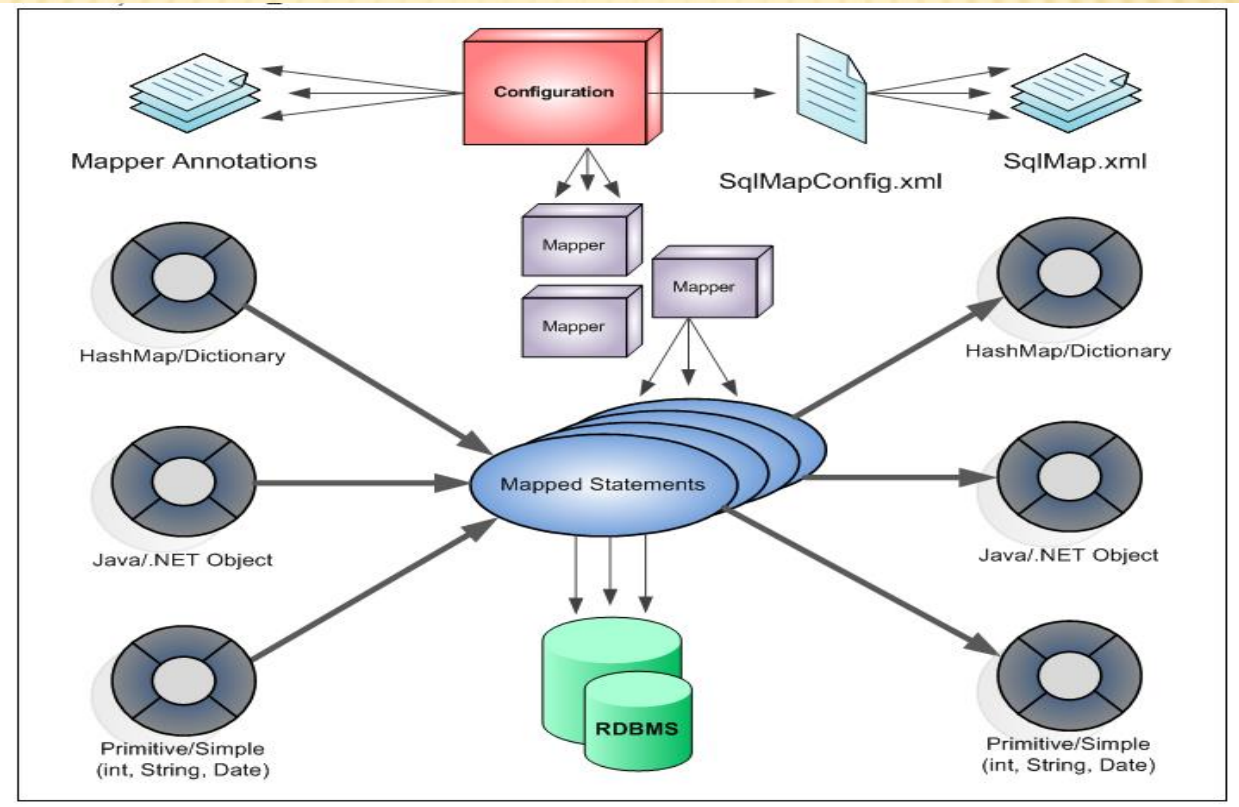
MYBATIS

백 성 애

MYBATIS란?

- MyBatis 는 개발자가 지정한 SQL, 저장프로시저 그리고 몇가지 고급 매핑을 지원하는 퍼시스턴스 프레임워크이다.
- http://blog.mybatis.org/
- http://mybatis.github.io/mybatis-3/



MYBATIS

- [1] Configuration 파일(mybatis-config.xml) :
- DB 설정과 트랜잭션 등 Mybatis가 동작하는 규칙을 정의.
  [2] 매퍼(Mapper) :
- SQL을 XML에 정의한 매퍼 XML 파일 또는 SQL을 인터페이스의 메소드마다 애노테이션으로 정의한 매퍼 인터페이스를 의미.
- [3] 매핑 구문(Mapped Statements) : 조회 결과를 자바 객체에 설정하는 규칙을 나타내는 결과매핑과 SQL을 XML에 정의한 매핑 구문을 의미.
- 사용자는 CRUD에 대한 각각의 SQL문은 Mapper XML 파일에 작성하고 이 파일들을 mybatis-Config XML 파일에 등록하면
- MyBatis API를 통해 자동으로 Mapping된 Statement 객체들을 생성하여 이를 통해 DB에 SQL문을 실행하게 된다.

