第9章 MyBatis的注解开发

湖南科技大学
计算机科学与工程学院

学习目标/Target

- 掌握基于注解的单表增删改查
- 熟悉基于注解的一对一关联查询
- 熟悉基于注解的一对多关联查询
- 熟悉基于注解的多对多关联查询

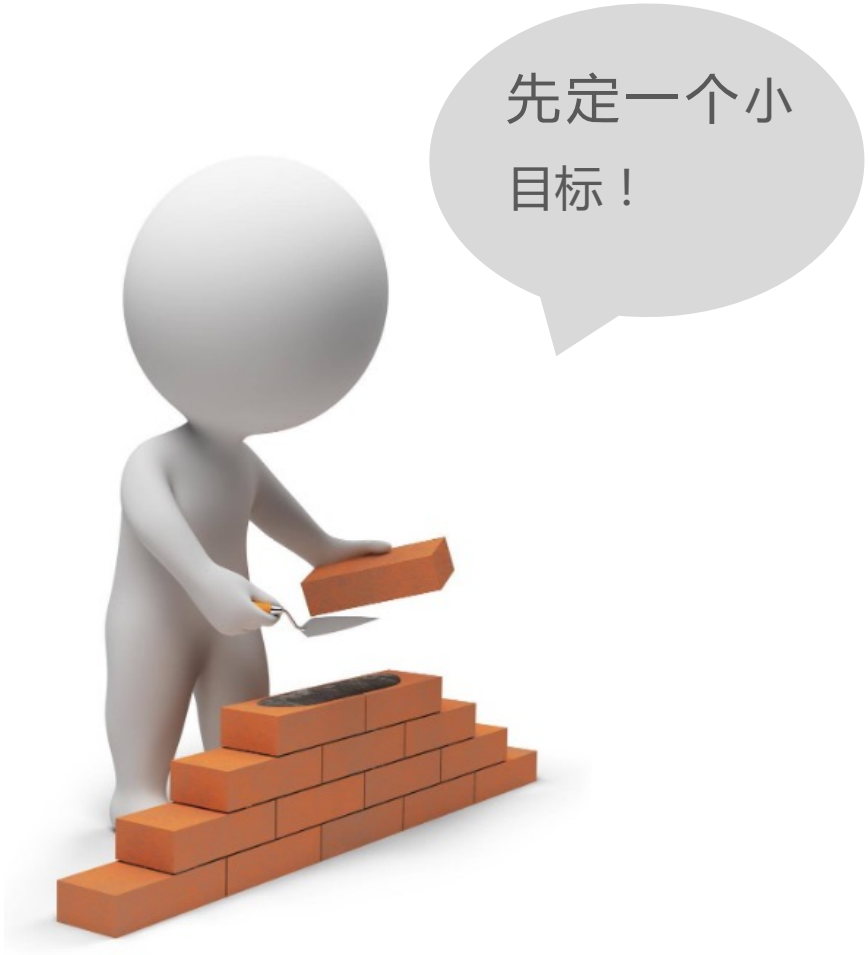
章节概述/ Summary

前面的章节介绍了MyBatis的基本用法、关联映射、动态SQL和缓存机制等知识，所有的配置都是基于XML文件完成的，但在实际开发中，大量的XML配置文件的编写是非常繁琐的，为此，MyBatis提供了更加简便的基于注解的配置方式。本章将对MyBatis的注解开发进行详细讲解。

目录/Contents

9.1 基于注解的单表增删改查

9.1.1 @Select注解



- 掌握@Select注解，能够使用@Select注解进行查询

9.1.1 @Select注解

下面通过一个案例演示@Select注解的使用，该案例要求根据员工的id查找员工信息，案例具体实现步骤如下。

STEP 01

建表：在mybatis数据库中创建名为tb_worker的数据表，同时预先插入几条测试数据。

```
CREATE TABLE  tb_worker(
    id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(32),       age INT,   sex VARCHAR(8),
    worker_id INT UNIQUE
);
INSERT INTO tb_worker(name,age,sex,worker_id)VALUES('张三',32,'女
',1001);
...
```

9.1.1 @Select注解

STEP 02

创建类：创建持久化类Worker，在Worker类中定义id、员工姓名、年龄、性别、工号等属性以及属性对应的getter/setter方法。

```
public class Worker {
    private Integer id;   private String name;     private Integer age;
    private String sex;
    private String worker_id;
    // 省略getter/setter方法
    @Override
    public String toString() {
       return "Worker{" + "id=" + id + ", name=" + name +
  ", age=" + age + ", sex=" + sex + ", worker_id=" + worker_id + '}';
}}
```

9.1.1 @Select注解

STEP 03

编写查询方法：创建WorkerMapper接口，用于编写@Select注解映射的select查询方法。

```
package com.itheima.dao;
import com.itheima.pojo.Worker;
import org.apache.ibatis.annotations.Select;
public interface WorkerMapper {
    @Select("select * from tb_worker where id = #{id}")
    Worker selectWorker(int id);
}
```

9.1.1 @Select注解

STEP 04

加载配置文件：在核心配置文件mybatis-config.xml中的<mappers>元素下引入WorkerMapper接口，将WorkerMapper.java接口加载到核心配置文件中。

```
<mapper class="com.itheima.dao.WorkerMapper"/>
```

