# 案例：员工管理系统

**【任务目标】**

本章对MyBatis的核心配置进行了详细讲解，包括MyBatis的核心对象、MyBatis的核心配置文件以及MyBatis映射文件中的元素。现有一员工表，如表2-1所示。

1. 员工表（employee）

|  |  |  |  |
| --- | --- | --- | --- |
| **员工编号**  **（id）** | **商品名称**  **（name）** | **员工年龄**  **（age）** | **员工职位**  **（position）** |
| 1 | 张三 | 20 | 员工 |
| 2 | 李四 | 18 | 员工 |
| 3 | 王五 | 35 | 经理 |

本案例要求根据表2-1在数据库中创建一个employee表，并利用本章所学知识完成一个员工管理系统，该系统需要实现以下几个功能。

1.根据id查询员工信息。

2.新增员工信息。

3.根据id修改员工信息。

4.根据id删除员工信息。

**【实现步骤】**

**1.项目搭建**

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

**2.数据准备**

在mybatis数据库中创建employee表，并在employee表中插入几条数据，具体代码如下所示。

use mybatis;

create table employee(

id int primary key auto\_increment,

name varchar(20) not null,

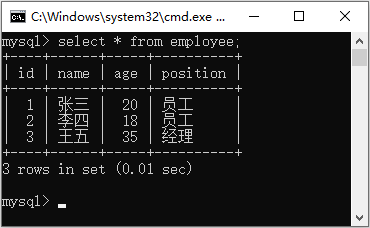
age int not null,

position varchar(20)

);

insert into employee(id,name,age,position) values(null,'张三',20,'员工 '),(null,'李四',18, '员工'),(null,'王五',35,'经理');

完成上述操作后，employee表中的数据如图2-1所示。



1. employee表中的数据（1）

**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Employee，并在类中声明id（编号）、name（姓名）、age（年龄）和position（职位）属性，以及属性对应的getter/setter方法。Employee类具体代码如文件2-1所示。

1. Employee.java
2. package com.itheima.pojo;
3. /\*\*
4. \*员工持久化类
5. \*/
6. public class Employee {
7. private Integer id; // 主键id
8. private String name; // 姓名
9. private Integer age; // 年龄
10. private String position; // 职位
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 Integer getAge() {
24. return age;
25. }
26. public void setAge(Integer age) {
27. this.age = age;
28. }
29. public String getPosition() {
30. return position;
31. }
32. public void setPosition(String position) {
33. this.position = position;
34. }
35. @Override
36. public String toString() {
37. return "Employee{" + "id=" + id + ", name=" + name +
38. ", age=" + age + ", position=" + position +'}';
39. }
40. }

**4.编写映射文件**

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EmployeeMapper.xml，该文件主要用于实现SQL语句和Java对象之间的映射。EmployeeMapper.xml具体代码如文件2-2所示。

1. EmployeeMapper.xml
2. <?xml version="1.0" encoding="UTF-8"?>
3. <!DOCTYPE mapper
4. PUBLIC "-//mybatis.org//DTD mapper 3.0//EN"
5. "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
6. <mapper namespace="com.itheima.mapper.EmployeeMapper">
7. <!--根据id查询员工信息 -->
8. <select id="findById" parameterType="Integer"
9. resultType="com.itheima.pojo.Employee">
10. select \* from employee where id = #{id}
11. </select>
12. <!-- 插入员工信息 -->
13. <insert id="addEmployee" parameterType="com.itheima.pojo.Employee">
14. insert into employee(id,name,age,position)values
15. (#{id},#{name},#{age},#{position})
16. </insert>
17. <!--修改员工信息-->
18. <update id="updateEmployee" parameterType="com.itheima.pojo.Employee">
19. update employee set name= #{name},age = #{age},position= #{position} where
20. id = #{id}
21. </update>
22. <!-- 删除员工信息 -->
23. <delete id="deleteEmployee" parameterType="Integer">
24. delete from employee where id=#{id}
25. </delete>
26. </mapper>

在文件2-2中，第7~10行代码是在<select>语句块中编写了一条根据id查询员工信息的SQL语句；第12~15行代码是在<insert>语句块中编写了一条插入员工信息的SQL语句；第17~20行代码是在<update>语句块中编写了一条根据id修改员工信息的SQL语句；第22~24行代码是在<delete>语句块中编写了一条根据id删除员工信息的SQL语句。

