### 案例：基于MyBatis注解的学生管理程序

#### 【任务目标】

现有一个学生表s\_student和一个班级表c\_class，其中，班级表c\_class和学生表s\_student是一对多的关系。学生表s\_student和班级表c\_class如表1和表2所示。

表1 学生表（s\_student）

|  |  |  |  |
| --- | --- | --- | --- |
| **学生id**  **id** | **学生姓名**  **name** | **学生年龄**  **age** | **所属班级**  **cid** |
| 1 | 张三 | 18 | 1 |
| 2 | 李四 | 18 | 2 |
| 3 | 王五 | 19 | 2 |
| 4 | 赵六 | 20 | 1 |

表2 班级表（c\_class）

|  |  |
| --- | --- |
| **班级id**  **id** | **班级名称**  **classname** |
| 1 | 一班 |
| 2 | 二班 |

请使用MyBatis注解完成以下几个要求：

1. MyBatis注解实现查询操作

根据表1和表2在数据库分别创建一个学生表s\_student和一个班级表c\_class， 并查询id为2的学生的信息。

1. MyBatis注解实现修改操作

修改id为4的学生的姓名修改为李雷，年龄修改为21。

MyBatis注解实现修改操作

1. MyBatis注解实现一对多查询

查询出二班所有学生的信息。

#### 【实现步骤】：MyBatis注解实现查询操作

##### 1.项目搭建

创建一个名称为mybatis-demo05的项目，项目的具体搭建过程请参考1.3节。

##### 2.数据库准备

在名为mybatis的数据库中，创建两个数据表，分别为学生表s\_student和班级表c\_class，同时在表中预先插入几条测试数据。执行的SQL语句如下所示。

USE mybatis;

# 创建一个名称为c\_class的表

CREATE TABLE c\_class (

id int(32) PRIMARY KEY AUTO\_INCREMENT,

classname varchar(40)

);

# 插入2条数据

INSERT INTO c\_class VALUES (1, '一班');

INSERT INTO c\_class VALUES (2, '二班');

# 创建一个名称为s\_student的表

CREATE TABLE s\_student (

id int(32) PRIMARY KEY AUTO\_INCREMENT,

name varchar(40),

age int,

cid int(32) NOT NULL,

FOREIGN KEY(cid) REFERENCES c\_class(id)

);

# 插入4条数据

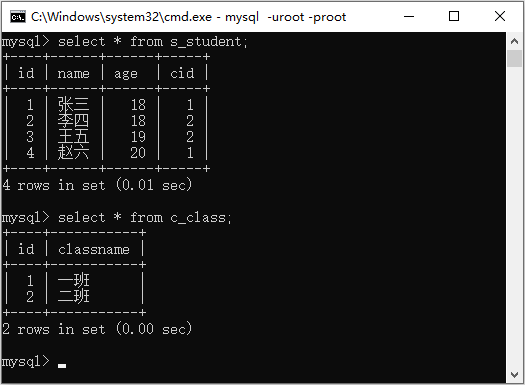
INSERT INTO s\_student VALUES (1, '张三', 18,1);

INSERT INTO s\_student VALUES (2, '李四', 18,2);

INSERT INTO s\_student VALUES (3, '王五', 19,2);

INSERT INTO s\_student VALUES (4, '赵六', 20,1);

执行完上述操作后， s\_student表和c\_class表中的数据如图5-1所示。



1. s\_student表和c\_class表中的数据

##### 3.POJO类准备

在项目的src/main/java目录下创建com.itheima.pojo包，在com.itheima.pojo包中创建持久化类IClass，并在类中定义相关属性和方法，该类用于封装IClass对象的id、班级名称以及关联的学生集合等属性。具体代码如下所示。

