# 案例：学生信息查询系统

**【案例目标】**

本章对MyBatis的动态SQL进行了详细讲解，包括使用动态SQL进行条件查询、更新以及复杂查询操作。本案例要求利用本章所学知识完成一个学生信息查询系统，该系统要求实现2个以下功能。

（1）多条件查询

当用户输入的学生姓名不为空，则只根据学生姓名进行学生信息的查询；

当用户输入的学生姓名为空，而学生专业不为空，则只根据学生专业进行学生的查询；

当用户输入的学生姓名和专业都为空，则要求查询出所有学号不为空的学生信息。

（2）单条件查询查询出所有id值小于5的学生的信息。

**【实现步骤】：多条件查询**

**1.项目搭建**

创建一个名称为mybatis-demo03的项目，项目的具体搭建过程请参考1.3节。

**2.数据准备**

在名称为mybatis的数据库中，创建一个dm\_student表，并插入几条测试数据，具体代码如下所示。

# 使用mybatis数据库

USE mybatis;

# 创建一个名称为dm\_student的表

CREATE TABLE dm\_student(

id int(32) PRIMARY KEY AUTO\_INCREMENT,

name varchar(50),

major varchar(50),

sno varchar(16)

);

# 插入7条数据

INSERT INTO dm\_student VALUES ('1', '张三', '数学', '10001');

INSERT INTO dm\_student VALUES ('2', '李四', '英语', '10002');

INSERT INTO dm\_student VALUES ('3', '王五', '计算机', '10003');

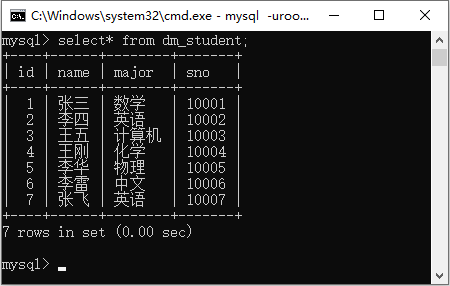
INSERT INTO dm\_student VALUES ('4', '王刚', '化学', '10004');

INSERT INTO dm\_student VALUES ('5', '李华', '物理', '10005');

INSERT INTO dm\_student VALUES ('6', '李雷', '中文', '10006');

INSERT INTO dm\_student VALUES ('7', '张飞', '英语', '10007');

完成上述操作后，dm\_student表中的数据如下图所示。



dm\_student表中的数据

**3.POJO类准备**

在项目src/main/java目录下创建com.itheima.pojo包，在com.itheima.pojo包下创建持久化类Student，在类中声明id、name、major和sno属性，以及属性对应的getter/setter方法。Student类具体代码如文件3-1所示。

1. Student.java
2. package com.itheima.pojo;
3. /\*\*
4. \*学生持久化类
5. \*/
6. public class Student {
7. private Integer id; //主键id
8. private String name; // 姓名
9. private String major; // 专业
10. private String sno; // 学号
11. public Integer getId() {
12. return id;
13. }
14. public void setId(Integer id) {
15. this.id = id;
16. }
17. public String getName() {
18. return name;
19. }
20. public void setName(String name) {
21. this.name = name;
22. }
23. public String getMajor() {
24. return major;
25. }
26. public void setMajor(String major) {
27. this.major = major;
28. }
29. public String getSno() {
30. return sno;
31. }
32. public void setSno(String sno) {
33. this.sno = sno;
34. }
35. @Override
36. public String toString() {
37. return "Student{" + "id=" + id +
38. ", name='" + name + ", major=" + major +
39. ", sno=" + sno + '}';
40. }
41. }

**4.创建映射文件**

在项目src/main/java目录下创建com.itheima.mapper包，在com.itheima.mapper包下创建映射文件StudentMapper.xml，在映射文件中，编写根据学生姓名和专业组合成的条件查询学生信息的动态SQL。StudentMapper.xml具体代码如文件3-2所示。

1. StudentMapper.xml
2. <?xml version="1.0" encoding="UTF-8"?>
3. <!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
4. "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
5. <mapper namespace="com.itheima.mapper.StudentMapper">
6. <select id="findStudentByNameAndMajor"
7. parameterType="com.itheima.pojo.Student"
8. resultType="com.itheima.pojo.Student">
9. select \* from dm\_student where 1=1
10. <choose>
11. <when test="name !=null and name !=''">
12. and name like concat('%',#{name}, '%')
13. </when>
14. <when test="major !=null and major !=''">
15. and major= #{major}
16. </when>
17. <otherwise>
18. and sno is not null
19. </otherwise>
20. </choose>
21. </select>
22. </mapper>

在文件3-2中，第9~19行代码使用<choose>元素进行SQL拼接，当第一个<when>元素中的条件为真时，只动态组装第一个<when>元素内的SQL片段并执行，否则就继续向下判断第二个<when>元素中的条件是否为真，以此类推，直到某一个<when>元素中的条件为真。当前面所有when元素中的条件都不为真时，则动态组装<otherwise>元素内的SQL片段并执行。

