1、基本的增删改

```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="daiwei.test.myInterface.UserMapper">
    <select id="getUserById" resultType="daiwei.test.pojo.User">
        select * from user where id = #{id}
    </select>

    <!-- public void insertUserById(User user) -->
    <insert id="insertUserById" parameterType="daiwei.test.pojo.User">
        INSERT INTO `user` (NAME, age, sex)
        VALUES
            (#{name}, #{age}, #{sex})
    </insert>

    <!-- public void updateUserById(User user); -->
    <update id="updateUserById" parameterType="daiwei.test.pojo.User">
        UPDATE `user`
            SET `name` = #{name},
                `age` = #{age},
                `sex` = #{sex}
            WHERE
                `id` = #{id}
    </update>

    <!-- public boolean deleteUserById(Integer id); -->
    <delete id="deleteUserById">
        DELETE
        FROM
            `user`
        WHERE
            `id` = #{id}
    </delete>
</mapper>
```

2、insert获取自增主键的值

1、支持自增主键的数据库，通过使用GenerateKey的方式获取自增主键 具体配置方法如下

```
<insert id="insertUserById" parameterType="daiwei.test.pojo.User"
useGeneratedKeys="true" keyProperty="id">
```

useGenerateKeys：是否要开启使用GenerateKey的方式获得主键（默认为false);
keyProperty：　把查出的主键值封装到pojo的哪个属性。

2、不支持自增主键的数据库通过selectKey的方法获得主键，如oracle，通过先执行序列sql，然后在执行插入sql，具体配置方法如下

```
<insert id="insertUserById" databaseId="oracle">
        <!--
            keyProperty      查出的主键值封装到pojo的哪个属性;
            order:BEFORE     当前sql在插入sql之前执行还是之后执行;
            resultTpye       查出值得返回类型;
         -->
         <selectKey keyProperty="id" order="BEFORE" resultType="integer">
            <!-- 此处编写查主键的key -->
         </selectKey>
         <!-- 此处编写要插入的sql语句 -->
    </insert>
```

keyProperty：　查出的主键值封装到pojo的哪个属性;
order : BEFORE　当前SQL在插入SQL之前执行还是之后执行;
resultType：　查出的主键值得类型;

3、参数处理

1、单个参数 ： mybatis不会做特殊处理
　　#{参数名}；取出参数

2、多个参数：mybatis会做特殊处理
　　多个参数会被封装成一个map
　　　　key：param1、param2.....paramN或者参数的索引也可以
　　　　value：传入的参数值

#{}就是从map中取出指定的key值

异常：

org.apache.ibatis.binding.BingingExecption：Paramter 'id' not found.
Available paramter are [1, 0, param1, param2]

3、命名参数：明确指定封装参数时map的key：@Param("id")

多个参数会被封装成一个map

key： 使用@Param注解指定的值

value：参数值

#{指定的key}：取出指定的参数值;

4、POJO：

如果传入的多个参数正好是业务逻辑的数据模型， 我们就直接传入pojo；

#{属性名}：取出传入的pojo属性值

5、Map：

如果多个参数不是业务模型中的数据，没有对应的pojo，不经常使用，为了方便 也可以传入map

#{key}：取出map中对应的值。

6、如果多个数据不是业务模型中的数据，但是要经常使用，推荐来编写一个TO（Transfer Object）数据传输对象

思考：

```
1  public User getUserById(@Param("id")Integer id, String name);
```

取值： id ==>#{id} name ==> #{param2}

```
1  public User getUserById(Integer id, @Param("u")User user);
```

取值： id ==> #{param1} name ==> #{u.name / param2.name}

```
public User getUserById(List<Integer> ids);
```

取值：　第一个id的值 ==> #{list[0}

特别注意：如果是Collection（List、Set）或者是数组，也会特殊处理，也是把传入的list或者数组封装在map中

key：Collection　key　==>　collection(如果是List还可以使用key ==> list)

数组　key ==> array

4、select用List封装查询结果

用List来封装结果，List中的每个元素都是对应的一条数据的pojo

usermapper接口

```
public List<User> getUserListByName(String name);
```

mapper.xml

```
<!-- public List<User> getUserListByName(String name); -->
    <select id="getUserListByName" resultType="daiwei.test.pojo.User">
        select * from user where name = #{name}
    </select>
```

5、select用map封装查询结果

使用map来封装select返回结果集（一般用于单独要查询出的结果集中属性联系分散，且没有专用的pojo对其进行封装）

- 当只要返回一条记录集且中间包含多个属性（map的key为属性名[String]，value 为属性的属性的值[Object] ），以下是代码示例：

usermapper接口

```
public Map<String, Object> getUserMapByLike(Integer id);
```

mapper.xml（返回类型为map）

```
<select id="getUserMapByLike" resultType="map">
        select * from user where id = #{id}
    </select>
```

- 当返回结果集是多条记录且每个value中又包含多个属性，在这种情况下需要一个对象来包装map中的value，则key[object为key字段的类型] value[pojo包装数据类型]

userMapper接口（注解mapkey指定，指定属性封装在map的key中）

```
@MapKey("id")
public Map<String, User> getUsersMapByName(String name);
```

mapper.xml（resultType为包装在map value中的pojo）

```
<select id="getUsersMapByName" resultType="daiwei.test.pojo.User">
        select * from user where name = #{name}
    </select>
```

```

```