MyBatis

一、概念

MyBatis SQL Mapper Framework for Java （sql映射框架）

1. **sql mapper** : sql映射

   可以把数据库表中的一行数据，映射为一个Java对象。操作这个对象，就相当于操作表中的数据。

2. **Data Access Object**（DAO）: 数据访问，对数据库执行增删改查。

MyBatis提供了哪些功能呢?

1. 提供了创建Collection，Statement, ResultSet的能力，不用开发人员创建
2. 提供了执行sql语句的能力，不用开发人员执行sql
3. 提供了循环sql,把sql的结果转为Java对象，List集合的能力。

```
while(rs.next()){
    Stduent std = new Stduent();
    std.setId(rs.getInt("id"));
    std.setName(rs.getString("name"));
    stuList.add(std);
}
```

4. 提供了关闭资源的能力，不用关闭Connection

开发人员提供sql语句---》MyBatis处理sql---》开发人员得到List集合或java对象

二、入门搭建

实现步骤

1. 新建的student表
2. 加入maven的mybatis坐标，MySQL驱动的坐标

```
  <dependencies>
<!--    mybatis的依赖-->
    <dependency>
      <groupId>org.mybatis</groupId>
      <artifactId>mybatis</artifactId>
      <version>3.5.1</version>
    </dependency>

    <dependency>
      <groupId>mysql</groupId>
      <artifactId>mysql-connector-java</artifactId>
      <version>8.0.13</version>
    </dependency>
  </dependencies>
```

3. 创建实体类，Student--保存表中的一行数据

```
public class Student {
    private Integer id;
    private String name;
    private String email;
    private Integer age;

    public Student() {
    }
    。。。。
```

4. 创建持久层的dao接口，定义操作数据库的方法

```
package com.qiangliu8.dao;

import com.qiangliu8.domain.Student;

import java.util.List;

public class StudentDaoImpl implements StudentDao {

    @Override
    public List<Student> selectStudents() {
        return null;
    }
}
```

5. 创建一个mybatis使用的配置文件：sql映射文件 ，写sql语句，一般一个表一个sql映射文件，xml文件

```
<?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="com.qiangliu8.dao.StudentDao">
    <select id="selectStudents"
resultType="com.qiangliu8.domain.Student">
        select * from student ORDER BY id
    </select>
</mapper>

<!--
指定约束文件，限制和检查在当前文件中出现的标签，属性必须符合mybatis的要求
<!DOCTYPE mapper
        PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
        "http://mybatis.org/dtd/mybatis-3-mapper.dtd">

mapper是文件的跟标签，必须的
namespace叫做命名空间，唯一值的，可以是自定义的字符串。要求你使用的dao接口的权限的名称。

在当前文件中，可以使用特定的变迁，表示数据库的特定操作。
    <select>表示执行查询
        id是sql语句的唯一标识，可以自定义。要求使用方法的名称.
        resultType表示结果欸写的。是sql语句执行后得到的Result,遍历这个ResultSet得到java对象的类型。
    <update>表示更新数据库的操作，就是在<update>标签中，写的是update sql语句
    <insert>表示插入，放的是insert语句
    <delete>表示删除，执行的是delete语句
-->
```

6. 创建mybatis的主配置文件

```
<!--mybatis的主配置文件，主要定义了数据库的配置信息，sql映射文件的位置-->
<?xml version="1.0" encoding="UTF-8" ?>
<!--    1、约束文件-->
<!DOCTYPE configuration
        PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
        "http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>
<!--    环境配置：数据库的连接信息
        default:必须和某个environment的id值一样
        噶苏mybatis使用哪个数据库的连接信息，也就是访问哪个数据库
-->
<!--    settings:控制mybatis全局行为-->
<settings>
    <!--        设置mybatis输出日志-->
    <setting name="logImpl" value="STDOUT_LOGGING"/>
</settings>
    <environments default="development">
<!--        environment :一个数据库信息的配置，环境id:一个唯一值，自定义，表示环境的名称。-->
        <environment id="development">
<!--            transactionManager：mybatis的事务类型-->
            <transactionManager type="JDBC"/>
<!--            dataSource表示数据源，连接数据库的，-->
<!--            type表示数据源的类型，POOLED表示使用连接池-->
```

```
            <dataSource type="POOLED">
                <property name="driver"
value="com.mysql.cj.jdbc.Driver"/>
                <property name="url"
value="jdbc:mysql://localhost:3306/springdb"/>
                <property name="username" value="root"/>
                <property name="password" value="Lq060528"/>
            </dataSource>
        </environment>
    </environments>

<!--    sql mapper（sql映射文件的位置）-->
    <mappers>
<!--        一个mapper指定一个文件的位置-->
<!--        从类路径开始的路径信息 target/classess(类路径) -->
        <mapper resource="com/qiangliu8/dao/StduentDao.xml"/>
    </mappers>
</configuration>
```