**5.修改mybatis-config.xml核心配置文件**

在mybatis-config.xml映射文件的<mappers>元素下添加StudentMapper.xml映射文件路径的配置，用于将StudentMapper.xml映射文件加载到程序中。具体配置代码如下。

<mapper resource="com/itheima/mapper/StudentMapper.xml"/>

上述代码中，<mapper>元素指定了StudentMapper.xml映射文件的路径。

**6.编写MyBatisUtils工具类**

在项目src/main/java目录下创建com.itheima.utils包，在com.itheima.utils包下创建MyBatisUtils工具类，该类用于封装读取配置文件信息的代码。MyBatisUtils类具体代码如文件3-3所示。

1. MyBatisUtils.java
2. package com.itheima.utils;
3. import java.io.Reader;
4. import org.apache.ibatis.io.Resources;
5. import org.apache.ibatis.session.SqlSession;
6. import org.apache.ibatis.session.SqlSessionFactory;
7. import org.apache.ibatis.session.SqlSessionFactoryBuilder;
8. /\*\*
9. \* 工具类
10. \*/
11. public class MyBatisUtils {
12. private static SqlSessionFactory sqlSessionFactory = null;
13. // 初始化SqlSessionFactory对象
14. static {
15. try {
16. // 使用MyBatis提供的Resources类加载MyBatis的配置文件
17. Reader reader =
18. Resources.getResourceAsReader("mybatis-config.xml");
19. // 构建SqlSessionFactory工厂
20. sqlSessionFactory sqlSessionFactory =
21. new SqlSessionFactoryBuilder().build(reader);
22. } catch (Exception e) {
23. e.printStackTrace();
24. }
25. }
26. // 获取SqlSession对象的静态方法
27. public static SqlSession getSession() {
28. return sqlSessionFactory.openSession();
29. }
30. }

在文件3-3中，第16~17行代码读取mybatis-config.xml文件内容到reader对象中；第19~20行代码创建SqlSessionFactory类的实例；第26~28行代码是定义了一个getSession()静态方法，并在静态方法中通过SqlSessionFactory类的实例创建SqlSession实例。

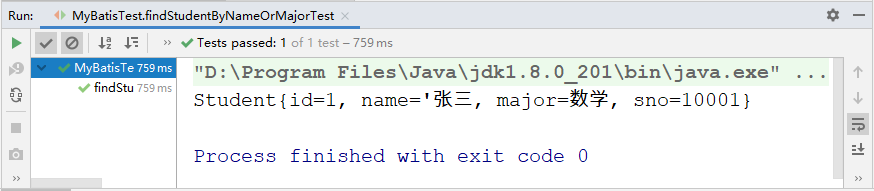
**7.编写测试方法**

为了验证上述配置，可以在测试类MyBatisTest中，编写测试方法findStudentByNameOrMajorTest()，该方法用于根据学生姓名或专业查询学生信息，具体代码如下所示。

1. /\*\*
2. 根据学生姓名或职业查询学生信息列表
3. \*/
4. @Test
5. public void findStudentByNameOrMajorTest(){
6. // 通过工具类生成SqlSession对象
7. SqlSession session = MyBatisUtils.getSession();
8. // Student，封装需要组合查询的条件
9. Student student = new Student();
10. student.setName("张三");
11. student.setMajor("英语");
12. // 执行SqlSession的查询方法，返回结果集
13. List<Student> students = session.selectList("com.itheima.mapper"
14. + ".StudentMapper.findStudentByNameAndMajor",student);
15. // 输出查询结果信息
16. for (Student student2 : students) {
17. // 打印输出结果
18. System.out.println(student2);
19. }
20. // 关闭SqlSession
21. session.close();
22. }

上述代码中，第7行代码通过MyBatisUtils工具类获取SqlSession对象；第9~11行代码创建Student对象，封装需要组合查询的条件；第13~14行代码执行SqlSession的查询方法，返回结果集；第16~19行代码使用foreach循环打印查询结果信息；第21行代码关闭SqlSession，释放资源。

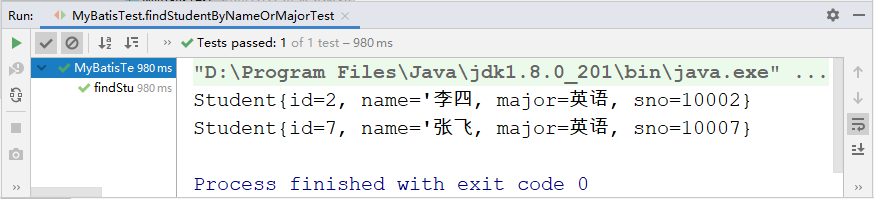
执行测试类MyBatisTest的findStudentByNameOrMajorTest()方法，控制台的输出结果如图3-1所示。



