MyBatis:

MyBatis 是一款优秀的持久层框架，它支持定制化 SQL、存储过程以及高级映射。MyBatis 避免了几乎所有的 JDBC 代码和手动设置参数以及获取结果集。MyBatis 可以使用简单的 XML 或注解来配置和映射原生信息，将接口和 Java 的 POJOs(Plain Old Java Objects,普通的 Java对象)映射成数据库中的记录

搭建MyBatis环境,导包,在xml配置,导入了数据库的值.映射关系,数据库,dao层接口,对应xml的实现方法pojo,如果表关系和对象对应不上,比如有前缀:g_girl要写,resultMap动态sql

学习过程:

1.搭建MyBatis环境

a.导包

```
<!-- https://mvnrepository.com/artifact/org.mybatis/mybatis -->
<dependency>
 <groupId>org.mybatis</groupId>
 <artifactId>mybatis</artifactId>
 <version>3.5.0</version>
</dependency>
<!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->
<dependency>
 <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <version>5.1.38</version>
 </dependency>
 <dependency>
```

b.在resource中配置xml文件,如:mybatis-cofig.注意映射关系和jdbc.properties文件

```
<?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">
```

```
5
6 <configuration>
7  <!--导入properties文件-->
8  <properties resource="jdbc.properties"/>
9
10  <!--数据库开发的环境-->
11  <environments default="a">
12  <!--开发环境-->
13
14  <environment id="a">
15  <!--开启事务管理-->
16  <transactionManager type="JDBC"/>
17  <!--配置数据源,不使用数据连接池-->
18  <dataSource type="UNPOOLED">
19  <property name="driver" value="${jdbc.driver}"/>
20  <property name="url" value="${jdbc.url}"/>
21  <property name="username" value="${jdbc.username}"/>
22  <property name="password" value="${jdbc.password}"/>
23  </dataSource>
24  </environment>
25  <!--生产环境-->
26  <environment id="b">
27  <transactionManager type=""></transactionManager>
28  <dataSource type=""></dataSource>
29  </environment>
30  </environments>
31
32  <!--映射关系:数据库中的表和对象的映射关系-->
33  <mappers>
34  <mapper resource="com/lanou/dao/UserDao.xml"/>
35  <mapper resource="com/lanou/dao/GirlDao.xml"/>
36  </mappers>
37
38 </configuration>
```

c.dbc.properties文件

```
1 jdbc.driver=com.mysql.jdbc.Driver
2 jdbc.url=jdbc:mysql://localhost:3306/db1227
```

```
jdbc.username=root
jdbc.password=root
```

d.映射关系:数据库中的表和对象的映射关系

```
<mappers>
 <mapper resource="com/lanou/dao/UserDao.xml"/>
 <mapper resource="com/lanou/dao/GirlDao.xml"/>
 </mappers>
```

2.数据库dao层接口.Mybatis的实现接口写在xml文件中,方式:在resource中新建com.lanou.dao,注意:此时一定要注意包路径,如:可以新建com包,一层一层添加包,以防Dao层接口找不到实现的xml

a数据库dao层接口,如下:

```
List<User> selectAll();
User selectByID(int id);
//删除
int delete(int id);
//添加
int add(@Param("username") String username,@Param("password")String passw
ord);
//修改
int modify(int id);
```

b.resource中的xml针对user的文件:(注意:1.select中和数据库操作保持一致,增删改,简单的保持一致.2.当表传的参数不止一个时,三种方式:arg,下标以0开始,param下标以1开始,但是我们最常用第三种方式,在多个参数前加@param)

```
int add(@Param("username") String username,@Param("password")String passw
ord);
```

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://myba
tis.org/dtd/mybatis-3-mapper.dtd" >
<!--namespace:xml对应的接口-->
<mapper namespace="com.lanou.dao.UserDao">