7. 创建使用mybatis类，通过mybatis访问数据库

```
public class MyApp {
    public static void main(String[] args) throws IOException {
//      访问mybatis读取student数据
//      1.定义mybatis著配置文件的名称，从类数据的根开始（target/classess）
        String config= "mybatis.xml";
//      2.读取这个config表示的文件
        InputStream inputStream = Resources.getResourceAsStream(config);
//      3.创建了SqlSessionFactoryBuilder对象
        SqlSessionFactoryBuilder sqlSessionFactoryBuilder = new
SqlSessionFactoryBuilder();
//      4.创建SqlSessionFactory对象
        SqlSessionFactory sqlSessionFactory =
sqlSessionFactoryBuilder.build(inputStream);
//      5.获取SqlSession对象，从SqlSessionFactory中获取SqlSession
        SqlSession sqlSession = sqlSessionFactory.openSession();
//      6.指定要执行的sql语句的表示。sql映射文件中的namespace+“。”+标签的id值
        String sqlId = "com.qiangliu8.dao.StudentDao.selectStudents";
//      7.执行sql语句，通过sqlId找到语句
        List<Student> studentList = sqlSession.selectList(sqlId);
//      8.输出结果
        studentList.forEach(stu -> System.out.println(stu));
//      9.关闭sqlSession对象
        sqlSession.close();
    }
}
```

提交操作

```
<select id="insertStudents">
    insert into student values (#{id},#{name},#{email},#{age})
</select>
```

```
String sqlId = "com.qiangliu8.dao.StudentDao.insertStudents";
Student student = new Student(1003,"吴云","1102644615@qq.com",24);
//      7.执行sql语句，通过sqlId找到语句
int i = sqlSession.insert(sqlId,student);
//mybatis默认不是自动提交事务的，所以在insert , update , delete后要手工提交事务
sqlSession.commit();
```

三、类的介绍

Resources	mybatis中的一个类，负责读取主配置文件	Resources in = Resources.getResourceAsStream("mybatis.xml");
SqlSessionFactoryBuilder	创建SqlSessionFactory对象	SqlSessionFactoryBuilder builder= new SqlSessionFactoryBuilder();
SqlSessionFactory	重量级对象，程序创建一个对象耗时比较长，使用资源比较多	接口，接口实现列DefaultSessionFactory。获取sqlSession对象

1. **Resources**: mybatis中的一个类，负责读取主配置文件

   ```
   Inputstream in = Resources.getResourceAsstream ( "mybatis.xml" );
   ```

2. **sqlsessionFactoryBuilder** :创建sqlsessionFactory对象，

   ```
   sqlsessionFactoryBuilder builder = new sqlsessionractoryBuilder();
   //创建sqlsessionractory对象
   sqlsessionFactory factory = builder.build(in) ;
   ```

3. **sqlsessionFactory** :重量级对象，程序创建一个对象耗时比较长，使用资源比较多。在整个项目中，有一个就够用了。

   sqlSessionFactory:接口，接口实现类:DefaultsqlsessionFactory

   sqlSessionFactory作用:获取sqlSession对象。

   ```
   SqlSession sqlSession = factory.openSession() ;
   ```

   openSession()方法说明：

   1. opensession() :无参数的，获取是非自动提交事务的sqlSession对象
   2. opensession (boolean): opensession(true)获取自动提交事务的sqlSession.
      opensession (false)：非自动提交事务的sqlSession

4. **SqlSession**：

   SqlSession接口：定义了操作数据的方法，例如selectOne(),selectList(),insert()....