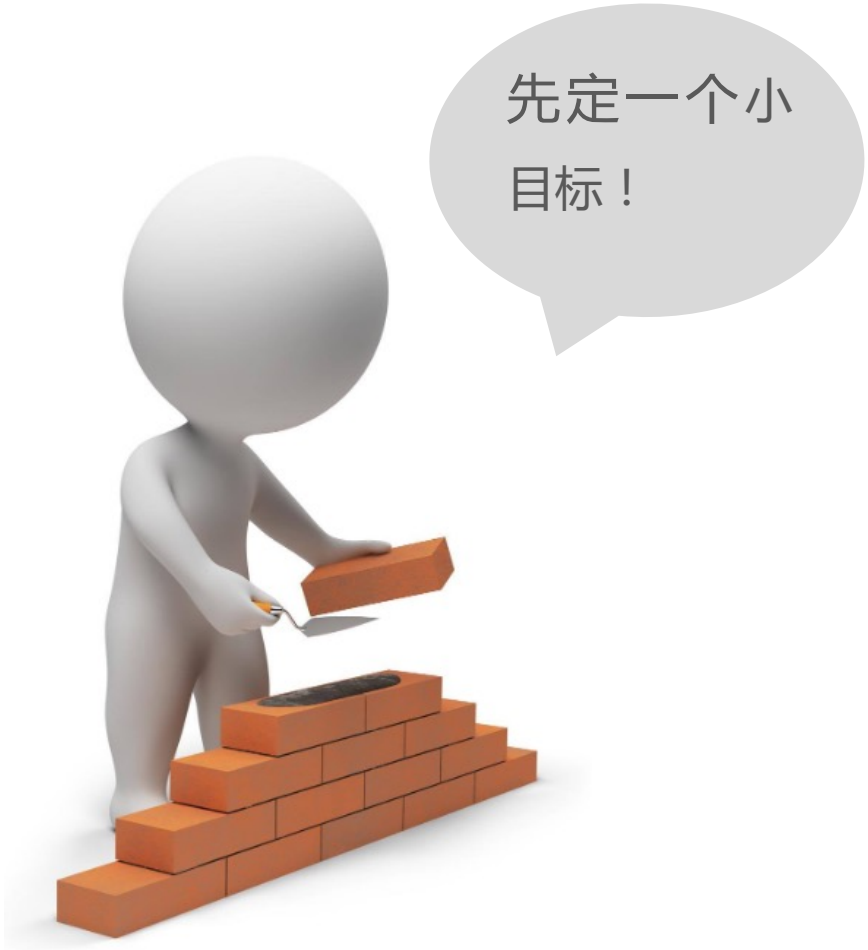
9.1.1 @Select注解

STEP 05

编写测试方法：在测试类MyBatisTest中，编写测试方法findWorkerByIdTest()。

```
public void findWorkerByIdTest() {
    // 1.获取SqlSession对象
    SqlSession session = MyBatisUtils.getSession();
    WorkerMapper mapper = session.getMapper(WorkerMapper.class);
    // 2.查询id为1的员工信息
    Worker worker = mapper.selectWorker(1);
    System.out.println(worker.toString());
    // 3.关闭SqlSession
    session.close();
}
```

9.1.2 @Insert注解



- 掌握@Insert注解，能够使用@Insert注解进行数据插入

9.1.2 @Insert注解

下面通过一个案例演示@Insert注解的使用，要求实现员工信息的插入，案例具体实现步骤如下。

STEP 01

添加注解：在WorkerMapper接口中添加向tb_worker数据表插入数据的方法insertWorker()，并在方法上添加@Insert注解。

```
@Insert("insert into tb_worker(name,age,sex,worker_id)"
        +"values(#{name},#{age},#{sex},#{worker_id})")
    int insertWorker(Worker worker);
```

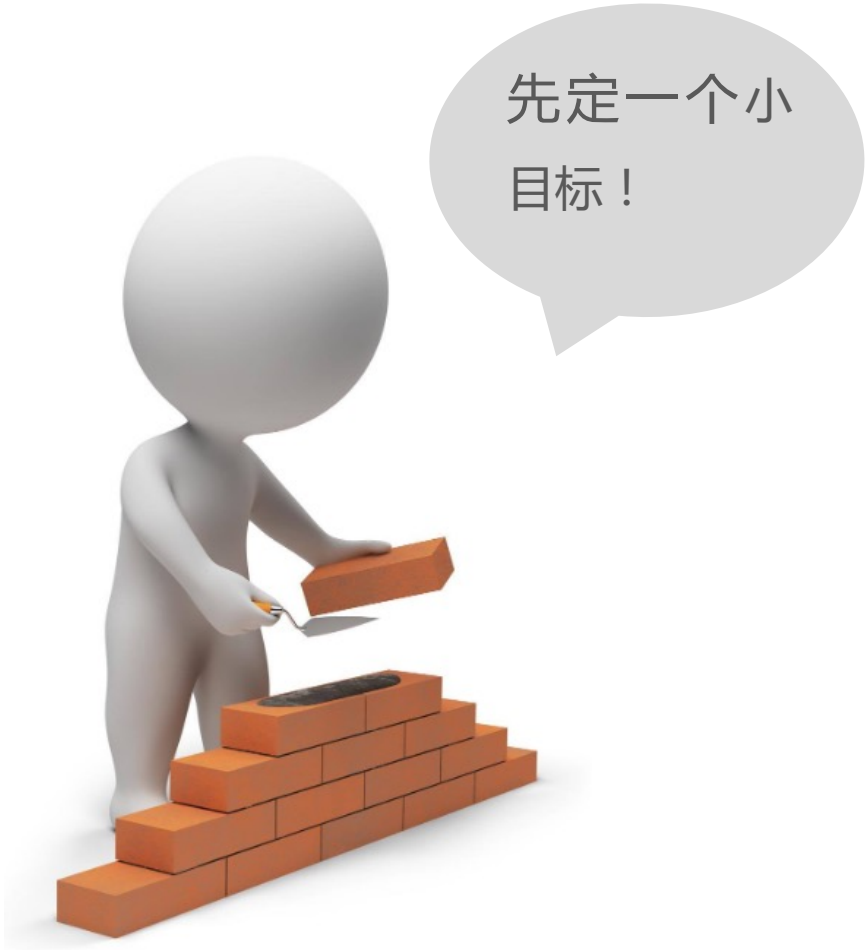
9.1.2 @Insert注解

STEP 02

编写测试类：在测试类MyBatisTest中，编写测试方法insertWorkerTest()。

```
public void insertWorkerTest() {
    // 1.生成SqlSession对象
    SqlSession session = MyBatisUtils.getSession();
    Worker worker = new Worker();
    worker.setId(4);  worker.setName("赵六");       worker.setAge(36);
    worker.setSex("女");          worker.setWorker_id("1004");
    WorkerMapper mapper = session.getMapper(WorkerMapper.class);
    // 2.插入员工信息
    int result = mapper.insertWorker(worker);
    // 输出语句省略...
    session.commit();             session.close(); // 3.关闭SqlSession
}
```

