



一、select



```
<!-- 查询学生, 根据id -->
<select id="getStudent" parameterType="String" resultMap="studentResultMap">
    SELECT ST.STUDENT_ID,
           ST.STUDENT_NAME,
           ST.STUDENT_SEX,
           ST.STUDENT_BIRTHDAY,
           ST.CLASS_ID
    FROM STUDENT_TBL ST
    WHERE ST.STUDENT_ID = #{studentID}
</select>
```



这条语句就叫做'getStudent'，有一个String参数，并返回一个StudentEntity类型的对象。
注意参数的标识是：#{studentID}。


select 语句属性配置细节：

属性	描述	取值	默认
id	在这个模式下唯一的标识符，可被其它语句引用		
parameterType	传给此语句的参数的完整类名或别名		

属性	描述	取值	默认
resultType	语句返回值类型的整类名或别名。注意，如果是集合，那么这里填写的是集合的项的整类名或别名，而不是集合本身的类名。（resultType与resultMap不能并用）		
resultMap	引用的外部resultMap名。结果集映射是MyBatis中最强大的特性。许多复杂的映射都可以轻松解决。（resultType与resultMap不能并用）		
flushCache	如果设为true，则会在每次语句调用的时候就会清空缓存。select语句默认设为false	true/false	false
useCache	如果设为true，则语句的结果集将被缓存。select语句默认设为false	true/false	false
timeout	设置驱动器在抛出异常前等待回应的最长时间，默认为不设值，由驱动器自己决定	正整数	未设置
fetchSize	设置一个值后，驱动器会在结果集数目达到此数值后，激发返回，默认为不设值，由驱动器自己决定	正整数	驱动器决定
statementType	statement，preparedstatement，callablestatement。预准备语句、可调用语句	STATEMENT、PREPARED、CALLABLE	PREPARED
resultSetType	forward_only、scroll_sensitive、scroll_insensitive 只转发，滚动敏感，不区分大小写的滚动	FORWARD_ONLY、SCROLL_SENSITIVE、SCROLL_INSENSITIVE	驱动器决定

二、insert

一个简单的insert语句：



```
<!-- 插入学生 -->
<insert id="insertStudent" parameterType="StudentEntity">
    INSERT INTO STUDENT_TBL (STUDENT_ID,
```

```
        STUDENT_NAME,  
        STUDENT_SEX,  
        STUDENT_BIRTHDAY,  
        CLASS_ID)  
  
VALUES    ({studentID},  
          #{studentName},  
          #{studentSex},  
          #{studentBirthday},  
          #{classEntity.classID})  
  
</insert>
```



insert可以使用数据库支持的自动生成主键策略，设置useGeneratedKeys="true"，然后把keyProperty 设成对应的列，就搞定了。比如说上面的StudentEntity 使用auto-generated 为id 列生成主键。

```
<insert id="insertStudent" parameterType="StudentEntity" useGeneratedKeys="true"  
keyProperty="studentID">
```

推荐使用这种用法。


另外，还可以使用selectKey元素。下面例子，使用MySQL数据库nextval('student')为自定义函数，用来生成一个key。



```
<!-- 插入学生 自动主键-->  
<insert id="insertStudentAutoKey" parameterType="StudentEntity">  
    <selectKey keyProperty="studentID" resultType="String" order="BEFORE">  
        select nextval('student')  
    </selectKey>  
    INSERT INTO STUDENT_TBL (STUDENT_ID,  
                            STUDENT_NAME,  
                            STUDENT_SEX,  
                            STUDENT_BIRTHDAY,  
                            CLASS_ID)
```

```
VALUES    (#{studentID},
           #{studentName},
           #{studentSex},
           #{studentBirthday},
           #{classEntity.classID})

</insert>
```



insert语句属性配置细节：

属性	描述	取值	默认
id	在这个模式下唯一的标识符，可被其它语句引用		
parameterType	传给此语句的参数的完整类名或别名		
flushCache	如果设为true，则会在每次语句调用的时候就会清空缓存。select 语句默认设为false	true/false	false
useCache	如果设为true，则语句的结果集将被缓存。select 语句默认设为false	true/false	false
timeout	设置驱动器在抛出异常前等待回应的最长时间，默认为不设值，由驱动器自己决定	正整数	未设置
fetchSize	设置一个值后，驱动器会在结果集数目达到此数值后，激发返回，默认为不设值，由驱动器自己决定	正整数	驱动器决定
statementType	statement、preparedstatement、callablestatement。预准备语句、可调用语句	STATEMENT、PREPARED、CALLABLE	PREPARED
useGeneratedKeys	告诉MyBatis 使用JDBC 的getGeneratedKeys 方法来获取数据库自己生成的主键（MySQL、SQLSERVER 等关系型数据库会有自动生成的字段）。默认：false	true/false	false