   SqlSession接口实现类：DefaultSqlSession

SqlSession对象不是线程安全的，需要在方法内部使用，在执行sql语句之前，使用openSession()获取SqlSession.在执行完sql语句之后，需要关闭close（），才能保证线程安全。

三、动态代理

使用动态代理的条件分析

1. dao对象，类型是studentDao，全限定名称是:com.qiangliu8.dao.studentDao全限定名称和namespace是一样的。
2. 方法名称，selectstudents，这个方法就是 mapper文件中的 id值 selectstudents
3. 通过dao中方法的返回值也可以确定MyBatis,要调用的sqlsession的方法
   如果返回值是list，调用的是SqLSession. seLectList()方法。
   如果返回值 int,或足非ist的，看mapper文件中的标签是, 就会调用sqLsession的insert，update等方法

mybatis的动态代理:mybatis根据 dao的方法调用，获取执行sql语句的信息。mybatis根据你的dao接口，创建出一个dao接口的实现类，并创建这个类的对象。完成SqLsession调用方法，访问数据库。

```
//普通方式
@Override
public List<Student> selectStudent() {
    SqlSession sqlSession = MyBatisUtils.getSqlSession();
    String sqlId = "com.qiangliu8.dao.StudentDao.selectStudents";
    List<Student> studentList = sqlSession.selectList(sqlId);
    sqlSession.close();
    return studentList;
}
//代理方式
@Test
public void testSelectStudents(){
    //使用mybatis动态代理机制，使用SqlSession.getmapper(接口)
    //getMapper能获取dao接口对于的实现类对象
    SqlSession sqlSession = MyBatisUtils.getSqlSession();
    StudentDao studentDao = sqlSession.getMapper(StudentDao.class);
    //调用dao的方法，执行数据库操作
    List<Student> studentList = studentDao.selectStudents();
    System.out.println(studentList);
}
```

深入理解参数

动态代理:使用SqlSession.getMapper(dao接口.class)获取这个dao接口的对象

传入参数：从Java代码中把数据传入到mapper文件的sql语句钟

1. **paramterType：**

   写在mapper文件中的一个属性。表示dao接口中放的参数的数据类型

   例如：StudentDao接口

   paramterType：dao接口中方法参数的数据类型

   它的值是Java的数据类型全限定名称或者是mybatis定义的别名

   parameterType="java.lang.Integer" 或者parameterType="int"

```
<select id="getStudentById" parameterType="java.lang.Integer"
resultType="com.qiangliu8.domain.Student">
 select * from student where id = #{id}
</select>
```

   注意: parameterType不是强制的，mybatis通过反射机制能够发现接口参数的数类型。所以可以没有。一般我们也不写。

2. **一个简单类型**

   简单类型:mybatis把java的基本数据类型和string都叫简单类型。

   在mapper文件获取简单类型的一个参数的值，使用#{任意字符}

```
<select id="getStudentById" parameterType="java.lang.Integer"
resultType="com.qiangliu8.domain.Student">
    select * from student where id = #{studentId}
</select>
```

   使用#{}之后，mybatis执行sql是使用的jdbc中的PreparedStatement对象

   由mybatis执行下面的代码：

   1. mybatis创建Connection,PreparedStatement对象

```
String sql = "select id,name,email,age from student where id =
?";
PreparedStatement pst = conn.preparedStatement(sql);
pst.setInt(1,1001);
```

   2. 执行sql封装为resultType ="com.qiangliu8.domian.Student"这个对象

```
ResultSet rs = pst.executeQuery();
while(rs.next()){
//从数据路取表的一行数据，存到一个java对象属性中
Student stu = new Student()
stu.setName(rs.getString("name"));
}
return student;// 给了dao方法调用的返回值
```