**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EmployeeMapper.xml映射文件路径的配置，用于将EmployeeMapper.xml映射文件加载到程序中。添加的具体代码如下。

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

上述代码中，<mapper>元素指定了EmployeeMapper.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2-3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 =
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2-3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行代码定义一个getSession()静态方法，并在静态方法中通过SqlSessionFactory类的实例创建SqlSession实例。

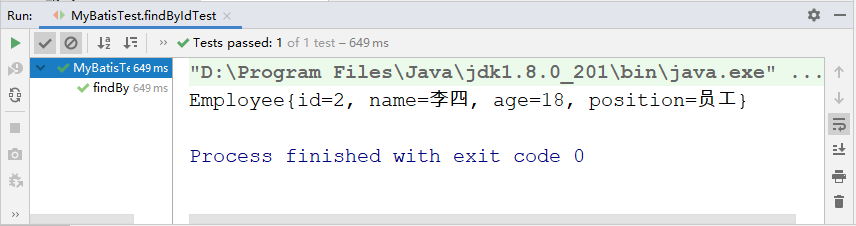
**7.编写测试类**

（1）在项目src/test/java目录下创建Test包，在Test包下创建MyBatisTest测试类，该类主要用于程序测试。在MyBatisTest测试类中添加findByIdTest()方法，该方法可以根据id查询员工信息。findByIdTest()方法具体代码如下所示。

1. @Test
2. public void findByIdTest(){
3. // 通过工具类生成SqlSession对象
4. SqlSession session = MyBatisUtils.getSession();
5. Employee employee =
6. session.selectOne("com.itheima.mapper.EmployeeMapper.findById",2);
7. System.out.println(employee);
8. session.commit();
9. // 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5~6行代码执行SqlSession的查询方法selectOne()，根据id查询员工信息，并返回查询到的Employee对象；第7行代码打印查询到的Employee对象；第9行代码关闭SqlSession，释放资源。

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



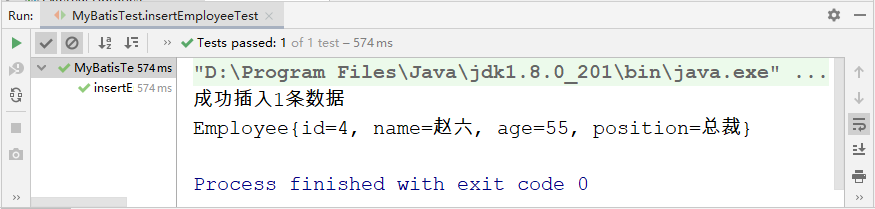
1. findByIdTest ()方法运行结果

（2）在MyBatisTest测试类中添加insertEmployeeTest()方法，该方法用于插入员工信息。insertEmployeeTest()方法具体代码如下所示。

1. @Test
2. public void insertEmployeeTest(){
3. // 通过工具类生成SqlSession对象
4. SqlSession session = MyBatisUtils.getSession();
5. Employee employee = new Employee();
6. employee.setId(4);
7. employee.setName("赵六");
8. employee.setAge(55);
9. employee.setPosition("总裁");
10. int result =
11. session.insert("com.itheima.mapper.EmployeeMapper.addEmployee",employee);
12. if(result>0){
13. System.out.println("成功插入"+result+"条数据");
14. }else {
15. System.out.println("插入数据失败");
16. }
17. System.out.println(employee.toString());
18. session.commit();
19. // 关闭SqlSession
20. session.close();
21. }

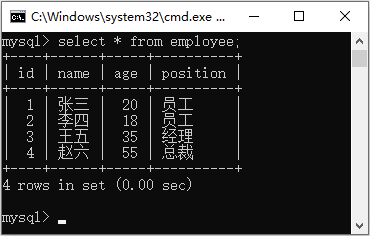
上述代码中，第4行代码通过MyBatisUtils工具类获取SqlSession对象；第5~9行代码创建了一个Employee对象并对其进行赋值；第10~11行代码执行SqlSession的插入方法insert()，插入新的员工信息，并返回插入记录的条数；第12~16行代码是判断数据是否插入成功；第17行代码打印插入的Employee对象；第18~20行代码提交和关闭SqlSession。

