MyBatis 接口代理方式实现 Dao 层

MyBatis 接口代理方式实现 Dao 层

区别

1、selectlist 和 getMapper 区别

```
//4.执行映射配置文件中的 sql 语句，并接收结果

list = sqlSession.selectList("StudentMapper.selectAll");

//4.获取 StudentMapper 接口的实现类对象

StudentMapper mapper = sqlSession.getMapper(StudentMapper.class);

//5.通过实现类对象调用方法，接收结果

result = mapper.delete(id);
```

两者其实并无区别，只不过 StudentMapper 利用代理对象操作 sqlSession.selectList();

2、namespace 的作用

```
<mapper namespace="StudentMapper">

<mapper namespace="com.zhu.mapper.StudentMapper">
```

1.完全限定名（比如"com.mypackage.MyMapper.selectAllThings"）将被直接查找并且找到即用。
2.短名称（比如"selectAllThings"）如果全局唯一也可以作为一个单独的引用。如果不唯一，有两个或两个以上的相同名称（比如"com.foo.selectAllThings "和"com.bar.selectAllThings"），那么使用时就会收到错误报告说短名称是不唯一的，这种情况下就必须使用完全限定名。
Mybatis 中 namespace 用于绑定 dao 接口，dao 接口的方法对应 mapper 中的 sql 语名

接口代理方式 - 实现规则

1）映射配置文件中的**名称空间**必须和 Dao 层接口的**全类名**相同
2）映射配置文件中的增删改查的**标签 id 属性**必须和 Dao 层接口层的**方法名**相同
3）映射配置文件中的增删改查标签的 **parameterType 属性**必须和 Dao 层接口**方法的参数**相同
4）映射配置文件中的增删改查标签的 **resultType 属性**必须和 Dao 层接口**方法的返回值**相同

接口代理方式 - 代码实现

1）删除 mapper 层接口的实现类
2）修改映射配置文件
3）修改 service 层接口的实现类，采用接口代理方式实现功能

在写 StudentMapper.java 时

注意

1、resultType="student" 方法的返回值为List<Student>

2、parameterType="student"方法的参数为Student student

3、如何取出参数中的值 #{}

- **MyBatisConfig.xml**

```
<?xml version="1.0" encoding="UTF-8" ?>

<!--MyBatis 的 DTD 约束-->

<!DOCTYPE configuration PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-config.dtd">

<configuration>

    <properties resource="jdbc.properties"/>

    <settings>

        <setting name="logImpl" value="log4j"/>

    </settings>

  <!--默认名为类名首字母小写-->

    <typeAliases>

        <package name="com.zhu.bean"/>

    </typeAliases>

    <environments default="mysql">

        <!--environment 配置数据库环境 id 属性唯一标识 -->

        <environment id="mysql">

            <!-- transactionManager 事务管理。type 属性，采用 JDBC 默认的事务-->

            <transactionManager type="JDBC"></transactionManager>

            <!--dataSource 数据源信息 type 属性 连接池 MyBatis 默认有三种数据源-
->

            <dataSource type="POOLED">
```

```
                <!--property 获取数据库连接的配置信息-->
                <property name="driver" value="${driver}"/>
                <property name="url" value="${url}"/>
                <property name="username" value="${username}"/>
                <property name="password" value="${password}"/>
            </dataSource>
        </environment>
    </environments>
    <!-- mappers 引入映射配置文件 -->
    <mappers>
        <!-- mapper 引入指定的映射配置文件  resource 属性指定映射配置文件的名称
-->
        <mapper resource="StudentMapper.xml"/>
    </mappers>
</configuration>
```

· StudentMapper.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!--MyBatis 的 DTD 约束-->
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<!--
    mapper: 核心根标签
    namespace 属性: 名称空间
-->
<mapper namespace="com.zhu.mapper.StudentMapper">
```

```
    <select id="selectAll" resultType="student">
        SELECT * FROM student
    </select>

    <insert id="insert" parameterType="student">
        INSERT INTO student VALUES (#{id},#{name},#{age});
    </insert>

    <delete id="delete" parameterType="int">
        DELETE FROM student WHERE id=#{id};
    </delete>
</mapper>
```

· **controller - studentController.java**