9.1.3 @Update注解



- 掌握@Update注解，能够使用@Update注解进行数据更新

9.1.3 @Update注解

下面通过一个案例演示@Update注解的使用，该案例要求实现员工信息的修改，案例具体实现步骤如下。

STEP 01

添加注解：在WorkerMapper接口中添加更新tb_worker表中数据的方法，并在方法上添加@Update注解。

```
@Update("update tb_worker set name = #{name},age = #{age} "
+"where id = #{id}")
int updateWorker(Worker worker);
```

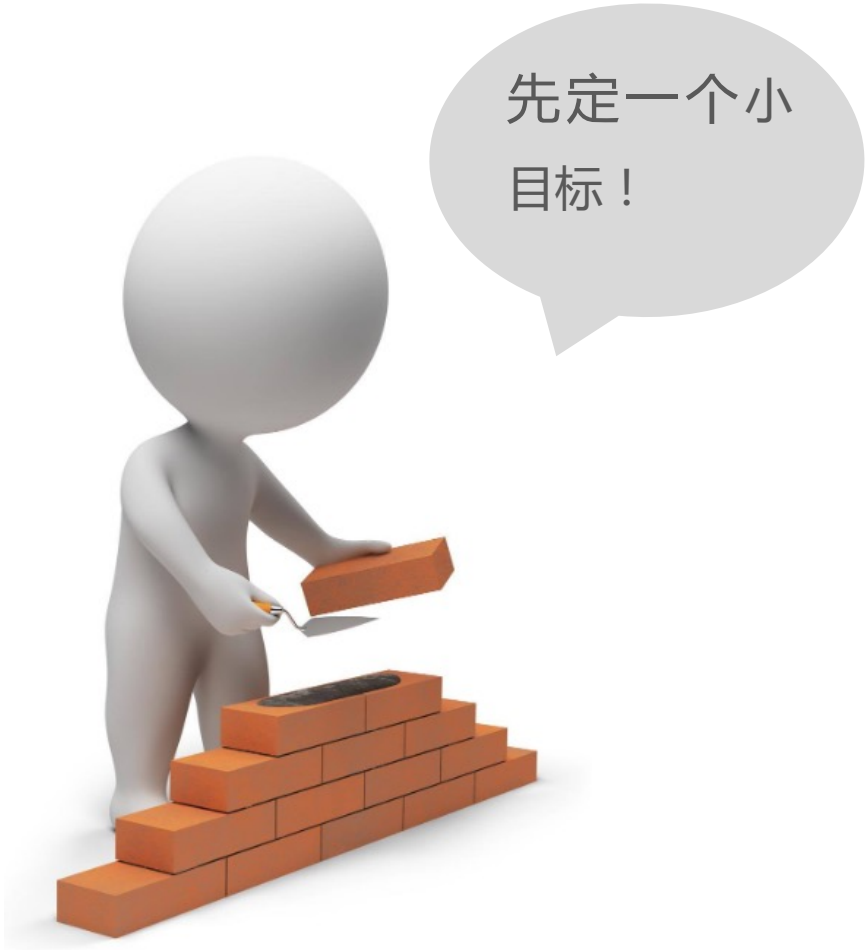
9.1.3 @Update注解

STEP 02

编写测试类：测试类MyBatisTest中，编写测试方法updateWorkerTest()。

```
public void updateWorkerTest() {
    // 1.生成SqlSession对象
    SqlSession session = MyBatisUtils.getSession();
    Worker worker = new Worker();
    worker.setId(4);    worker.setName("李华");
    worker.setAge(28);
    WorkerMapper mapper = session.getMapper(WorkerMapper.class);
    // 2.更新员工信息
    int result = mapper.updateWorker(worker);
   // 输出语句省略...
    session.commit();
  session.close(); // 3.关闭SqlSession
}
```

9.1.4 @Delete注解



- 掌握@Delete注解，能够使用@Delete注解删除数据

9.1.4 @Delete注解

下面通过一个案例演示@Delete注解的使用，该案例要求实现员工信息的删除，案例具体实现步骤如下。

STEP 01

添加注解：在WorkerMapper接口中添加删除数据库中数据的方法，并在方法上添加@Delete注解。

```
@Delete("delete from tb_worker where id = #{id}")
int deleteWorker(int id);
```