IBATIS와 MYBATIS의 차이점

iBatis	MyBatis
Apache Project팀에서	Google code팀으로 이동하면서 명칭 변경
Jdk1.4이상 사용가능	Jdk1.5이상 사용 가능
[사용용어] SqlMapConfig=>	Configuration
sqlMap	Mapper
resultClass	resultType
네임스페이스 방식 변경 iBatis에선 네임스페이스가 **선택**이었으나 Ex) <sqlMap namespace="Board">	MyBatis에선 **네임스페이스**는 **필수 항목** <mapper namespace="com.sample.mapper.BoardMapper">

IBATIS와 MYBATIS의 차이점

4 iBatis	MyBatis	비고
com.ibatis.*	org.apache.ibatis.*	패키지 구조 변경
SqlMapConfig	Configration	용어변경
sqlMap	mapper	용어변경
sqlMapClient	sqlSession	구문대체
rowHandler	resultHandler	구문대체
resultHandler	SqlSessionFactory	구문대체
parameterMap, parameterClass	parameterType	속성 통합
resultClass	resultType	용어변경
#var#	#{var}	구문대체
var	${var}	구문대체
<isEqual> , <isNull>	<if>	구문대체

MYBATIS CONFIGURATION 파일

```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>

<properties resource="db.properties" />

<typeAliases>
    <typeAlias type="com.altibase.sidu.model.UserVo" alias="User" />
</typeAliases>

<!--  DB 연결 옵션 : UNPOOLED/POOLED/JNDI -->
<!--  transanctionManager 옵션 : JDBC/MANAGED -->
<environments default="development">
    <environment id="development">
    <transactionManager type="JDBC" />
    <dataSource type="POOLED">
        <property name="driver" value="${jdbc.driver}" />
        <property name="url" value="${jdbc.url}" />
        <property name="username" value="${jdbc.username}" />
        <property name="password" value="${jdbc.password}" />
        <property name="poolPingQuery" value="select 1 from dual"/>
            <property name="poolMaximumActiveConnections" value="100"/>
            <property name="poolMaximumIdleConnections" value="50"/>
            <property name="poolMaximumCheckoutTime" value="20000"/>
    </dataSource>
    </environment>
</environments>

<mappers>
    <mapper resource="com/altibase/sidu/mapper/UserMapper.xml" />
</mappers>

</configuration>
```

<properties> 태그에는 properties 파일의 경로 및 이름을 명시해주고,
<settings> 태그에는 Configuration을 제어할 property들을,

<transactionManager> 와 <dataSource>에는 연결할 DB 정보를 작성한다.

또, <mappers> 태그에는 미리 작성한 Mapper 파일들의 경로 및 이름을 작성한다.

MAPPER설정

- 매핑된 SQL문을 정의하는 곳이 매퍼 파일이다.
- 매퍼 위치 설정은 클래스패스에 상대적으로 리소스를 지정할 수도 있고, url 을 통해서 지정할 수 도 있다.

```
// Using classpath relative resources
<mappers>
  <mapper resource="org/mybatis/builder/AuthorMapper.xml"/>
  <mapper resource="org/mybatis/builder/BlogMapper.xml"/>
  <mapper resource="org/mybatis/builder/PostMapper.xml"/>
</mappers>

// Using url fully qualified paths
<mappers>
  <mapper url="file:///var/sqlmaps/AuthorMapper.xml"/>
  <mapper url="file:///var/sqlmaps/BlogMapper.xml"/>
  <mapper url="file:///var/sqlmaps/PostMapper.xml"/>
</mappers>
```

SQL MAP XML 파일

- **resultMap** – 데이터베이스 결과데이터를 객체에 로드하는 방법을 정의하는 요소
- **parameterMap** – 비권장됨! 예전에 파라미터를 매핑하기 위해 사용되었으나 현재는 사용하지 않음
- **sql** – 다른 구문에서 재사용하기 위한 SQL 조각
- **insert** – 매핑된 INSERT 구문
- **update** – 매핑된 UPDATE 구문.
- **delete** – 매핑된 DELEETE 구문.
- **select** – 매핑된 SELECT 구문.