 <!--映射文件里-->

```

```
<!--
select:和做数据库操作保持一致
 id:和dao层中接口的方法保持一致
 resultType:一条数据结果以什么类型展示
 -->
 <select id="selectAll" resultType="com.lanou.pojo.User">
 select * from user;
 </select>
 <select id="selectByID" resultType="com.lanou.pojo.User">
 select * from user where id = #{id};
 </select>
 <delete id="delete">
 delete from user where id = #{id};
 </delete>
 <!--<insert id="add">-->
 <!--insert into user (username, password) values(#{arg0},#{arg1})-->
 <!--</insert>-->

 <insert id="add">
 insert into user (username, password) values(username,password)
 </insert>
</mapper>
```

c.特殊情况:如果表关系和对象对应不上,比如有前缀:g_girl,要写resultMap,自定义映射关系.配置如下:

```
List<Girl> selectAll();

int insert(@Param("name") String name, @Param("age") int age);

int insert1(Girl girl);

int update1(@Param("id") int id,@Param("name") String name);
int update2(@Param("id") int id,@Param("age") int age);
int update3(@Param("id") int id,@Param("name") String name,@Param("age") int age);

 int update4(Girl girl);

```

```
List<Girl> selectAll1(Girl girl);
```

```
<?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" >
<!--namespace:xml对应的接口-->
<mapper namespace="com.lanou.dao.GirlDao">

 <!--<select id="selectAll" resultType="com.lanou.pojo.Girl">-->
 <!--select * from sms_girl;-->
 <!--</select>-->

  <!--<select id="selectAll" resultType="com.lanou.pojo.Girl">-->
  <!--select g_id id,g_name name,g_age age from sms_girl girl ;-->
  <!--</select>-->
```

```
<!--表中的映射-->
<!--属性和字段相匹配-->
<resultMap id="girlMap" type="com.lanou.pojo.Girl">
<id property="id" column="g_id"/>
<result property="name" column="g_name"/>
<result property="age" column="g_age"/>
</resultMap>
<!--
resultType:一条数据最终形成的数据类型
 resultMap:当数据类型和表中字段不一致时,自定义映射关系
 -->
 <select id="selectAll" resultMap="girlMap">
 select * from sms_girl;
 </select>
```

d.当添加或者查询的条件字段不一致时,比如:只通过id添加女孩的名字不添加年龄,或者只添加年龄不添加女孩的名字,或者都添加.我们这个时候可以写set标签,通过条件添加字段.

查询时,只有女孩的名字,怎样查,只有id,怎么查.可以使用,为where和if来写

```
<insert id="insert">
```

```
insert into sms_girl(g_name,g_age) values (#{name},#{age});
</insert>
<insert id="insert1">
insert into sms_girl(g_name,g_age) values (#{name},#{age});
</insert>

<update id="update1">
update sms_girl set g_name=#{name} where g_id=#{id};
 </update>

 <update id="update2">
 update sms_girl set g_age=#{age} where g_id=#{id};
 </update>

 <update id="update3">
 update sms_girl set g_name=#{name},g_age=#{age} where g_id=#{id};
 </update>

 <!--
 set:
 set配合if动态进行update的操作
 1.如果set中没有成立的if,SQL语句中会省略set
 2.如果set中有成立的if,会去掉有多余的逗号
 -->
 <update id="update4">
 update sms_girl
 <set>
 <if test="name!=null">
 g_name=#{name},
 </if>
 <if test="age!=null">
 g_age=#{age}
 </if>
 </set>
 where g_id=#{id};
 </update>

 <select id="selectAll1" resultMap="girlMap">
```

```
40  select * from sms_girl
41  <where>
42  <if test="name!=null">
43  g_name=#{name}
44  </if>
45  <if test="age!=null">
46  AND g_age=#{age}
47  </if>
48  </where>
49  </select>
50 </mapper>
```

测试方法时,通过XML配置文件,创建sqlSessionFactory,根据需求返回数据库的操作