属性	描述	取值	默认
keyProperty	标识一个将要被MyBatis设置进getGeneratedKeys的key 所返回的值，或者为insert 语句使用一个selectKey子元素。		

selectKey语句属性配置细节：

属性	描述	取值
keyProperty	selectKey 语句生成结果需要设置的属性。	
resultType	生成结果类型，MyBatis 允许使用基本的数据类型，包括String 、int类型。	
order	可以设成BEFORE 或者AFTER，如果设为BEFORE，那它会先选择主键，然后设置keyProperty，再执行insert语句；如果设为AFTER，它就先运行insert 语句再运行selectKey 语句，通常是insert 语句中内部调用数据库（像Oracle）内嵌的序列机制。	BEFORE/AFTER
statementType	像上面的那样，MyBatis 支持STATEMENT，PREPARED和CALLABLE 的语句形式，对应Statement，PreparedStatement 和CallableStatement 响应	STATEMENT、PREPARED、CALLABLE

批量插入

方法一：

```
<insert id="add" parameterType="EStudent">
  <foreach collection="list" item="item" index="index" separator=";">
    INSERT INTO TStudent(name,age) VALUES (#{item.name}, #{item.age})
  </foreach>
</insert>
```


上述方式相当语句逐条INSERT语句执行，将出现如下问题：

1. mapper接口的add方法返回值将是最一条INSERT语句的操作成功的记录数目（就是0或1），而不是所有INSERT


语句的操作成功的总记录数目

2. 当其中一条不成功时，不会进行整体回滚。

方法二：




```
<insert id="insertStudentAutoKey" parameterType="java.util.List">
    INSERT INTO STUDENT_TBL (STUDENT_NAME,
                             STUDENT_SEX,
                             STUDENT_BIRTHDAY,
                             CLASS_ID)
    VALUES
    <foreach collection="list" item="item" index="index" separator=",">
        ( #{item.studentName},#{item.studentSex},#{item.studentBirthday},#{item.classEntity.classID})
    </foreach>
</insert>
```



三、update

一个简单的update：



```
<!-- 更新学生信息 -->
<update id="updateStudent" parameterType="StudentEntity">
    UPDATE STUDENT_TBL
    SET STUDENT_TBL.STUDENT_NAME = #{studentName},
        STUDENT_TBL.STUDENT_SEX = #{studentSex},
        STUDENT_TBL.STUDENT_BIRTHDAY = #{studentBirthday},
        STUDENT_TBL.CLASS_ID = #{classEntity.classID}
    WHERE STUDENT_TBL.STUDENT_ID = #{studentID};
</update>
```



update语句属性配置细节：

属性	描述	取值	默认
id	在这个模式下唯一的标识符，可被其它语句引用		
parameterType	传给此语句的参数的完整类名或别名		
flushCache	如果设为true，则会在每次语句调用的时候就会清空缓存。select 语句默认设为false	true/false	false
useCache	如果设为true，则语句的结果集将被缓存。select 语句默认设为false	true/false	false
timeout	设置驱动器在抛出异常前等待回应的最长时间，默认为不设值，由驱动器自己决定	正整数	未设置
fetchSize	设置一个值后，驱动器会在结果集数目达到此数值后，激发返回，默认为不设值，由驱动器自己决定	正整数	驱动器决定
statementType	statement、preparedstatement、callablestatement。预准备语句、可调用语句	STATEMENT、PREPARED、CALLABLE	PREPARED

批量更新

情景一：更新多条记录为多个字段为不同的值
方法一：



```
<update id="updateBatch" parameterType="java.util.List">
  <foreach collection="list" item="item" index="index" open="" close="" separator=";">
    update course
    <set>
      name=${item.name}
    </set>
  </foreach>
  where id = ${item.id}
```

```
</foreach>
</update>
```