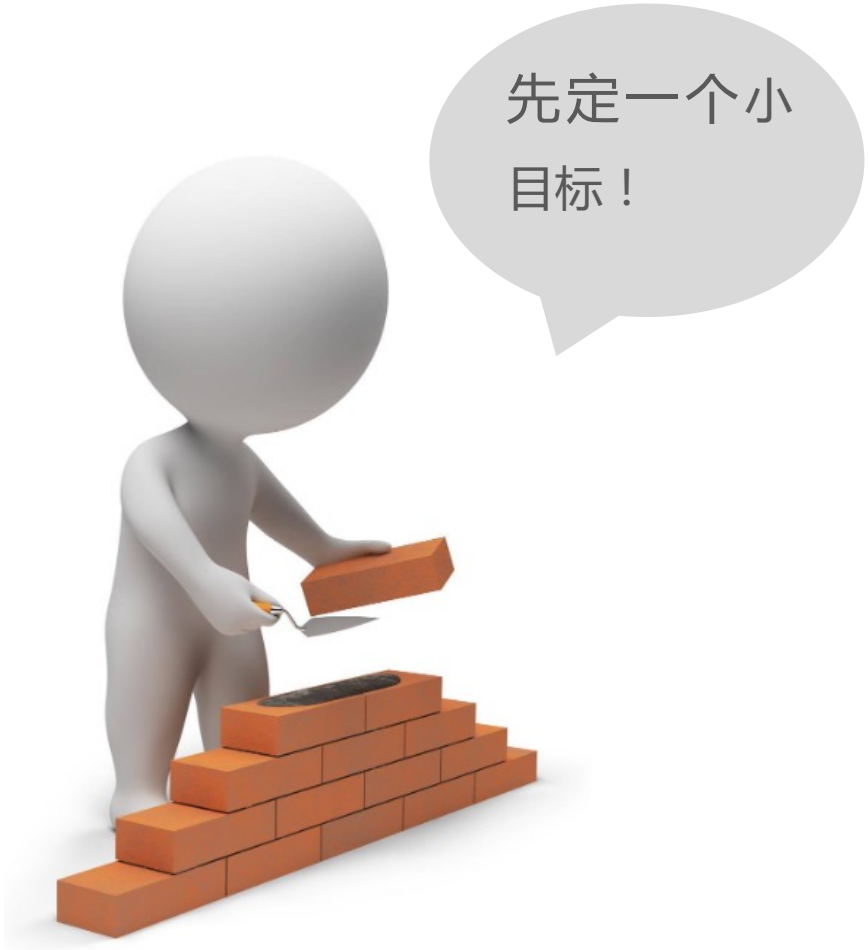
9.1.4 @Delete注解

STEP 02

编写测试类：在测试类MyBatisTest中，编写测试方法deleteWorkerTest()。

```
public void deleteWorkerTest() {
    SqlSession session = MyBatisUtils.getSession(); // 1.生成SqlSession对象
    WorkerMapper mapper = session.getMapper(WorkerMapper.class);
    // 2.删除员工信息
    int result = mapper.deleteWorker(4);
    if(result>0){
        System.out.println("成功删除"+result+"条数据");
    }else { System.out.println("删除数据失败");}
    session.commit();
   session.close(); // 3.关闭SqlSession
}
```

9.1.5 @Param注解



- 掌握@Param注解，能够运用@Param注解制定SQL语句中的参数

9.1.5 @Param注解

下面通过一个案例演示@Param注解的使用，该案例要求根据员工的id和姓名查询员工信息，案例具体实现步骤如下。

STEP 01

添加注解：在WorkerMapper接口中添加多条件查询的方法。

```
@Select("select * from tb_worker where id = #{param01}
                                and name = #{param02}")
Worker selectWorkerByIdAndName(@Param("param01") int id,
                               @Param("param02") String name);
```

9.1.5 @Param注解

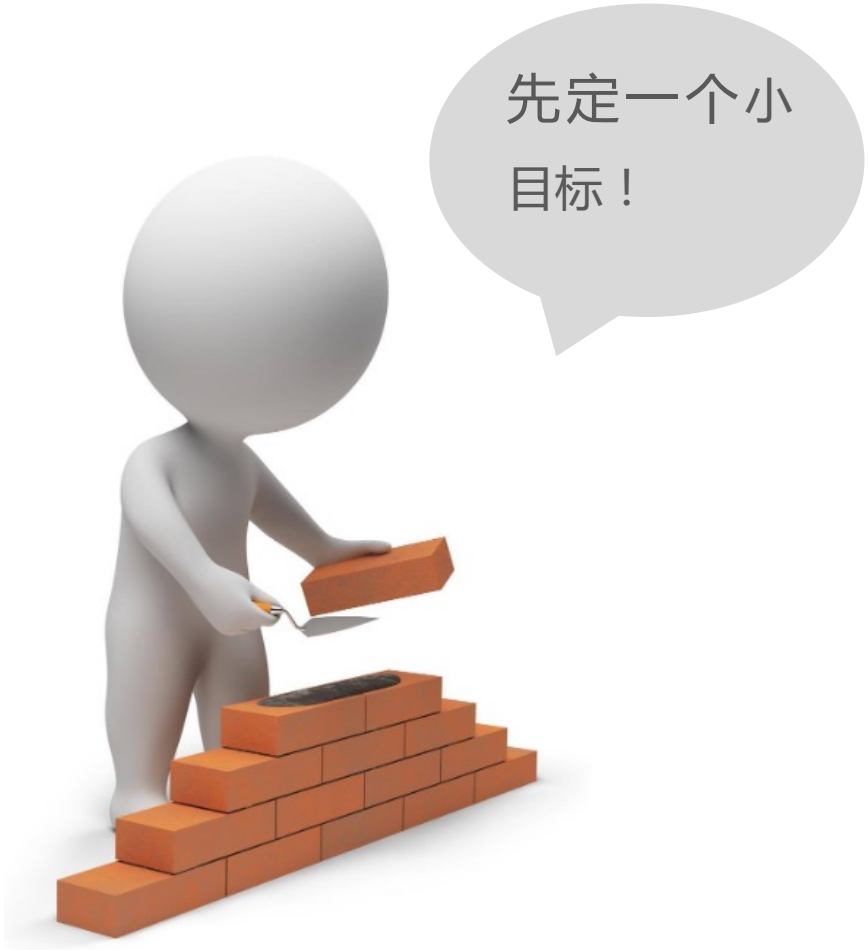
STEP 02

编写测试类：在测试类MyBatisTest中，编写测试方法selectWorkerByIdAndNameTest()。

```
public void selectWorkerByIdAndNameTest() {
    // 1.通过工具类生成SqlSession对象
    SqlSession session = MyBatisUtils.getSession();
    WorkerMapper mapper = session.getMapper(WorkerMapper.class);
    // 2.查询id为3姓名为王五的员工的信息
    Worker worker = mapper.selectWorkerByIdAndName(3,"王五");
    System.out.println(worker.toString());
    session.close();
}
```