```
public class StudentController {
    private StudentService studentService = new StudentServiceImpl();

    @Test
    public void selectAll(){
        List<Student> list = studentService.selectAll();
        for (Student student : list) {
            System.out.println(student);
        }
    }
```

```
    @Test
    public void insert(){
        Student st = new Student(10,"路飞",23);
        Integer insert = studentService.insert(st);
        System.out.println(insert);
    }
    @Test
    public void delete(){
        Integer s = 3;
        Integer i = studentService.delete(3);
        System.out.println(i);
    }
}
```

· mapper - StudentMapper.java

```
public interface StudentMapper{
    /*
        注意:  resultType 值为 student  指定结果映射的对象类型
                方法的返回值为   List<Student>
        resultType:
                1、基本类型  : resultType=基本类型
                2、List 类型 : resultType=List 中元素的类型
                3、Map 类型  单条记录:resultType =map
                            多条记录:resultType =Map 中 value 的类型
              方法的返回值可以为 List<Student>
        parameterType:
                 基本数据类型: int,string,long,Date;
                 复杂数据类型: 类和 Map
                如何获取参数中的值:
                                基本数据类型: #{value}或${value}  获取参数中的值
```

复杂数据类型：#{属性名}或${属性名} ，map 中则是#{key}或${key}

```
     */
    public abstract List<Student> selectAll();


    Integer insert(Student student);


    Integer delete(Integer integer);
}
```

· service - StduentService.java - StudentServiceImpl.java

```
public interface StudentService {
    List<Student> selectAll();
    Integer insert(Student student);
    Integer delete(Integer integer);
}
public class StudentServiceImpl implements StudentService {
    @Override
    public List<Student> selectAll() {
        List<Student> list = null;
        SqlSession sqlSession = null;
        InputStream is = null;
        try {
            is = Resources.getResourceAsStream("MyBatisConfig.xml");
```

```
            SqlSessionFactory factory = new
SqlSessionFactoryBuilder().build(is);

            sqlSession = factory.openSession(true);



            //获取 StudentMapper 接口的实现类对象

            //获取对应的 Mapper，让映射器通过命名空间和方法名称找到对应的 SQL，发
送给数据库执行后返回结果。

            //操作数据库主要是通过 SQL 语句，那么只要找到 SQL 语句然后执行不就可以

            //sqlSession.getMapper()的内部产生了 StudentMapper 的实现类，那怎么产
生的呢？

            //动态代理

            /*
                被代理对象：真实的对象
                代理对象：内存中的一个对象

             */

            StudentMapper mapper = sqlSession.getMapper(StudentMapper.class);

            //StudentMapper studentMapper1 = new StudentServiceImpl();

            //return
Proxy.newProxyInstance(this.mapperInterface.getClassLoader(), new
Class[]{this.mapperInterface}, mapperProxy);

            /*
                类加载器：和被代理对象使用相同的类加载器
                接口类型 Class 数组：和被代理对象使用相同接口
                代理规则：完成代理增强的功能

             */

            //通过代理对象调用方法，接收结果

            list = mapper.selectAll();
```

```
        } catch (IOException e) {
            e.printStackTrace();
        }finally {
            if(sqlSession != null) {
                sqlSession.close();
            }
            if(is != null) {
                try {
                    is.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }
        return list;
    }
```

接口代理方式 - 问题

分析动态代理对象如何生成的？

通过动态代理开发模式，我们只编写一个接口，不写实现类，我们通过 getMapper() 方法最终获取到 org.apache.ibatis.binding.MapperProxy 代理对象，然后执行功能，而这个代理对象正是 MyBatis 使用了 JDK 的动态代理技术，帮助我们生成了代理实现类对象。从而可以进行相关持久化操作。

分析方法是如何执行的？

动态代理实现类对象在执行方法的时候最终调用了 mapperMethod.execute() 方法，这个方法中通过 switch 语句根据操作类型来判断是新增、修改、删除、查询操作，最后一步回到了 MyBatis 最原生的 SqlSession 方式来执行增删改查。

小结

- 原生：Dao ---> DaoImpl

- 接口：Mapper ---> xxMapper.xml
- 接口代理方式可以让给我们只编写接口即可，而实现类对象由 MyBatis 生成
- 获取动态代理对象

SqlSession 功能类中的 getMapper(Mapper 接口.class)方法。

MyBatis 映射配置文件 - 动态 SQL

"if 标签" +