比较普通的写法，是通过循环，依次执行update语句。

方法二：



```
UPDATE TStudent SET Name = R.name, Age = R.age
from (
    SELECT 'Mary' as name, 12 as age, 42 as id
    union all
    select 'John' as name , 16 as age, 43 as id
) as r
where ID = R.id
```



情景二：更新多条记录的同一个字段为同一个值

```
<update id="updateOrders" parameterType="java.util.List">
    update orders set state = '0' where no in
    <foreach collection="list" item="id" open="(" separator="," close=")">
        #{id}
    </foreach>
</update>
```


四、delete

一个简单的delete：


```
<!-- 删除学生 -->
<delete id="deleteStudent" parameterType="StudentEntity">
    DELETE FROM STUDENT_TBL WHERE STUDENT_ID = #{studentID}
</delete>
```

delete语句属性配置细节同update

批量删除：



```
<!-- 通过主键集合批量删除记录 -->

<delete id="batchRemoveUserByPks" parameterType="java.util.List">

    DELETE FROM LD_USER WHERE ID in

    <foreach item="item" index="index" collection="list" open="(" separator="," close=")">

        #{item}


    </foreach>

</delete>
```



五、sql元素

Sql元素用来定义一个可以复用的SQL 语句段，供其它语句调用。比如：



```
<!-- 复用sql语句 查询student表所有字段 -->
<sql id="selectStudentAll">
    SELECT ST.STUDENT_ID,
```

```
        ST.STUDENT_NAME,  
        ST.STUDENT_SEX,  
        ST.STUDENT_BIRTHDAY,  
        ST.CLASS_ID  
FROM STUDENT_TBL ST
```

</sql>



这样，在select的语句中就可以直接引用使用了，将上面select语句改成：

```
<!-- 查询学生，根据id -->  
<select id="getStudent" parameterType="String" resultMap="studentResultMap">  
    <include refid="selectStudentAll"/>  
    WHERE ST.STUDENT_ID = #{studentID}  
</select>
```

六、 *parameters*

上面很多地方已经用到了参数，比如查询、修改、删除的条件，插入，修改的数据等，MyBatis可以使用Java的基本数据类型和Java的复杂数据类型。如：基本数据类型，String，int，date等。

但是使用基本数据类型，只能提供一个参数，所以需要使用Java实体类，或Map类型做参数类型。通过#{ }可以直接得到其属性。

1、基本类型参数

根据入学时间，检索学生列表：

```
<!-- 查询学生list，根据入学时间 -->  
<select id="getStudentListByDate" parameterType="Date" resultMap="studentResultMap">  
    SELECT *  
    FROM STUDENT_TBL ST LEFT JOIN CLASS_TBL CT ON ST.CLASS_ID = CT.CLASS_ID
```

```
WHERE CT.CLASS_YEAR = #{classYear};
</select>
```

```
List<StudentEntity> studentList = studentMapper.getStudentListByClassYear(StringUtil.parse("2007-9-1"));
for (StudentEntity entityTemp : studentList) {
    System.out.println(entityTemp.toString());
}
```

2、Java 实体类型参数

根据姓名和性别，检索学生列表。使用实体类做参数：

```
<!-- 查询学生list, like姓名、=性别, 参数entity类型 -->
<select id="getStudentListWhereEntity" parameterType="StudentEntity" resultMap="studentResultMap">
    SELECT * from STUDENT_TBL ST
        WHERE ST.STUDENT_NAME LIKE CONCAT(CONCAT('%', #{studentName}), '%')
        AND ST.STUDENT_SEX = #{studentSex}
</select>
```



```
StudentEntity entity = new StudentEntity();
entity.setStudentName("李");
entity.setStudentSex("男");
List<StudentEntity> studentList = studentMapper.getStudentListWhereEntity(entity);
for (StudentEntity entityTemp : studentList) {
    System.out.println(entityTemp.toString());
}
```



3、Map 参数

根据姓名和性别，检索学生列表。使用Map做参数：

```
<!-- 查询学生list, =性别, 参数map类型 -->
<select id="getStudentListWhereMap" parameterType="Map" resultMap="studentResultMap">
```

```
SELECT * from STUDENT_TBL ST
WHERE ST.STUDENT_SEX = #{sex}
AND ST.STUDENT_SEX = #{sex}
</select>
```



```
Map<String, String> map = new HashMap<String, String>();
map.put("sex", "女");
map.put("name", "雪");
List<StudentEntity> studentList = studentMapper.getListWhereMap(map);

for (StudentEntity entityTemp : studentList) {
    System.out.println(entityTemp.toString());
}
```