IClass.java

1. package com.itheima.pojo;
2. import java.util.List;
3. /\*\*
4. \* 班级持久化类
5. \*/
6. public class IClass {
7. private Integer id; // 主键id
8. private String classname; // 班级名称
9. private List<IStudent> studentList; // 学生集合
10. public Integer getId() {
11. return id;
12. }
13. public void setId(Integer id) {
14. this.id = id;
15. }
16. public String getClassname() {
17. return classname;
18. }
19. public void setClassname(String classname) {
20. this.classname = classname;
21. }
22. public List<IStudent> getStudentList() {
23. return studentList;
24. }
25. public void setStudentList(List<IStudent> studentList) {
26. this.studentList = studentList;
27. }
28. @Override
29. public String toString() {
30. return "IClass{" +
31. "id=" + id +
32. ", classname='" + classname +
33. ", studentList=" + studentList + '}';
34. }
35. }

上述代码中，分别定义了各自的属性以及对应的getter/setter方法，同时为了方便查看输出结果，重写了toString()方法。

在com.itheima.pojo包中，创建持久化类IStudent，并在类中定义相关属性和方法，该类用户封装IStudent对象的id、姓名和年龄等属性。具体代码如下所示。

IStudent.java

1. package com.itheima.pojo;
2. /\*\*
3. \* 学生持久化类
4. \*/
5. public class IStudent {
6. private Integer id; // 主键id
7. private String name; // 姓名
8. private int age; // 年龄
9. public Integer getId() {
10. return id;
11. }
12. public void setId(Integer id) {
13. this.id = id;
14. }
15. public String getName() {
16. return name;
17. }
18. public void setName(String name) {
19. this.name = name;
20. }
21. public int getAge() {
22. return age;
23. }
24. public void setAge(int age) {
25. this.age = age;
26. }
27. @Override
28. public String toString() {
29. return "Student{" + "id=" + id +
30. ", name='" + name + ", age=" + age + '}';
31. }
32. }

上述代码中，分别定义了各自的属性以及对应的getter/setter方法，同时为了方便查看输出结果，重写了toString()方法。

##### 4.创建Mapper接口

在项目的src/main/java目录下创建com.itheima.dao包，并在com.itheima.dao包下创建IStudentMapper接口，用于编写@Select注解映射的select查询语句。IStudentMapper接口具体代码如下所示。

IStudentMapper.java

1. package com.itheima.dao;
2. import com.itheima.pojo.IStudent;
3. import org.apache.ibatis.annotations.Select;
4. public interface IStudentMapper {
5. @Select("select \* from s\_student where id = #{id}")
6. IStudent selectStudent(int id);
7. }

上述代码中，@Select注解的参数是一个查询语句，当程序调用@Select注解标注的selectStudent ()方法时，@Select注解中映射的查询语句将被执行。

##### 5. 修改mybatis-config.xml核心配置文件

在核心配置文件mybatis-config.xml中的<mappers>元素下引入IStudentMapper接口，将IStudentMapper接口加载到核心配置文件中，具体代码如下所示。

<mapper class="com.itheima.dao.IStudentMapper"/>

##### 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。具体代码如下所示。

MyBatisUtils.java

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

上述代码中，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行代码是创建了一个getSession()静态方法，并在静态方法中通过SqlSessionFactory类的实例创建SqlSession实例。

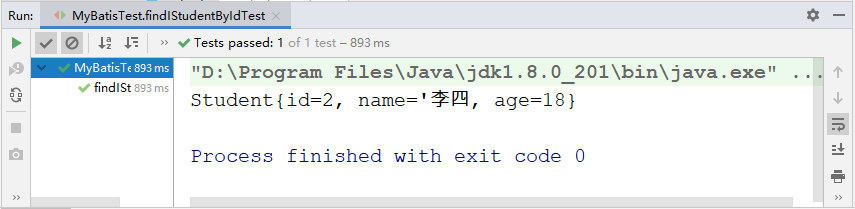
##### 7 编写测试方法

为了验证上述配置，在项目的src/test/java目录下创建Test文件夹，在Test文件夹创建测试类MyBatisTest，在测试类MyBatisTest中编写测试方法findIStudentByIdTest ()，具体代码如下所示。

1. @Test
2. public void findIStudentByIdTest() {
3. // 1.通过工具类获取SqlSession对象
4. SqlSession session = MyBatisUtils.getSession();
5. IStudentMapper mapper = session.getMapper(IStudentMapper.class);
6. // 2.使用IStudentMapper对象查询id为1的学生的信息
7. IStudent student = mapper.selectStudent(2);
8. System.out.println(student.toString());
9. // 3.关闭SqlSession
10. session.close();
11. }