1. findStudentByNameOrMajorTest()方法执行结果（1）

由图3-1中的输出结果分析可知，在查询学生信息时，虽然同时传入了姓名和专业两个查询条件，但MyBatis所生成的SQL只是动态组装了学生姓名条件进行查询。

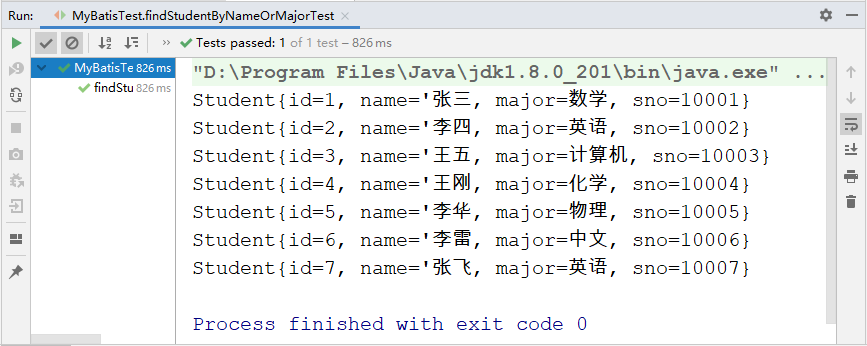
如果将上述代码中的第10行代码“student.setName("张三");”删除或者注释掉，使SQL只按专业进行查询。再次执行findStudentByNameOrMajorTest ()方法，控制台的输出结果如图3-2所示。



1. findStudentByNameOrMajorTest()方法执行结果（2）

由图3-2中的输出结果分析可知，MyBatis生成的SQL组装了学生职业进行条件查询，同样查询出了学生信息。

如果将上述代码中的第10~11行代码都删除或者注释掉（即学生姓名和专业都为空），那么程序的执行结果如图3-3所示。



1. findStudentByNameOrMajorTest()方法执行结果（3）

由图3-3中的输出结果分析可知，当姓名和专业参数都为空时，MyBatis的SQL组装了<otherwise>元素中的SQL片段进行条件查询。

**【实现步骤】：单条件查询**

**1.修改映射文件**

在映射文件StudentMapper.xml中的<mapper>元素下，编写查询所有id值小于5的学生信息的动态SQL。具体代码如下所示。

<!--<foreach>遍历List -->

<select id="findByList" parameterType="java.util.List"

resultType="com.itheima.pojo.Student">

select \* from dm\_student where id in

<foreach item="id" index="index" collection="list"

open="(" separator="," close=")">

#{id}

</foreach>

</select>

上述配置代码中，使用<foreach>元素迭代List集合，实现学生信息的批量查询操作。其中，<foreach>元素的collection属性用于设置传入的参数为List类型，<foreach>元素将学生id信息存储在List集合中，并对List集合进行遍历，遍历出的值用于构建SQL语句中的in条件语句。

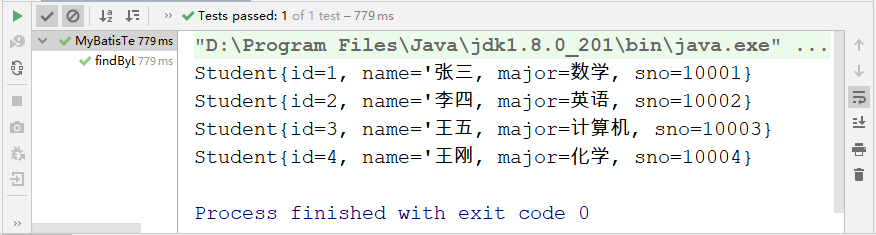
**2.编写测试方法**

为了验证上述配置，可以在测试类MyBatisTest中，编写测试方法findByListTest()，findByListTest()具体代码如下所示。

1. /\*\*
2. \* 根据学生id批量查询学生信息
3. \*/
4. @Test
5. public void findByListTest(){
6. // 获取SqlSession
7. SqlSession session = MyBatisUtils.getSession();
8. // 创建List集合，封装查询id
9. List<Integer> ids=new ArrayList<Integer>();
10. // 将小于5的id值放入list中
11. for(int i =1;i<5;i++){
12. ids.add(i);
13. }
14. // 执行SqlSession的查询方法，返回结果集
15. List<Student> students = session.selectList("com.itheima.mapper"
16. + ".StudentMapper.findByList", ids);
17. // 输出查询结果信息
18. for (Student student : students) {
19. // 打印输出结果
20. System.out.println(student);
21. }
22. // 关闭SqlSession
23. session.close();
24. }

上述代码中，第7行代码通过MyBatisUtils工具类获取了SqlSession对象；第9~13行通过for循环创建List集合，并将小于5的id值放入List集合中；第15~16行代码执行SqlSession的查询方法，并返回查询到的结果集；第18~21行代码遍历查询到的结果集信息并输出；第23行代码关闭SqlSession，释放资源。

执行MyBatisTest测试类的findByListTest()方法，控制台的输出结果如图3-4所示。



1. findByListTest()方法执行结果