4、多参数的实现

如果想传入多个参数，则需要在接口的参数上添加@Param注解。给出一个实例：
接口写法：

```
public List<StudentEntity> getListWhereParam(@Param(value = "name") String name, @Param(value = "sex") String sex, @Param(value = "birthday") Date birthday, @Param(value = "classEntity") ClassEntity classEntity);
```

sql写法：



```
<!-- 查询学生list, like姓名、=性别、=生日、=班级，多参数方式 -->
<select id="getListWhereParam" resultMap="studentResultMap">
    SELECT * from STUDENT_TBL ST
    <where>
        <if test="name!=null and name!='' ">
            ST.STUDENT_NAME LIKE CONCAT(CONCAT('%', #{name}),'%')
        </if>
```

```
<if test="sex!= null and sex!= ' ' ">
    AND ST.STUDENT_SEX = #{sex}
</if>
<if test="birthday!=null">
    AND ST.STUDENT_BIRTHDAY = #{birthday}
</if>
<if test="classEntity!=null and classEntity.classID !=null and classEntity.classID!=' ' ">
    AND ST.CLASS_ID = #{classEntity.classID}
</if>
</where>
</select>
```



进行查询：

```
List<StudentEntity> studentList = studentMapper.getStudentListWhereParam("", "", StringUtil.parse("1985-05-28"), classMapper.getClassByID("20000002"));
for (StudentEntity entityTemp : studentList) {
    System.out.println(entityTemp.toString());
}
```

七、#{ }与\${ }的区别

默认情况下,使用#{ }语法,MyBatis会产生PreparedStatement语句中,并且安全的设置PreparedStatement参数,这个过程中MyBatis会进行必要的安全检查和转义。

示例1：

执行SQL：Select * from emp where name = #{employeeName}

参数：employeeName=>Smith

解析后执行的SQL：Select * from emp where name = ?

执行SQL : Select * from emp where name = \${employeeName}

参数 : employeeName传入值为 : Smith

解析后执行的SQL : Select * from emp where name =Smith

说明 :

1. #将传入的数据都当成一个字符串,会对自动传入的数据加一个双引号。如 : order by #{user_id} , 如果传入的值是111,那么解析成sql时的值为order by "111", 如果传入的值是id , 则解析成的sql为order by "id".

2. \$将传入的数据直接显示生成在sql中。如 : order by \${user_id} , 如果传入的值是111,那么解析成sql时的值为order by 111, 如果传入的值是id , 则解析成的sql为order by id.

综上所述,\${} 方式会引发SQL注入的问题、同时也会影响SQL语句的预编译,所以从安全性和性能的角度出发,能使用#{ }的情况下就不要使用\${ }。

\${ }在什么情况下使用呢 ?

有时候可能需要直接插入一个不做任何修改的字符串到SQL语句中。这时候应该使用\${ }语法。

比如,动态SQL中的字段名,如 : ORDER BY \${columnName}

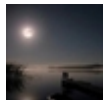
```
<select id="queryMetaList" resultType="Map" statementType="STATEMENT">
    Select * from emp where name = ${employeeName} ORDER BY ${columnName}
</select>
```

由于\${ } 仅仅是简单的取值,所以以前sql注入的方法适用此处,如果我们order by语句后用了\${ },那么不做任何处理的时候是存在sql注入危险的。

参考文章 : <http://limingnihao.iteye.com/blog/781911>
<http://www.tuicool.com/articles/zyUjqjI>
<http://blog.csdn.net/szwangdf/article/details/26714603>
http://blog.csdn.net/bear_wr/article/details/52386257

分类: [Mybatis](#)

[好文要顶](#)[关注我](#)[收藏该文](#)



风止雨歇
关注 - 1
粉丝 - 5
[+加关注](#)

« [上一篇](#) : [dataTables常用参数](#)
» [下一篇](#) : [SpringMVC处理静态资源](#)

posted on 2017-03-26 14:09 [风止雨歇](#) 阅读(10978) 评论(0) [编辑](#) [收藏](#)

0

0