SELECT 예제

```
<select id="selectPerson" parameterType="int" resultType="hashmap">
  SELECT * FROM PERSON WHERE ID = #{id}
</select>
```

DML구문 예제

Insert, update, delete 구문의 예제이다.

```
<insert id="insertAuthor" parameterType="domain.blog.Author">
  insert into Author (id,username,password,email,bio)
  values (#{id},#{username},#{password},#{email},#{bio})
</insert>

<update id="updateAuthor" parameterType="domain.blog.Author">
 update Author set
        username = #{username},
        password = #{password},
        email = #{email},
        bio = #{bio}
 where id = #{id}
</update>

<delete id="deleteAuthor" parameterType="int">
  delete from Author where id = #{id}
</delete>
```

INSERT시 자동생성키 사용 예

- insert 는 key 생성과 같은 기능을 위해 몇가지 추가 속성과 하위 요소를 가진다.
- 먼저, 사용하는 데이터베이스가 자동생성키(예를 들면, MySQL과 SQL 서버)를 지원한다면, useGeneratedKeys=”true” 로 설정하고 대상 프로퍼티에 keyProperty 를 셋팅할 수 있다.
- 예를 들어, Author 테이블이 id칼럼에 자동생성키를 적용했다고 하면, 구문은 아래와 같은 형태일 것이다.

- <insert id="insertAuthor" parameterType="domain.blog.Author"
- useGeneratedKeys=”true” keyProperty=”id”>
- insert into Author (username,password,email,bio)
- values (#{username},#{password},#{email},#{bio})
- </insert>

SELECTKEY 사용 예-랜덤 ID생성

- MyBatis 는 자동생성키 칼럼을 지원하지 않는 다른 데이터베이스를 위해 다른 방법 또한 제공한다.
- 이 예제는 랜덤 ID를 생성하고 있다.

```
<insert id="insertAuthor" parameterType="domain.blog.Author">
  <selectKey keyProperty="id" resultType="int" order="BEFORE">
    select CAST(RANDOM()*1000000 as INTEGER) a from SYSIBM.SYSDUMMY1
  </selectKey>
  insert into Author
          (id, username, password, email,bio, favourite_section)
  values
          (#{id}, #{username}, #{password}, #{email}, #{bio}, #{favouriteSection,jdbcType=VARCHAR}
  )
</insert>
```

SELECTKEY 사용 예-랜덤 ID생성

- 앞 예제에서, selectKey 구문이 먼저 실행되고, Author id프로퍼티에 셋팅된다. 그리고 나서 insert 구문이 실행된다. 이건 복잡한 자바코드 없이도 데이터베이스에 자동생성키의 행위와 비슷한 효과를 가지도록 해준다.
- selectKey 요소는 다음처럼 설정가능하다.
- <selectKey keyProperty="id" resultType="int"
- order="BEFORE" statementType="PREPARED">

SELECTKEY 속성

속성	설명
keyProperty	selectKey 구문의 결과가 셋팅될 대상 프로퍼티
resultType	결과의 타입. MyBatis 는 이 기능을 제거할 수 있지만 추가하는게 문제가 되지는 않을것이다. MyBatis 는 String 을 포함하여 키로 사용될 수 있는 간단한 타입을 허용한다.
order	BEFORE 또는 AFTER 를 셋팅할 수 있다. BEFORE 로 셋팅하면, 키를 먼저 조회하고 그 값을 keyProperty 에 셋팅한 뒤 insert 구문을 실행한다. AFTER 로 셋팅하면, insert 구문을 실행한 뒤 selectKey 구문을 실행한다. Oracle 과 같은 데이터베이스에서는 insert 구문 내부에서 일관된 호출 형태로 처리한다.
statementType	위 내용과 같다. MyBatis 는 Statement, PreparedStatement 그리고 CallableStatement 을 매핑하기 위해 STATEMENT, PREPARED 그리고 CALLABLE 구문타입을 지원한다.