3. **多个参数，使用@Param命名参数**

   1. 接口 public List selectMulitParam(@Param("myname")String name,@Param("myage")Integer age);

      使用@Param("参数名") String name

      Dao文件：

```
//多个参数：命名参数，在形参定义的前面加入@Param("自定义参数名称")
public List<Student> selectMulitParam(@Param("myName") String
name, @Param("myAge") Integer age);
```

mapper文件

```
<select>
    select * from student where name=#{myName} or age=#{myAge}
</select>
```

Test文件

```
@Test
public void testSelectMulitParam() throws IOException {
    SqlSession sqlSession = MyBatisUtils.getSqlSession();
    StudentDao studentDao =
sqlSession.getMapper(StudentDao.class);
    List<Student> studentList = studentDao.selectMulitParam("刘
强",24);
    sqlSession.close();
    System.out.println(studentList);
}
```

4. **多个参数，使用对象传参**

传入对象

```
public class QueryParam {
    private String paramName;
    private Integer paramAge;
}
```

StudentDao接口

```
//多个参数：使用Java对象作为接口中方法的参数
List<Student> selectMultiObject(QueryParam param);
```

mapper方法

```
<!--    多个参数：使用Java对象的属性值，作为参数实际值
        使用对象语法：#{属性名,javaType=类型名称， jdbcType=数据类型}很少用。
        javaType:指java中的属性数据类型。
        jdbcType:在数据库中的数据类型。
        例如:#{paramName,javaType=java.lang ,string,jdbcType=VARCHAR}
        简化方式：#{属性名}
-->
    <select id="selectMultiObject"
resultType="com.qiangliu8.domain.Student">
        select * from student where name=#{paramName} or age = #
{paramAge}
    </select>
```

测试方法

```
@Test
public void testSelectMultiObject() throws IOException {
    SqlSession sqlSession = MyBatisUtils.getSqlSession();
    StudentDao studentDao = sqlSession.getMapper(StudentDao.class);
    QueryParam queryParam = new QueryParam("刘强",18);
    List<Student> studentList = studentDao.selectMultiObject(queryParam);
    System.out.println(studentList);
}
```

5. **按位置传参**

多个参数-简单类型的，按位置传值，

mybatis.3.4之前，使用#{0},#{1]
mybatis。3.4之后，使用#{arg0} ,#{arg1}

StudentDao接口

```
List<Student> selectIndex(String name,Integer age);
```

mapper方法

```
<select id="selectIndex" resultType="com.qiangliu8.domain.Student">
    select * from student where name=#{arg0} or age = #{arg1}
</select>
```

测试方法

```
    @Test
    public void testSelectIndex() throws IOException {
        SqlSession sqlSession = MyBatisUtils.getSqlSession();
        StudentDao studentDao = sqlSession.getMapper(StudentDao.class);
        List<Student> studentList = studentDao.selectIndex("刘强",18);
        System.out.println(studentList);
        sqlSession.close();
    }
```

6. **多个参数，使用Map存放多个值**

StudentDao接口

```
List<Student> selectMap(Map<String,Object> map);
```

mapper方法

```
<!--    #{属性名.属性名}-->
<select id="selectMap" resultType="com.qiangliu8.domain.Student">
    select * from student where name=#{mapName} or age = #{mapAge}
</select>
```

测试方法

```
@Test
public void testSelectMap() throws IOException {
    SqlSession sqlSession = MyBatisUtils.getSqlSession();
    StudentDao studentDao = sqlSession.getMapper(StudentDao.class);

    Map<String,Object> map = new HashMap<>();
    map.put("mapName","刘强");
    map.put("mapAge",18);
    List<Student> studentList = studentDao.selectMap(map);
    System.out.println(studentList);
    sqlSession.close();
}
```

7. **#和$**

   #：占位符，告诉mybatis使用实际的参数值代替。并使用PrepareStatement对象执行sql语句，#{...}代替sql语句的“？”。这样更加安全，迅速。

