  <where>
    <if test="state != null">
        state = #{state}
    </if>
    <if test="title != null">
       AND title like #{title}
    </if>
  </where>
</select>
```

```
<select id=" " resultType="hashmap" parameterType="hashmap">
    SELECT name, age
    FROM 테이블명
    WHERE
        age = #{sUser_age} AND
        <foreach collection="sUser_type" item="type"
                    open="(" close=")" separator="or">
          user_type = #{type.value}
        </foreach>
</select>
```

```
List sUserTP = new ArrayList();
sUserTP.add("SP");
sUserTP.add("BX");

HashMap hm = new HashMap();
hm.put("sUser_age", 23) ;
hm.put("sUser_type", sUserTP)
```

```
   user_tyle IN
   <foreach collection="sUser_type" item="type" index="index"
        open="(" close=")" separator=",">
        #{type[index]}
   </foreach>
```

```
String[] sUserTP = {"SP", "BX"} ;

HashMap hm = new HashMap();
hm.put("sUser_age", 23) ;
hm.put("sUser_type", sUserTP) ;
```