9.2 基于注解的关联查询

9.2.1 一对一查询



- 熟悉一对一查询，能够在MyBatis中使用@One注解进行一对一关联查询

9.2.1 一对一查询

接下来，以4.2节中使用的tb_idcard和tb_person数据表为例，详细讲解基于注解@One实现tb_idcard和tb_person数据表之间的一对一关联查询，具体步骤如下。

STEP 01

创建持久化类：本案例使用4.2节中的IdCard类和Person类作为持久类。

9.2.1 一对一查询

STEP 02

编写接口方法：（1）创建IdCardMapper接口，在该接口中编写selectIdCardById()方法，通过id查询人员对应的身份证信息。

```
package com.itheima.dao;
import com.itheima.pojo.IdCard;
import org.apache.ibatis.annotations.Select;
public interface IdCardMapper {
    @Select("select * from tb_idcard where id=#{id}")
    IdCard selectIdCardById(int id);
}
```

9.2.1 一对一查询

STEP 02

编写接口方法：（2）在项目的com.itheima.dao包下创建PersonMapper接口，在该接口中编写selectPersonById()，通过id查询人员信息。

```
package com.itheima.dao;
public interface PersonMapper {
    @Select("select * from tb_person where id=#{id}")
    @Results({@Result(column = "card_id",property = "card",
            one = @One(select =
"com.itheima.dao.IdCardMapper.selectIdCardById"))})
    Person selectPersonById(int id);
}
```

9.2.1 一对一查询

@Result注解的三个属性及含义

（1）property属性用来指定关联属性，这里为card。

（2）column属性用来指定关联的数据库表中的字段，这里为card_id。

（3）one属性用来指定数据表之间属于哪种关联关系，通过@One注解表明数据表tb_idcard和tb_person之间是一对一关联关系。

9.2.1 一对一查询

STEP 03

引入接口：在核心配置文件mybatis-config.xml中的<mappers>元素下引入IdCardMapper和PersonMapper接口。

```
<mapper class="com.itheima.dao.IdCardMapper"/>
<mapper class="com.itheima.dao.PersonMapper"/>
```

9.2.1 一对一查询

<mappers>元素引入XML文件顺序

由于mybatis-config.xml文件中的扫描方式是从上往下扫描，所以<mappers>元素下引入IdCardMapper和PersonMapper接口的位置，必须在引入IdCardMapper.xml和PersonMapper.xml文件位置前面，否则程序将会首先读取到引入的IdCardMapper.xml和PersonMapper.xml文件，程序将会报错。

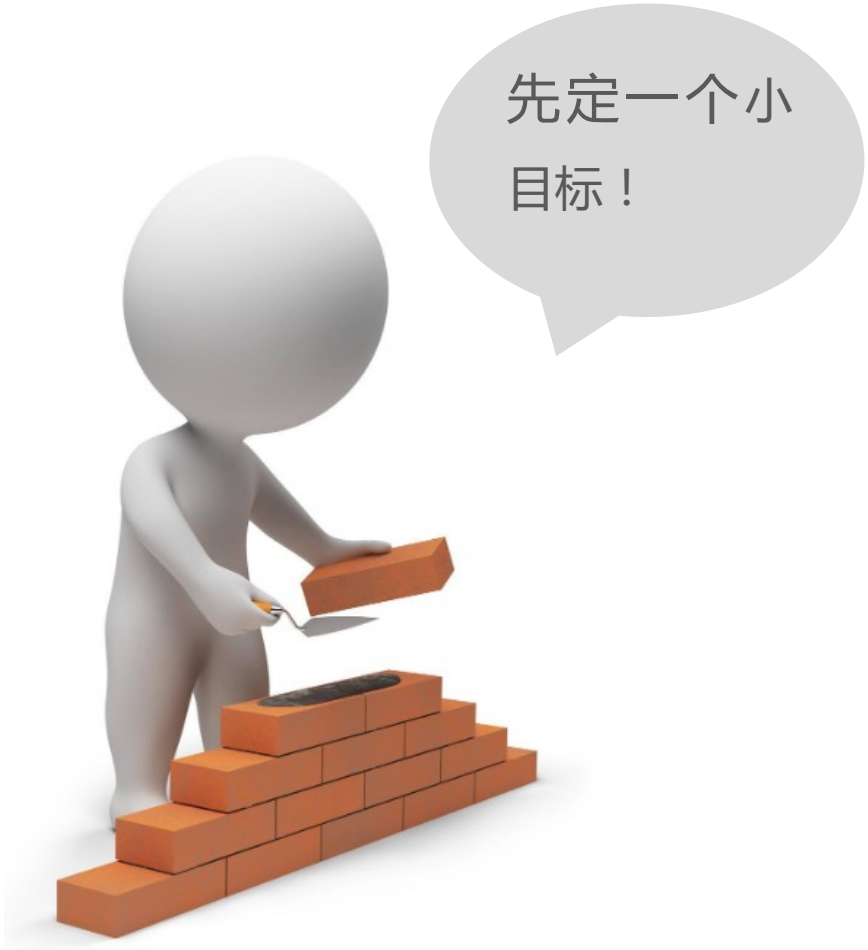
9.2.1 一对一查询

STEP 04

编写测试类：在测试类MyBatisTest中，编写测试方法selectPersonByIdTest()。

```
public void selectPersonByIdTest() {
    // 1.通过工具类生成SqlSession对象
    SqlSession session = MyBatisUtils.getSession();
    PersonMapper mapper = session.getMapper(PersonMapper.class);
    // 2.查询id为1的人员的信息
    Person person = mapper.selectPersonById(2);
    System.out.println(person.toString());
    session.close(); // 3.关闭SqlSession
}
```