   由mybatis执行下面的代码：

   mybatis创建Connection,PreparedStatement对象

```
String sql = "select id,name,email,age from student where id = ?";
PreparedStatement pst = conn.preparedStatement(sql);
pst.setInt(1,1001);
```

   $字符串替换：告诉mybatis使用$包含的“字符串”替换所在位置。使用Statement把sql语句和${}内容连接起来。主要用在替换表明，列名，不同列排序等操作。

   使用Statement对象执行sql,效率比PreparedStatement低

   $可以替换表名或者列名，你能确定数据是安全的，可以使用$

```
 List<Student> selectUserOrder$(@Param("colName") String colName);

 <select id="selectUserOrder$" resultType="com.qiangliu8.domain.Student">
    select * from student order by  ${colName}
 </select>

@Test
public void testSelectUserOrder$() throws IOException {
    SqlSession sqlSession = MyBatisUtils.getSqlSession();
    StudentDao studentDao = sqlSession.getMapper(StudentDao.class);
    List<Student> studentList = studentDao.selectUserOrder$("name");
    System.out.println(studentList);
    sqlSession.close();
}
```

   对比：

```
select * from student where id = #{studentId}
等同于 select * from student where id = ?
select * from student where id = ${studentId}
等同于 select * from student where id = + 1001
```

      1. #使用？在sql语句中做占位的。使用PreparedStatement执行sql,效率高
      2. #能够避免sql注入，更安全

3. $不使用占位符，是字符串连接方式，使用Statement对象执行sql，效率低
4. $有sql注入的风险，缺乏安全性。
5. $可以替换表明或者列名

封装MyBatis输出结果

resultType参数

resultType结果类型，指sql语句执行完毕后，数据转为的Java对象，Java类型是任意的

resultType结果类型的值：

1. 类型的全限定名称
2. 类型的别名，例如java.lang.Integer别名是int

处理方式：

1. mybatis执行sql语句，然后mybatis调用类的无参构造方法，创建对象
2. mybatis把ResultSet指定列值付给同名的属性

```
 <select id="selectMultiPosition" resultType="com.bjpowernode.domain.Student">
  select id,name, email,age from student
</select>

对等的jdbc
ResultSet rs = executeQuery(" select id,name, email,age from student" )
while(rs.next()){
       Student  student = new Student();
       student.setId(rs.getInt("id"));
       student.setName(rs.getString("name"))
}
```

定义自定义类型的别名

1. 在mybatis主配置文件中定义，使用定义别名

```
<typeAliases>
    <!--        指定一个类型一个自定义别名
            type:自定义类型的全限定名称
            alias:别名
-->
    <typeAlias type="com.qiangliu8.vo.QueryParam" alias="qp"></typeAlias>
    <!--        第二种方式
        <package>name是包名，这个包中的所有类，类名就是别名（类名不区分大小写）-->
    <package name="com.qiangliu8.domain"/>
</typeAliases>
```

2. 再resultType中使用自定义别名

```
<select id="selectViewStudent" resultType="vs">
    select * from  student where id =#{id}
</select>
```

```
<select id="selectViewStudent" resultType="ViewStudent">
    select * from  student where id =#{id}
</select>
```

resultMap参数

查询返回Map

```
Map<Object,Object> selectMapById(Integer id);

<!--    列名是map的key,列值是map的value
    只能最多返回一行记录，多余一行是错误
    -->
    <select id="selectMapById" resultType="java.util.HashMap">
    select * from student where id=#{id}
</select>

@Test
public  void TestSelectMapById(){
    SqlSession sqlSession = MyBatisUtils.getSqlSession();
    StudentDao studentDao = sqlSession.getMapper(StudentDao.class);
    Map<Object,Object> student = studentDao.selectMapById(1001);
    System.out.println(student);
}
```