上述代码中，第4行代码通过MyBatisUtils工具类获取SqlSession对象；第5~7行代码通过SqlSession对象调用getMapper()方法获取WorkerMapper类对象，并使用mapper对象调用IStudentMapper接口的selectStudent ()方法查询id为2的学生的信息。第8行代码使用输出语句输出查询结果信息；第10行代码关闭SqlSession，释放资源。

执行MyBatisTest测试类的findIStudentByIdTest()方法后，控制台的输出结果如图5-2所示。



1. findIStudentByIdTest()方法的运行结果

#### 【实现步骤】：MyBatis注解实现修改操作

##### 1. 修改Mapper接口

在IStudentMapper接口中添加更新s\_student表中数据的方法，并在方法上添加@Update注解，具体代码如下所示。

@Update("update s\_student set name = #{name},age = #{age} "

+"where id = #{id}")

int updateStudent(IStudent student);

在上述代码中，@Insert注解的参数是一条插入语句，当程序调用@Insert注解标注的insertWorker()方法时，@Insert注解中映射的插入语句将被执行。

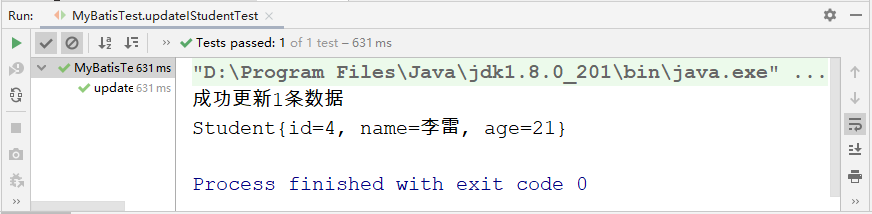
##### 2. 编写测试方法

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

1. @Test
2. public void updateIStudentTest() {
3. // 1.通过工具类生成SqlSession对象
4. SqlSession session = MyBatisUtils.getSession();
5. IStudent student = new IStudent();
6. student.setId(4);
7. student.setName("李雷");
8. student.setAge(21);
9. IStudentMapper mapper = session.getMapper(IStudentMapper.class);
10. // 2.更新学生信息
11. int result = mapper.updateStudent(student);
12. if(result>0){
13. System.out.println("成功更新"+result+"条数据");
14. }else {
15. System.out.println("更新数据失败");
16. }
17. System.out.println(student.toString());
18. session.commit();
19. // 3.关闭SqlSession
20. session.close();
21. }

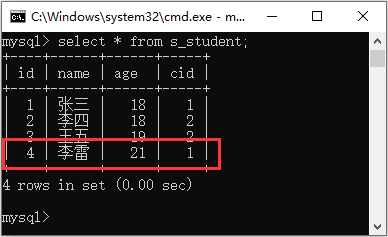
上述代码中，第4行代码通过MyBatisUtils工具类获取SqlSession对象；第9~11行代码通过SqlSession对象调用getMapper()方法获取IStudentMapper类对象，并使用IStudentMapper类对象调用IStudentMapper接口的updateStudent()方法更新学生的信息；第12~16行代码判断数据更新是否成功；第17行代码使用输出语句输出查询结果信息；第20行代码关闭SqlSession，释放资源。

执行MyBatisTest测试类的updateIStudentTest()方法后，控制台的输出结果如图5-3所示。



1. updateIStudentTest()方法的运行结果

完成上述操作后，数据库s\_student表中的数据如图5-4所示。



1. s\_student表中的数据

#### 【实现步骤】：MyBatis注解实现一对多查询

##### 1. 修改Mapper接口

（1）在IStudentMapper接口中编写selectStudentByCid()方法，通过cid查询对应班级中的学生信息。selectStudentByCid()方法具体代码如下所示。

1. @Select("select \* from s\_student where cid=#{id} ")
2. @Results({@Result(id = true,column = "id",property = "id"),
3. @Result(column = "classname",property = "classname")
4. })
5. List<IStudent> selectStudentByCid(int cid);