执行MyBatisTest测试类的insertEmployeeTest ()方法，控制台的输出结果如图2-3所示。



1. insertEmployeeTest()方法运行结果

由图2-3可知，控制台提示成功插入了1条数据。为了验证插入操作是否真的执行成功，此时查看employee表中的数据，如图2-4所示。



1. employee表中的数据（2）

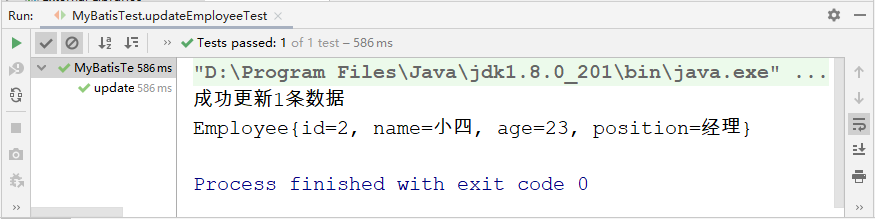
分析上图可知，程序成功插入了id为4的员工的信息。

（3）在MyBatisTest测试类中添加updateEmployeeTest()方法，该方法用于更新员工信息。updateEmployeeTest()方法具体代码如下所示。

1. @Test
2. public void updateEmployeeTest(){
3. // 通过工具类生成SqlSession对象
4. SqlSession session = MyBatisUtils.getSession();
5. Employee employee = new Employee();
6. employee.setId(2);
7. employee.setName("小四");
8. employee.setAge(23);
9. employee.setPosition("经理");
10. int result =
11. session.update("com.itheima.mapper.EmployeeMapper.updateEmployee",
12. employee);
13. if(result>0){
14. System.out.println("成功更新"+result+"条数据");
15. }else {
16. System.out.println("更新数据失败");
17. }
18. System.out.println(employee.toString());
19. session.commit();
20. // 关闭SqlSession
21. session.close();
22. }

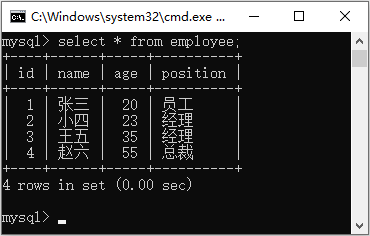
上述代码中，第4行代码通过MyBatisUtils工具类获取SqlSession对象；第5~9行代码创建一个Employee对象并对其进行赋值；第10~12行代码是执行SqlSession的更新方法update()，更新员工信息，并返回更新记录的条数；第13~17行代码判断数据是否更新成功；第18行代码打印更新的Employee对象；第19~21行代码提交和关闭SqlSession。

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



1. updateEmployeeTest()方法运行结果

由图2-5可知，控制台提示成功更新了1条数据。为了验证更新操作是否真的执行成功，此时查看数据库employee表中的数据，如图2-6所示。



1. employee表中的数据（3）

由图2-6分析可知，id为2的员工信息已经更新成功。

（4）在MyBatisTest测试类中添加deleteEmployeeTest()方法，该方法用于删除员工信息。deleteEmployeeTest()方法具体代码如下所示。

1. @Test
2. public void deleteEmployeeTest(){
3. // 通过工具类生成SqlSession对象
4. SqlSession session = MyBatisUtils.getSession();
5. int result =
6. session.delete("com.itheima.mapper.EmployeeMapper.deleteEmployee",4);
7. if(result>0){
8. System.out.println("成功删除"+result+"条数据");
9. }else {
10. System.out.println("删除数据失败");
11. }
12. session.commit();
13. // 关闭SqlSession
14. session.close();
15. }

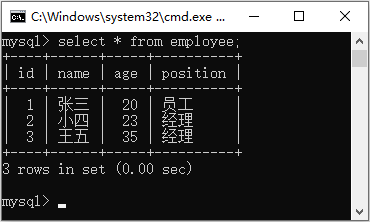
上述代码中，第4行代码通过MyBatisUtils工具类获取SqlSession对象；第5~6行代码执行SqlSession的删除方法delete()，删除员工信息，并返回删除记录的条数；第7~11行代码判断数据是否删除成功；第12~14行代码提交和关闭SqlSession。

执行MyBatisTest测试类的deleteEmployeeTest()方法，控制台的输出结果如图2-7所示。



1. deleteEmployeeTest()方法运行结果

由图2-7，控制台提示成功删除了1条数据。为了验证删除操作是否真的执行成功，此时查看数据库employee表中的数据，如图2-8所示。



1. employee表中的数据（4）

由图2-8分析可知，id为4的员工信息已经删除成功。