resultMap:结果映射，指定列名和Java对象的属性对应关系

1. 你自定义列值赋值给哪个属性
2. 当你的列名和属性名不一样时，一定使用resultMap

```
//使用resultMap定义映射关系
List<Student> selectStudents();
```

```
<!--    使用resultMap-->
<!--    1.先定义resultMap-->
<!--    2.在select标签，使用resutMap来引用1的定义-->
    <resultMap id="studentMap" type="com.qiangliu8.domain.Student">
<!--        列名和属性的关系-->
        <id column="id" property="id"></id>
        <result column="name" property="email"></result>
        <result column="email" property="name"></result>
        <result column="age" property="age"></result>
    </resultMap>
    <select id="selectStudents" resultMap="studentMap">
        select * from student
    </select>
```

```
@Test
public  void TestSelectStudents(){
    SqlSession sqlSession = MyBatisUtils.getSqlSession();
    StudentDao studentDao = sqlSession.getMapper(StudentDao.class);
    List<Student> studentList = studentDao.selectStudents();
    System.out.println(studentList);
}
```

resultType和resultMap不要一起用

```
public class MyStudent {
    private Integer MyId;
    private String MyName;
    private String MyEmail;
    private Integer MyAge;
}
```

列名和属性名不一样的方式

第一种方式

```
List<Student> selectStudents1();
```

```
<!--    使用resultMap-->
<!--    1.先定义resultMap-->
<!--    2.在select标签，使用resutMap来引用1的定义-->
<resultMap id="studentMap" type="com.qiangliu8.domain.Student">
    <!--        列名和属性的关系-->
    <id column="id" property="id"></id>
    <result column="name" property="email"></result>
    <result column="email" property="name"></result>
    <result column="age" property="age"></result>
</resultMap>
<select id="selectStudents" resultMap="studentMap">
    select * from student
</select>
```

```
@Test
public  void TestSelectStudents1(){
    SqlSession sqlSession = MyBatisUtils.getSqlSession();
    StudentDao studentDao = sqlSession.getMapper(StudentDao.class);
    List<Student> studentList = studentDao.selectStudents1();
    System.out.println(studentList);
}
```

第二种方式

```
List<Student> selectStudentser2();
```

```
<!--    resultType原则：同名的列值赋值给同名的属性-->
<select id="selectStudents1" resultType="com.qiangliu8.domain.MyStudent">
    select id as MyId, name as MyName,email as MyEmail,age as MyAge from student
</select>
```

```
@Test
public  void TestSelectStudents2(){
    SqlSession sqlSession = MyBatisUtils.getSqlSession();
    StudentDao studentDao = sqlSession.getMapper(StudentDao.class);
    List<Student> studentList = studentDao.selectStudents1();
    System.out.println(studentList);
}
```

模糊查询

第一种模糊查询，在Java代码中指定like的内容

StudentDao接口

```
List<Student> selectStudentsLike(String name);
```

mapper方法

```
<select id="selectStudentsLike" resultType="com.qiangliu8.domain.Student">
    select * from student where name like #{name}
</select>
```

测试方法

```
@Test
public  void TestSelectStudentsLike(){
    SqlSession sqlSession = MyBatisUtils.getSqlSession();
    StudentDao studentDao = sqlSession.getMapper(StudentDao.class);

    String name = "%强%";
    List<Student> studentList = studentDao.selectStudentsLike(name);
    System.out.println(studentList);
}
```

第二种模糊查询，在mapper文件中拼接like的内容

StudentDao接口

```
List<Student> selectStudentsLike2(String name);
```

mapper方法

```
<select id="selectStudentsLike2" resultType="com.qiangliu8.domain.Student">
    select * from student where name like "%" #{name} "%"
</select>
```

测试方法

```
@Test
public  void TestSelectStudentsLike2(){
    SqlSession sqlSession = MyBatisUtils.getSqlSession();
    StudentDao studentDao = sqlSession.getMapper(StudentDao.class);

    String name = "强";
    List<Student> studentList = studentDao.selectStudentsLike2(name);
    System.out.println(studentList);
}
```