```
<select id="selectCondition" resultType="student" parameterType="student">
        SELECT * FROM student
        <where>
                <if test="id != null">
                    id = #{id}
                </if>
                <if test="name != null">
                    AND name = #{name}
                </if>
                <if test="age != null">
                    AND age = #{age}
                </if>
        </where>
    </select>
```

"foreach"

foreach 用来迭代用户传过来的 List 或者 Array
如：使用 foreach 元素来构建 in 子语句
collection：指定要遍历的集合名称 list、array
item：用来临时存放迭代集合中当前元素的值，便于在 foreach 中使用
open：将该属性指定的值添加到 foreach 迭代后拼出字符串的开始
close：将该属性指定的值添加到 foreach 迭代拼出字符串的结尾
separator：用来分割 foreach 元素迭代的每个元素

```
<!--SELECT * FROM student WHERE id IN (1,2);

-->

    <select id="selectByIds" resultType="student" parameterType="list">

        SELECT * from student

        <where>

            <foreach collection="list" open="id IN(" close=")" item="id"
separator=",">

                #{id}

            </foreach>

        </where>

    </select>


    <select id="selectByIds2" resultType="student" parameterType="list">

        SELECT * from student where id IN

            <foreach collection="list" open="(" close=")" item="id"
separator=",">

                #{id}

            </foreach>

    </select>


<!--如果是 selectById(List<Student> stus) 这种-->

    <select id="selectByIds3" resultType="student" parameterType="list">

        SELECT * from student

            <where>
```

```
                <foreach collection="list" open="id IN(" close=")" item="stu"
separator=",">
                    #{stu.id}
                </foreach>
            </where>
    </select>
```

使用

StudentMapper.java

```
List<Student> selectByIds(List<Integer> list);
```

StudentService.java

```
List<Student> selectByIds(List<Integer> list);
```

sql 片段抽取

```
<sql id="select">SELECT * FROM student</sql>
```

使用

```
<include refid="select"/>
```

小结

- 动态 SQL 指的就是 SQL 语句可以根据条件或者参数的不同进行动态的变化。
- ：条件标签。
- ：条件判断的标签。
- ：循环遍历的标签。
- ：抽取 SQL 片段的标签。
- ：引入 SQL 片段的标签。

MyBatis 核心配置文件 - 分页查询

使用 PageHelper

功能实现

(1) 导入 jar 包 分页插件 pagehelper-5.1.10.jar 和依赖的包 jsqlparser-3.1.jar

(2) 在核心配置文件中集成分页助手插件

(3) 在测试类中使用分页助手相关 API 实现分页功能。

MyBatisConfig.xml

```
<!-- 环境配置顺序
<!ELEMENT configuration (properties?, settings?, typeAliases?, typeHandlers?,
objectFactory?,
objectWrapperFactory?, reflectorFactory?, plugins?, environments?,
databaseIdProvider?, mappers?)>

-->

<plugins>

    <plugin interceptor="com.github.pagehelper.PageInterceptor"></plugin>

</plugins>
```

使用

StudentServiceImpl.java

PageHelper.startPage(1,3); 一定要写在 SQL 执行之前

```
//通过分页助手来实现分页功能

// 第一页：显示 3 条数据

//PageHelper.startPage(1,3);

// 第二页：显示 3 条数据

//PageHelper.startPage(2,3);

// 第三页：显示 3 条数据

PageHelper.startPage(3,3);