```
1  public class MybatisTest {
2
3
4   private SqlSessionFactory sqlSessionFactory;
5
6   @BeforeEach
7   void test() throws IOException {
8   //通过XML配置文件,创建sqlSessionFactory
9   InputStream resourceAsStream = Resources.getResourceAsStream("mybatis-co
nfig.xml");
10   sqlSessionFactory=new
SqlSessionFactoryBuilder().build(resourceAsStream);
11   System.out.println(sqlSessionFactory);
12   }
13
14   @Test
15   void test1() {
16   SqlSession sqlSession = sqlSessionFactory.openSession();
17   System.out.println(sqlSession);
18   List<User> userList = sqlSession.selectList("selectAll");
19   System.out.println(userList);
20   for (User user : userList) {
21   System.out.println(user);
```

```
22  }
23  }
24
25  @Test
26  void test2() {
27  SqlSession sqlSession = sqlSessionFactory.openSession();
28 // 省略了实现的部分
29  UserDao userDao = sqlSession.getMapper(UserDao.class);
30  User user = userDao.selectByID(1);
31  System.out.println(user);
32
33  // sqlSession.selectOne("selectByID",new )
34  }
35
36  @Test
37  void test3() {
38  SqlSession sqlSession = sqlSessionFactory.openSession();
39  UserDao userDao = sqlSession.getMapper(UserDao.class);
40  int row = userDao.delete(1);
41  System.out.println(row);
42  }
43
44  @Test
45  void test4() {
46  //openSession获取数据库链接
47  SqlSession sqlSession = sqlSessionFactory.openSession();
48  UserDao userDao = sqlSession.getMapper(UserDao.class);
49  int row = userDao.add("123", "qqq");
50  System.out.println(row);
51  }
52
53  @Test
54  void test5() {
55  SqlSession sqlSession = sqlSessionFactory.openSession();
56  GirlDao girlDao = sqlSession.getMapper(GirlDao.class);
57  List<Girl> list = girlDao.selectAll();
58  for (Girl girl : list) {
59  System.out.println(girl);
```

```
60  }
61  }
62
63  @Test
64  void test6() {
65  SqlSession sqlSession = sqlSessionFactory.openSession();
66  GirlDao girlDao = sqlSession.getMapper(GirlDao.class);
67  int row = girlDao.insert("如花", 15);
68  System.out.println(row);
69  }
70
71  @Test
72  void test7() {
73  SqlSession sqlSession = sqlSessionFactory.openSession();
74  GirlDao girlDao = sqlSession.getMapper(GirlDao.class);
75  Girl girl=new Girl();
76  girl.setName("如花呀");
77  girl.setAge(19);
78  int row = girlDao.insert1(girl);
79  System.out.println(row);
80
81  }
82
83  @Test
84  void test8() {
85  SqlSession sqlSession = sqlSessionFactory.openSession();
86  GirlDao girlDao = sqlSession.getMapper(GirlDao.class);
87  int row = girlDao.update1(3, "周杰伦");
88  System.out.println(row);
89  }
90
91  @Test
92  void test9() {
93  SqlSession sqlSession = sqlSessionFactory.openSession();
94  GirlDao girlDao = sqlSession.getMapper(GirlDao.class);
95  Girl girl=new Girl();
96  girl.setName("安妮");
97  girl.setAge(19);
```

```
98  girl.setId(2);
99  int row = girlDao.update4(girl);
100  System.out.println(row);
101  }
102
103  @Test
104  void test10() {
105  SqlSession sqlSession = sqlSessionFactory.openSession();
106  GirlDao girlDao = sqlSession.getMapper(GirlDao.class);
107  Girl girl=new Girl();
108  girl.getAge();
109  girl.getName();
110  girl.getId();
111  List<Girl> list = girlDao.selectAll1(girl);
112  for (Girl girl1 : list) {
113  System.out.println(girl1);
114  }
115  }
```

摘自:http://www.mybatis.org/mybatis-3/zh/configuration.html

这是Mybatis的中文官网.