SQL요소

- sql 요소는 다른 구문에서 재사용 가능한 SQL구문을 정의할 때 사용된다.
- <sql id="userColumns"> id,username,password </sql>
- 위 SQL 조각은 다른 구문에 포함시킬 수 있다.
- <select id="selectUsers" parameterType="int" resultType="hashmap">
- select <include refid="userColumns"/>
- from some_table
- where id = #{id} </select>

PARAMETER

```
<select id="selectUsers" parameterType="int" resultType="User">
        select id, username, password
        from users
        where id = #{id}
</select>
```

```
<insert id="insertUser" parameterType="User" >
        insert into users (id, username, password)
        values (#{id}, #{username}, #{password})
</insert>
```

- 만약 특정 칼럼에 null 이 전달되면, JDBC Type을 명시해야 한다. null 가능한 칼럼을 위해 필요

```
#{property,javaType=int,jdbcType=NUMERIC}
```

내장된 타입 별칭

- 공통의 자바타입에 대해서는 내장된 타입 별칭이 있다. 이 모두 대소문자를 가린다

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

별칭	매핑된 타입
float	Float
boolean	Boolean
date	Date
decimal	BigDecimal
bigdecimal	BigDecimal
object	Object
map	Map
hashmap	HashMap
list	List
arraylist	ArrayList
collection	Collection
iterator	Iterator

MYBATIS와 연동할 APPLICATION작성

- // mybatis-config.xml로부터 설정값을 읽어옴

```
InputStream    is =
Resources.getResourceAsStream("mybatis-config.xml");
```

- // 읽어온 xml 파일로부터 SqlSessionFactory를 생성함.
  `sqlSessionFactory = new SqlSessionFactoryBuilder()build(is);`
- // sqlSession 객체 생성. 이 때 인자값이 autoCommit 여부를 결정함.
- //true값을 주면 자동 커밋이며 false 값을 주게 되면 autocommit=false가 됨
- `sqlSession = sqlSessionFactory.openSession(false);`

동적 SQL

- MyBatis 의 가장 강력한 기능 중 하나는 동적 SQL 기능이다. 동적sql을 위해OGNL기반의 표현식을 사용함.
- if
- choose (when, otherwise)
- trim (where, set)
- foreach

IF 사용예

- <select id=”findActiveBlogWithTitleLike”
- parameterType=”Blog” resultType=”Blog”>
- SELECT * FROM BLOG
- WHERE state = ‘ACTIVE’
- <if test=”title != null”>
- AND title like #{title}
- </if>
- </select>

IF 사용예

```
<select id="findActiveBlogLike"
        parameterType="Blog" resultType="Blog">

  SELECT * FROM BLOG WHERE state = 'ACTIVE'
  <if test="title != null">
    AND title like #{title}
  </if>
  <if test="author != null and author.name != null">
    AND author_name like #{author.name}
  </if>
</select>
```

CHOOSE, WHEN, OTHERWISE

```
<select id="findActiveBlogLike"
        parameterType="Blog" resultType="Blog">
  SELECT * FROM BLOG WHERE state = 'ACTIVE'
  <choose>
    <when test="title != null">
      AND title like #{title}
    </when>
    <when test="author != null and author.name != null">
      AND author_name like #{author.name}
    </when>
    <otherwise>
      AND featured = 1
    </otherwise>
  </choose>
</select>
```

WHERE

```
<select id="findActiveBlogLike"
        parameterType="Blog" resultType="Blog">
  SELECT * FROM BLOG
  <where>
        <if test="state != null">
                state = #{state}
        </if>
        <if test="title != null">
                AND title like #{title}
        </if>
        <if test="author != null and author.name != null">
                AND author_name like #{author.name}
        </if>
  </where>
</select>
```

SET

```
<update id="updateAuthorIfNecessary"
        parameterType="domain.blog.Author">
  update Author
    <set>
      <if test="username != null">username=#{username},</if>
      <if test="password != null">password=#{password},</if>
      <if test="email != null">email=#{email},</if>
      <if test="bio != null">bio=#{bio}</if>
    </set>
  where id=#{id}
</update>
```

set 요소는 update 하고자 하는 칼럼을 동적으로 포함시키기 위해 사용될 수 있다.

set 요소는 동적으로 SET 키워드를 붙히고, 필요없는 콤마를 제거한다.

FOREACH

- 동적 SQL 에서 공통적으로 필요한 것은 collection 에 대해 반복처리를 하는 것이다.

```
<select id="selectPostIn" resultType="domain.blog.Post">
  SELECT *
  FROM POST P
  WHERE ID in
  <foreach item="item" index="index" collection="list"
    open="(" separator="," close=")">
      #{item}
  </foreach>
</select>
```