四、动态SQL

动态sql：sql的内容是变化的，可以根据条件获取到不同的sql语句。主要是where部分发生变化。

动态sql的实现，使用的的是mybatis提供的标签，,,

动态SQL之if

语法:

```
<if test="判断Java对象的属性值">
    ....
</if>
```

案例

StudentDao接口

```
List<Student> selectStudentIf(Student student);
```

mapper方法

```
<!--<if:test="使用参数java对象的属性值作为判断条件，语法属性=xxx值">-->
<select id="selectStudentIf" resultType="com.qiangliu8.domain.Student">
    select * from student
    where id>0
    <if test="name!=null and name!=''">
        and name =#{name}
    </if>
    <if test="age>0">
        and age >#{age}
    </if>
</select>
```

测试方法

```
@Test
public void TestSelectStudentIf(){
    SqlSession sqlSession = MyBatisUtils.getSqlSession();
    StudentDao studentDao = sqlSession.getMapper(StudentDao.class);
    Student student = new Student(1001,"刘强","1102644615@sqq.com",20);
    List<Student> studentList = studentDao.selectStudentIf(student);
    System.out.println(studentList);
}

Student student = new Student(1001,"刘强","1102644615@sqq.com",20);
相等于==>  Preparing: select * from student where name =? and age >?

Student student = new Student(1001,"刘强","1102644615@sqq.com",null);
相等于==>  ==>  Preparing: select * from student where name =?
```

为了防止第一个判断条件出错导致 select * from student where __ and age >? 这种情况我们可以在判断条件前加上id>0这个恒等条件

动态SQL之where

用来包含多个 的，当多个if有一个成立的话，会自动增加一个where关键字，并去掉if中多余的and，or等。

语法:

```
where: <where> <if><if>...</where>
```

案例

StudentDao接口

```
List<Student> selectStudentwhere(Student student);
```

mapper方法

```
<select id="selectStudentwhere" resultType="com.qiangliu8.domain.Student">
    select * from student
    <where>
        <if test="name!=null and name!=''">
            name =#{name}
        </if>
        <if test="age>0">
            or age >#{age}
        </if>
    </where>
```

测试方法

```
@Test
public  void TestSelectStudentWhere(){
    SqlSession sqlSession = MyBatisUtils.getSqlSession();
    StudentDao studentDao = sqlSession.getMapper(StudentDao.class);
    Student student = new Student(1001,"刘强","1102644615@sqq.com",20);
    List<Student> studentList = studentDao.selectStudentwhere(student);
    System.out.println(studentList);
}

Student student = new Student(1001,null,"1102644615@sqq.com",20);
相等于==>  Preparing: select * from student where age >?

    Student student = new Student(1001,"刘强","1102644615@sqq.com",20);
相等于==>  Preparing: select * from student WHERE name =? or age >?
```

动态SQL之foreach

用来循环Java中的数组，list集合的。主要用在sql的in语句中。

例如学生id是1001，1002，1003的三个学生。

select * form student where id in (1001,1002,1003)

```
List<Integer> list = new ArrayList<>();
list.add(1001);
```

```
list.add(1002);
list.add(1003);
String sql = "select * from student where id in ";
StringBuilder builder = new StringBuilder("");
int init = 0;
int len = list.size();
builder.append("(");
for (Integer i:list) {
    builder.append(i).append(",");
}
builder.deleteCharAt(builder.length()-1);
builder.append(")");
System.out.println(sql+builder.toString());//select * from student where id in
(1001,1002,1003)
```

案例--参数是integer集合

StudentDao接口

```
List<Student> selectStuForeachOne(List<Student> studentList);
```

mapper方法

```
<select id="selectStuForeachOne" resultType="com.qiangliu8.domain.Student">
    select * from student where id in
    --          colloection用来表示接口中的方法参数的类型，如果是数据则用array。如果是list
集合使用list
    --          item:自定义的，表示数组和集合成员的变量
    --          open:循环开始的字符
    --          close:循环结束时的字符
    --          separator:集合成员之间的分割符
    <foreach collection="list" item="myid" open="(" close=")" separator=",">
        #{myid}
    </foreach>
</select>
```