9.2.2 一对多查询



- 熟悉一对多查询，能够在MyBatis中使用@Many注解进行一对多关联查询

9.2.2 一对多查询

接下来，以4.3节中的tb_user和tb_orders数据表为例，详细讲解基于@Many注解配置实现tb_user和tb_orders数据表之间的一对多关联查询，具体步骤如下。

STEP 01

创建持久化类：本案例使用4.3节中的Users类和Orders类作为持久类。

9.2.2 一对多查询

STEP 02

编写接口方法：（1）创建OrdersMapper接口，在该接口中编写selectOrdersByUserId()方法，通过user_id查询用户对应的订单信息。

```
public interface OrdersMapper {
    @Select("select * from tb_orders where user_id=#{id} ")
    @Results({@Result(id = true,column = "id",property = "id"),
            @Result(column = "number",property = "number") })
    List<Orders> selectOrdersByUserId(int user_id);
}
```

9.2.2 一对多查询

STEP 02

编写接口方法：（2）创建UsersMapper接口，在该接口中编写selectUserById()方法，通过id查询用户信息。

```
public interface UsersMapper {
    @Select("select * from tb_user where id=#{id} ")
    @Results({@Result(id = true,column = "id",property = "id"),
            @Result(column = "username",property = "username"),
            @Result(column = "address",property = "address"),
            @Result(column = "id",property = "ordersList",
                    many = @Many(select =
"com.itheima.dao.OrdersMapper.selectOrdersByUserId"))})
    Users selectUserById(int id);
}
```

9.2.2 一对多查询

STEP 03

引入接口：在核心配置文件mybatis-config.xml中的<mappers>元素下引入UsersMapper和OrdersMapper接口。

```
<mapper class="com.itheima.dao.UsersMapper"/>
<mapper class="com.itheima.dao.OrdersMapper"/>
```

9.2.2 一对多查询

<mappers>元素引入XML文件顺序

由于mybatis-config.xml文件中的扫描方式是从上往下扫描，所以<mappers>元素下引入UsersMapper和OrdersMapper接口的位置，必须在引入UsersMapper.xml和OrdersMapper.xml文件位置前面，否则程序将会首先读取到引入的UsersMapper.xml和OrdersMapper.xml文件，程序将会报错。

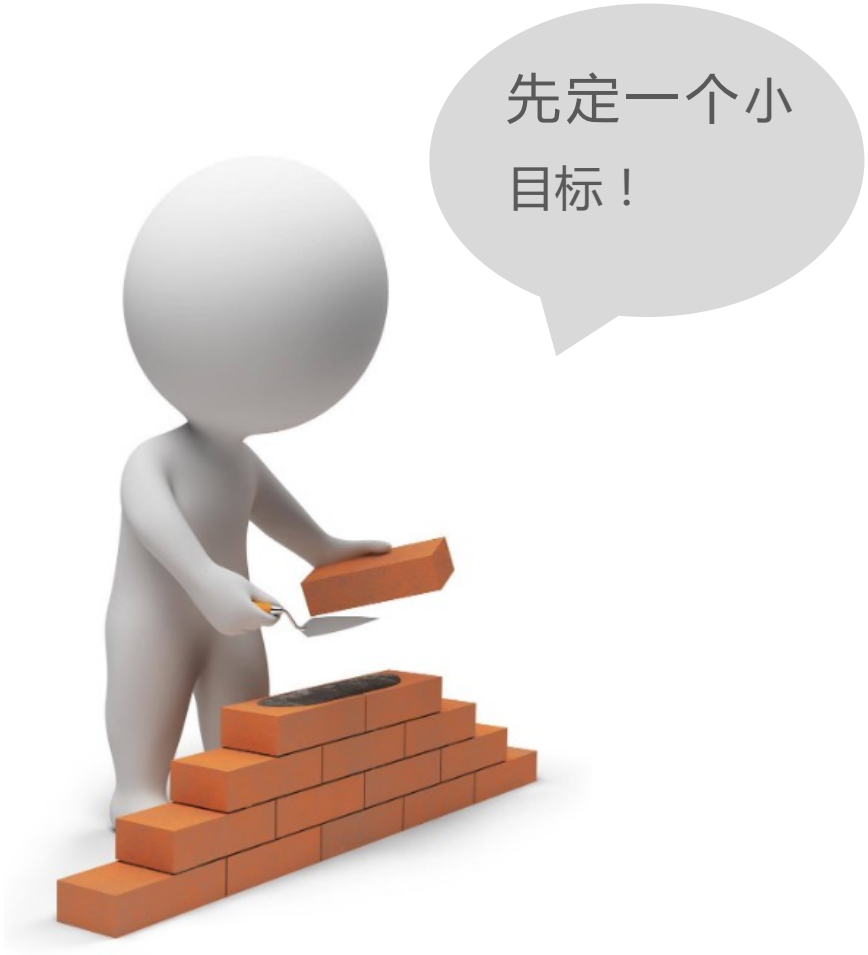
9.2.2 一对多查询

STEP 04

编写测试类：在测试类MyBatisTest中，编写测试方法selectUserByIdTest()。

```
public void selectUserByIdTest() {
    // 1.通过工具类生成SqlSession对象
    SqlSession session = MyBatisUtils.getSession();
    UsersMapper mapper = session.getMapper(UsersMapper.class);
    // 2.查询id为1的人的信息
    Users users = mapper.selectUserById(1);
    System.out.println(users.toString());
   session.close();
}
```