//5.调用实现类的方法，接收结果

List<Student> list = mapper.selectAll();
```

相关 API

返回值	方法名	说明
long	getTotal()	获取总条数

返回值	方法名	说明
int	getPages()	获取总页数
int	getPageNum()	获取当前页
int	getPageSize()	获取每页显示条
int	getPrePage()	获取上一页
int	getNextPage()	获取下一页
boolean	isIsFirstPage()	获取是否是第一
boolean	isIsLastPage()	获取是否是最后一

MyBatis 多表操作

外键字段的建立是体现表与表关系的所在

- 一对一 ： 在任意一方建立外键，关联对方的主键

用户基本信息表 和 用户详细信息表

- 一对多 ： 在多的一方建立外键，关联一的一方主键

店铺表 和 商品表

用户表 和 订单表

- 多对多 ： 借助中间表，中间表至少两个字段，分别关联两张表的主键

学生表 和 课程表

用户表 和 兴趣爱好表

多表操作 一对一

模型 ： 人和身份证 、 一个人只有一张身份证

数据准备

```
CREATE DATABASE db2;


CREATE TABLE person(
        id INT PRIMARY KEY AUTO_INCREMENT,
        NAME VARCHAR(20),
        age INT
);
```

```
INSERT INTO person VALUES (NULL,'张三',23);

INSERT INTO person VALUES (NULL,'李四',24);

INSERT INTO person VALUES (NULL,'王五',25);


CREATE TABLE card(

        id INT PRIMARY KEY AUTO_INCREMENT,

        number VARCHAR(30),

        pid INT,

        CONSTRAINT cp_fk FOREIGN KEY (pid) REFERENCES person(id)

)

INSERT INTO card VALUES (NULL,'12345',1);

INSERT INTO card VALUES (NULL,'23456',2);

INSERT INTO card VALUES (NULL,'34567',3);
```

bean

Student.java

```
public class Person {

    private Integer id;      //主键 id

    private String name;     //人的姓名

    private Integer age;     //人的年龄

}
```

Card.java

```
public class Card {

    private Integer id;

    private String number;
```

```
    private Person p;

}
```

OneToOneMapper.xml

```
<mapper namespace="com.mybatis.oneToOne.OneToOneMapper">

<!--
<resultMap>是 Mybatis 的结果集封装

-->

<!--通过 <resultMap>: 配置字段和对象属性的映射关系标签
    属性
        id 属性：resultMap 的唯一标识，此 id 值用于 select 元素 resultMap 属性的
引用。
        type 属性：表示该 resultMap 的映射结果类型（通常是 Java 实体类）。
  子节点
        id 子节点：一般对应数据库中该行的主键 id，设置此项可以提升 MyBatis 性能。
        配置主键映射关系标签
        result 子节点：映射到 JavaBean 的某个 “简单类型” 属性，如基础数据类
型、包装类等。
        配置非主键映射关系标签
  子节点属性
        column 属性：表示从数据库中查询的字段名或别名。表中字段名称
        property 属性：表示查询出来的字段对应的值赋给实体对象的哪个属性。实体对
象变量名称
说明：子节点 id 和 result 均可实现最基本的结果集映射，将列映射到简单数据类型的属
性。
这两者唯一不同的是：在比较对象实例时 id 将作为结果集的标识属性。
这有助于提高总体性能，特别是应用缓存和嵌套结果映射的时候。
而若要实现高级结果映射，就需要学习下面两个配置项: association 和 collection。
    association：映射到 JavaBean 的某个 “复杂类型” 属性，比如 JavaBean 类，
    即 JavaBean 内部嵌套一个复杂数据类型(JavaBean)属性，这种情况就属于复杂类型的
关联。
    但是需要注意: association 仅处理一对一的关联关系。
     association：配置被包含对象的映射关系
        property：被包含对象的变量名
        javaType：被包含对象的数据类型

-->
```

```
    <resultMap id="oneToOne" type="card">
        <id column="cid" property="id"/>
        <result column="number" property="number"/>

        <association property="p" javaType="person">
            <id column="pid" property="id"/>
            <result column="name" property="name"/>
            <result column="age" property="age"/>
        </association>
    </resultMap>

<select id="selectAll" resultMap="oneToOne">
        SELECT
            c.id cid,
            number,
            pid,
            NAME,
            age
        FROM
            card c , person p
        WHERE
            c.pid = p.id
</select>
```

```
</mapper>
```

MyBatisConfig.xml

```
    <mappers>
        <!-- mapper 引入指定的映射配置文件  resource 属性指定映射配置文件的名称
-->
       <mapper resource="com/mybatis/oneToOne/OneToOneMapper.xml"></mapper>
    </mappers>
```

测试