上述代码中，第1行代码使用@Select注解映射根据c\_id查询IStudent对象的SQL语句，当程序调用@Select注解标注的selectStudentByCid ()方法时，@Select注解中映射的查询语句将被执行；第2~4行代码使用@Results注解映射查询结果，在@Results注解中，使用@Result注解完成IStudent实体类中属性和数据表中字段的映射。

（2）在项目的com.itheima.dao包下创建IClassMapper接口，在该接口中编写selectClassById ()方法，通过id查询班级信息。IClassMapper接口具体代码如下所示。

IClassMapper.java

1. package com.itheima.dao;
2. import com.itheima.pojo.IClass;
3. import org.apache.ibatis.annotations.Many;
4. import org.apache.ibatis.annotations.Result;
5. import org.apache.ibatis.annotations.Results;
6. import org.apache.ibatis.annotations.Select;
7. public interface IClassMapper {
8. @Select("select \* from c\_class where id=#{id} ")
9. @Results({@Result(id = true,column = "id",property = "id"),
10. @Result(column = "classname",property = "classname"),
11. @Result(column = "id",property = "studentList",
12. many = @Many(select=
13. "com.itheima.dao.IStudentMapper.selectStudentByCid"))})
14. IClass selectClassById(int id);
15. }

上述代码中，第8行代码使用@Select注解映射根据id查询IClass对象的SQL语句，当程序调用@Select注解标注的selectClassById ()方法时，@Select注解中映射的查询语句将被执行；第9~13行代码使用@Results注解映射查询结果，在@Results注解中，使用3个@Result注解完成IClass实体类中属性和数据表中字段的映射。其中，第12~13行代码通过@Many注解表明数据表c\_class和s\_student之间是一对多关联关系。在@Many注解中，select属性用于指定关联属性studentList的值是通过执行com.itheima.dao包中IStudentMapper接口定义的selectStudentByCid ()方法获得的。

##### 2. 修改mybatis-config.xml核心配置文件

在核心配置文件mybatis-config.xml中的<mappers>元素下引入IClassMapper接口，将IClassMapper接口加载到核心配置文件中，具体代码如下所示。

<mapper class="com.itheima.dao.IClassMapper"/>

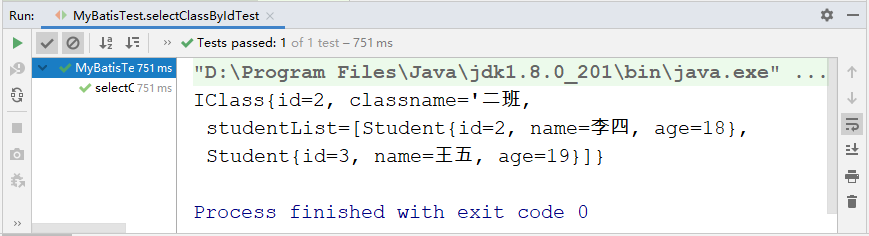
##### 3. 编写测试方法

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

1. @Test
2. public void selectClassByIdTest() {
3. // 1.通过工具类生成SqlSession对象
4. SqlSession session = MyBatisUtils.getSession();
5. IClassMapper mapper = session.getMapper(IClassMapper.class);
6. // 2.查询id为2的班级中学生的信息
7. IClass icalss = mapper.selectClassById(2);
8. System.out.println(icalss.toString());
9. session.close();
10. }

上述代码中，第4行代码通过MyBatisUtils工具类获取SqlSession对象；第5~7行代码通过SqlSession对象调用getMapper()方法获取IClassMapper类对象，并使用mapper对象调用IClassMapper接口的selectClassById()方法查询id为2的班级中学生的信息。第8行代码使用输出语句输出查询结果信息；第10行代码关闭SqlSession，释放资源。

执行MyBatisTest测试类的selectClassByIdTest()方法后，控制台的输出结果如图5-11所示。



1. selectClassByIdTest()方法的运行结果

由图5-11可知，控制台输出了id为2的班级中学生的信息。这表明在查询出IClass对象信息的同时，其关联的IStudent对象的信息也被查询出来了。