9.2.3 多对多查询



- 熟悉多对多查询，能够在MyBatis中进行多对多关联查询

9.2.3 多对多查询

多对多关联使用中间表

在数据库中，表与表之间的多对多关联关系通常使用一个中间表来维护，以4.4节中使用的订单表tb_orders和商品表tb_product为例，这两个表之间的关联关系使用了一个中间表tb_ordersitem来维护，订单表tb_orders和商品表tb_product，都与中间表tb_ordersitem形成了一对多关联关系，即中间表tb_ordersitem将订单表tb_orders和商品表tb_product拆分成了两个一对多的关联关系。

9.2.3 多对多查询

接下来，以4.4节中使用的订单表tb_orders、商品表tb_product和中间表tb_ordersitem为例，详细讲解多对多关联查询，具体步骤如下。

STEP 01

在订单持久化类（Orders.java）中增加商品集合的属性及其对应的getter/setter方法，并修改Orders类和Product类中的toString()方法。

```
@Override
public String toString() {return "Orders{" +
    "id=" + id + ", number=" + number + ", productList=" + productList + '}’  ;}
@Override
public String toString() {return "Product{" +
    "id=" + id + ", name=" + name +", price=" + price + '}';
}
```

9.2.3 多对多查询

STEP 02

（1）创建ProductMapper接口，在该接口中编写selectProductByOrdersId()方法，通过user_id查询用户对应的订单信息。

```
public interface ProductMapper {
    @Select("select * from tb_product where id in
            (select product_id from  tb_ordersitem where orders_id = #{id} )")
    List<Product> selectProductByOrdersId(int orders_id);
}
```

9.2.3 多对多查询

STEP 02

（2）在OrdersMapper接口中添加selectOrdersById()方法，该方法用于通过id查询订单信息。

```
@Select("select * from tb_orders where id=#{id} ")
    @Results({@Result(id = true,column = "id",property = "id"),
            @Result(column = "number",property = "number"),
            @Result(column = "id",property = "productList",many = @Many(select =
"com.itheima.dao.ProductMapper.selectProductByOrdersId"))})
    Orders selectOrdersById(int id);
```

9.2.3 多对多查询

STEP 03

在核心配置文件mybatis-config.xml中的<mappers>元素下引入ProductMapper和OrdersMapper接口，将这两个接口加载到核心配置文件中。

```
<mapper class="com.itheima.dao.ProductMapper"/>
<mapper class="com.itheima.dao.OrdersMapper "/>
```

9.2.3 多对多查询

<mappers>元素引入XML文件的顺序

注意：由于mybatis-config.xml文件中的扫描方式是从上往下扫描，所以<mappers>元素下引入ProductMapper和OrdersMapper接口的位置，必须在引入ProductMapper.xml和OrdersMapper.xml文件位置前面，否则程序将会首先读取到引入的ProductMapper.xml和OrdersMapper.xml文件，程序将会报错。

9.2.3 多对多查询

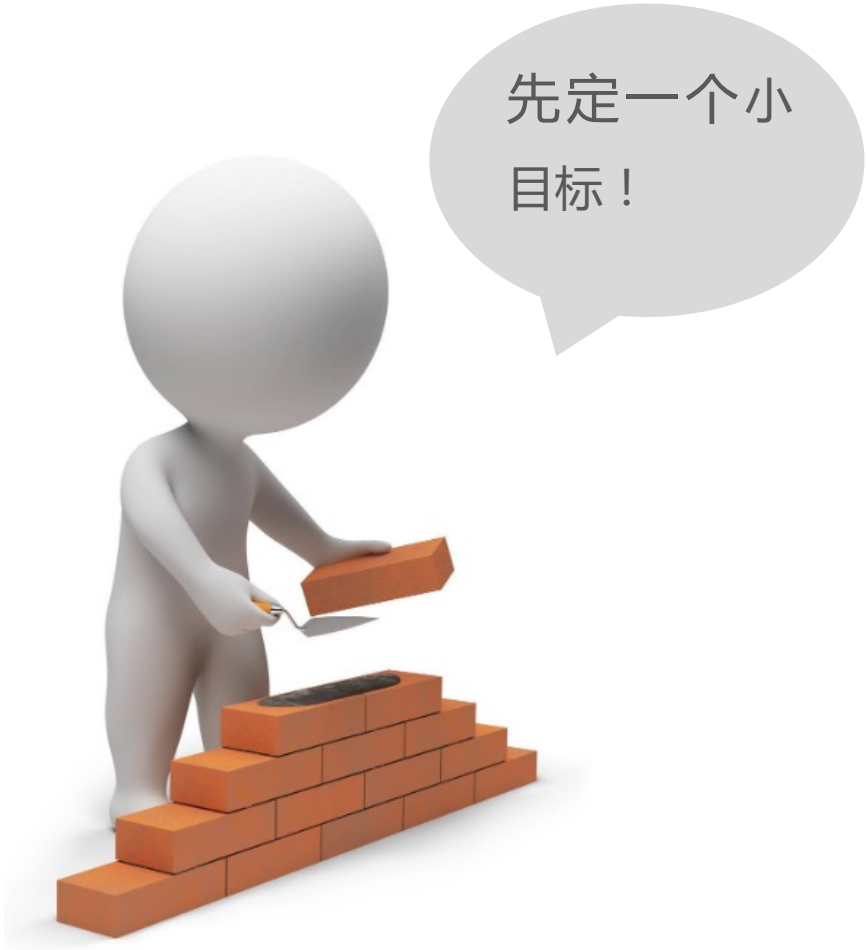
STEP 04

在测试类MyBatisTest中，编写测试方法selectOrdersByIdTest()，查询id为3的订单的信息。

```
public void selectOrdersByIdTest() {
    // 1.通过工具类生成SqlSession对象
    SqlSession session = MyBatisUtils.getSession();
    OrdersMapper mapper = session.getMapper(OrdersMapper.class);
    // 2.查询id为3的订单的信息
    Orders orders = mapper.selectOrdersById(3);
    System.out.println(orders.toString());
    session.close();
}
```