测试方法

```
@Test
public  void TestForeach2(){
    SqlSession sqlSession = MyBatisUtils.getSqlSession();
    StudentDao studentDao = sqlSession.getMapper(StudentDao.class);
    List<Integer> integerList = new ArrayList<>();
    integerList.add(1001);
    integerList.add(1002);
    integerList.add(1003);
    List<Student> studentList = studentDao.selectStuForeachOne(integerList);
    System.out.println(studentList);
    sqlSession.close();
}
```

案例--参数是对象用对象.属性

StudentDao接口

```
List<Student> selectStuForeach2(List<Student> studentList);
```

mapper方法

```
<select id="selectStuForeach2" resultType="com.qiangliu8.domain.Student">
    select * from student where id in
    <foreach collection="list" item="mystu" open="(" close=")" separator=",">
        #{mystu.id}
    </foreach>
</select>
```

测试方法

```
@Test
public  void TestForeach3(){
    SqlSession sqlSession = MyBatisUtils.getSqlSession();
    StudentDao studentDao = sqlSession.getMapper(StudentDao.class);
    List<Student> studentList = new ArrayList<>();
    studentList.add(new Student(1001,null,null,null));
    studentList.add(new Student(1002,null,null,null));
    studentList.add(new Student(1003,null,null,null));
    List<Student> studentList2 = studentDao.selectStuForeach2(studentList);
    System.out.println(studentList2);
    sqlSession.close();
}
```

动态SQL之代码片段

标签用于定于SQL片段，以便其他SQL标签复用。而其他标签使用该SQL片段，需要使用子标签。该标签可以定义SQL语句中的任何部分，所以子标签可以放在动态SQL的任何位置。

即复用语法。

```
<sql id="studentSql">
    select * from student
</sql>


<select id="selectStudentIf" resultType="com.qiangliu8.domain.Student">
    <include refid="studentSql"/>
    where
    <if test="name!=null and name!=''">
        name =#{name}
    </if>
    <if test="age>0">
        and age >#{age}
    </if>
</select>
```

五、MyBatis配置文件

数据库的属性配置文件

把数据库连接信息放到一个单独的文件中，和mybatis主配置文件分开。

目的是便于修改，保存，处理多个数据库的信息。步骤如下：

1. 在resources目录中定义一个属性配置文件，xxx.properties。

   在属性配置文件中，定义数据，格式是 key=value

2. 在mybatis的主配置文件，使用指定文件的位置

   在需要使用值的地方，${key}

```
<!--    指定properties文件的位置，从类路径根开始找文件-->
<properties resource="jdbc.properties"></properties>


。。。。。
<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}"/>
</dataSource>
```

指定多个mapper文件的方式

第一种方式:指定多个mapper文件

```
<mappers>
    <!--        从类路径开始的路径信息 target/classess(类路径) -->
    <mapper resource="com/qiangliu8/dao/StudentDao.xml"/>
    <mapper resource="com/qiangliu8/dao/StudentDao2.xml"/>
</mappers>
```

第二种方式:使用包名

name: xml文件（mapper文件）所在的包名。要求：

1. mapper文件名称需要和接口名称一样，区分大小写的一样
2. mapper文件和dao接口需要在同一目录

```
<mappers>
    <package name="com.qiangliu8.dao"/>
</mappers>
```

六、PageHelper分页

1. 加入依赖

```
<dependency>
  <groupId>com.github.pagehelper</groupId>
  <artifactId>pagehelper</artifactId>
  <version>5.1.10</version>
</dependency>
```

2. 配置插件加在环境前面

```
<!--    配置插件-->
<plugins>
    <plugin interceptor="com.github.pagehelper.PageInterceptor"></plugin>
</plugins>

<environments default="development">
```

3. 代码

```
@Test
public  void TestAll(){
    SqlSession sqlSession = MyBatisUtils.getSqlSession();
    StudentDao studentDao = sqlSession.getMapper(StudentDao.class);
    //加入PageHelper方法，分页
    //pageNum：第几页，从1开始
    //pageSize:一夜中有多少行数据
    PageHelper.startPage(2,3);
    List<Student> studentList = studentDao.selectAll();
    System.out.println(studentList);
    sqlSession.close();
}
```