9.3 案例：基于MyBatis注解的学生管理程序

9.3 案例：基于MyBatis注解的学生管理程序



- 完成基于MyBatis注解的学生管理程序，能够用MyBatis注解实现查询操作、实现修改操作、实现一对多查询

9.3 案例：基于MyBatis注解的学生管理程序

学生表(s_student)与班级表(c_class)详情

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

班级id	班级名称 classname
1	一班
2	二班

9.3 案例：基于MyBatis注解的学生管理程序

使用MyBatis注解实现下列要求

（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。

（2）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。

（3）MyBatis注解实现一对多查询。查询出二班所有学生的信息。

9.3 案例：基于MyBatis注解的学生管理程序

MyBatis注解实现查询操作

STEP 01

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。

9.3 案例：基于MyBatis注解的学生管理程序

STEP 02

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。

```
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));
INSERT INTO c_class VALUES (1, '一班');
INSERT INTO c_class VALUES (2, '二班' );
# 创建s_student表同理
```

9.3 案例：基于MyBatis注解的学生管理程序

STEP 03

POJO类准备：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。

```
public class IClass {
    private Integer id;          private String classname;    // 主键id，班级名称
    private List<IStudent> studentList;    // 学生集合
    // 省略getter/setter方法
    @Override
    public String toString() {return "IClass{" +"id=" + id +
                ", classname='" + classname +", studentList=" + studentList + '}' ;
}}
```

9.3 案例：基于MyBatis注解的学生管理程序

STEP 04

创建Mapper接口：创建IStudentMapper接口，用于编写@Select注解映射的select查询语句。

```
package com.itheima.dao;
import com.itheima.pojo.IStudent;
import org.apache.ibatis.annotations.Select;
public interface IStudentMapper {
    @Select("select * from s_student where id = #{id}")
    IStudent selectStudent(int id);
}
```

9.3 案例：基于MyBatis注解的学生管理程序

STEP 05

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。

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

9.3 案例：基于MyBatis注解的学生管理程序

STEP 06

编写MyBatisUtils工具类：创建MyBatisUtils工具类，该类用于封装读取配置文件信息的代码。

```
public class MyBatisUtils {
    private static SqlSessionFactory sqlSessionFactory = null;
    static {            try {
        Reader reader =
            Resources.getResourceAsReader("mybatis-config.xml");
        sqlSessionFactory =
            new SqlSessionFactoryBuilder().build(reader);
    } catch (Exception e) {    e.printStackTrace();            }}
    public static SqlSession getSession() {
        return sqlSessionFactory.openSession();
}}
```

9.3 案例：基于MyBatis注解的学生管理程序

STEP 07

编写测试方法：创建测试类MyBatisTest，在测试类MyBatisTest中编写测试方法findIStudentByIdTest()。

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

9.3 案例：基于MyBatis注解的学生管理程序

MyBatis注解实现修改操作

STEP 01

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。

```
@Update("update s_student set name = #{name},age =
#{age} "
            +"where id = #{id}")
int updateStudent(IStudent student);
```

9.3 案例：基于MyBatis注解的学生管理程序

STEP 02

编写测试方法：在测试类MyBatisTest中，编写测试方法updateIStudentTest()。

```
public void updateIStudentTest() {
    SqlSession session = MyBatisUtils.getSession();
    IStudent student = new IStudent();
    student.setId(4);   student.setName("李雷");  student.setAge(21);
    IStudentMapper mapper = session.getMapper(IStudentMapper.class);
    int result = mapper.updateStudent(student); // 更新学生信息
    if(result>0){ System.out.println("成功更新"+result+"条数据");
    } else { System.out.println("更新数据失败");}
    System.out.println(student.toString());          session.commit();
    session.close(); // 关闭SqlSession}
```

9.3 案例：基于MyBatis注解的学生管理程序

MyBatis注解实现一对多操作

STEP 01

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。

```
@Select("select * from s_student where cid=#{id} ")
@Results({@Result(id = true,column = "id",property = "id"),
        @Result(column = "classname",property = "classname")
})
List<IStudent> selectStudentByCid(int cid);
```

9.3 案例：基于MyBatis注解的学生管理程序

MyBatis注解实现一对多操作

STEP 01

修改Mapper接口：（2）创建IClassMapper接口，在该接口中编写selectClassById()方法，通过id查询班级信息。

```
public interface IClassMapper {
    @Select("select * from c_class where id=#{id} ")
    @Results({@Result(id = true,column = "id",property = "id"),
            @Result(column = "classname",property = "classname"),
            @Result(column = "id",property = "studentList",
            many =
@Many(select"com.itheima.dao.IStudentMapper.selectStudentByCid"))})
                IClass selectClassById(int id);              }
```

9.3 案例：基于MyBatis注解的学生管理程序

STEP 02

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。

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

9.3 案例：基于MyBatis注解的学生管理程序

STEP 03

编写测试方法：在测试类MyBatisTest中，编写测试方法selectClassByIdTest()。

```
public void selectClassByIdTest() {
    // 1.生成SqlSession对象
    SqlSession session = MyBatisUtils.getSession();
    IClassMapper mapper = session.getMapper(IClassMapper.class);
    // 2.查询id为2的班级中学生的信息
    IClass icalss = mapper.selectClassById(2);
    System.out.println(icalss.toString());
    session.close();
}
```

本章小结

本章小结

本章主要讲解了MyBatis的注解开发。首先讲解了基于注解的单表增删改查常用注解，包括@Select注解、@Insert注解、@Update注解、@Delete注解、@Param注解等；然后讲解了基于注解的关联查询，包括一对一查询、一对多查询和多对多查询。通过本章的学习，读者可以了解MyBatis中常用注解的主要作用，并能够掌握这些注解在实际开发中的应用。在MyBatis框架中，这些注解十分重要，熟练的掌握它们能够极大的提高开发效率。