```
  //4.获取 OneToOneMapper 接口的实现类对象
        OneToOneMapper mapper = sqlSession.getMapper(OneToOneMapper.class);


        //5.调用实现类的方法，接收结果
        List<Card> list = mapper.selectAll();


        //6.处理结果
        for (Card c : list) {
            System.out.println(c);
        }
```

多表操作 一对多

数据准备

多表操作 一对多 班级对学生

```
CREATE TABLE classes(
        id INT PRIMARY KEY AUTO_INCREMENT,
        NAME VARBINARY(20)
```

```
);
INSERT INTO classes VALUES (NULL,'s一班');
INSERT INTO classes VALUES (NULL,'s二班');


CREATE TABLE student(
        id INT PRIMARY KEY AUTO_INCREMENT,
        NAME VARCHAR(30),
        age INT,
        cid INT,
        CONSTRAINT cs_fk FOREIGN KEY (cid) REFERENCES classes(id)
);
INSERT INTO student VALUES (NULL,'张三',23,1);
INSERT INTO student VALUES (NULL,'李四',24,1);
INSERT INTO student VALUES (NULL,'王五',25,2);
INSERT INTO student VALUES (NULL,'赵六',26,2);
```

bean

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

```
    private Integer id;

    private String name;

    private List<Student> students; //班级中所有学生对象

}
```

OneToManyMapper.xml

```
<mapper namespace="com.mybatis.oneToMany.OneToManyMapper">


    <resultMap id="oneToMany" type="classes">

        <id column="cid" property="id"/>

        <result column="cname" property="name"/>

        <!--
           collection：配置被包含的集合对象映射关系
           property：被包含对象的变量名
           ofType：被包含对象的实际数据类型

        -->

       <collection property="students" ofType="student">

           <id column="sid" property="id"/>

           <result column="sname" property="name"/>

           <result column="sage" property="age"/>

       </collection>

    </resultMap>

    <select id="selectAll" resultMap="oneToMany">

            SELECT

                c.id cid,

                c.name cname,
```

```
            s.id sid,
            s.name sname,
            s.age sage
        FROM
            classes c,student s
        WHERE
            c.id=s.cid
    </select>
```

测试

```
        //4.获取 OneToManyMapper 接口的实现类对象
        OneToManyMapper mapper = sqlSession.getMapper(OneToManyMapper.class);

        //5.调用实现类的方法，接收结果
        List<Classes> classes = mapper.selectAll();

        //6.处理结果
for (Classes cls : classes) {
    System.out.println(cls.getId() + "," + cls.getName());
    List<Student> students = cls.getStudents();
    for (Student student : students) {
        System.out.println("\t" + student);
    }
}
```

多表操作 多对多

数据准备

学生和课程 一个学生可以选择多门课程 一个课程可以被多个学生选择

```
CREATE TABLE course(
        id INT PRIMARY KEY AUTO_INCREMENT,
        NAME VARCHAR(20)
);

INSERT INTO course VALUES (NULL,'语文');
INSERT INTO course VALUES (NULL,'数学');

CREATE TABLE stu_cr(
        id INT PRIMARY KEY AUTO_INCREMENT,
        sid INT,
        cid INT,
        CONSTRAINT sc_fk1 FOREIGN KEY (sid) REFERENCES student(id),
        CONSTRAINT sc_fk2 FOREIGN KEY (cid) REFERENCES course(id)
);

INSERT INTO stu_cr VALUES (NULL,1,1);
INSERT INTO stu_cr VALUES (NULL,1,2);
INSERT INTO stu_cr VALUES (NULL,2,1);
INSERT INTO stu_cr VALUES (NULL,2,2);
```

bean

Course.java

```
public class Course {

    private Integer id;

    private String name;

}
```

Student.java

```
public class Student {

    private Integer id;

    private String name;

    private Integer age;


    private List<Course> courses;  //学生所选的课程集合

}
```

ManyToManyMapper.xml

```
<?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.mybatis.manyToMany.ManyToManyMapper">

    <resultMap id="ManyToMany" type="student">

        <id column="sid" property="id"/>

        <result column="sname" property="name"/>

        <result column="sage" property="age"/>
```

```
<!--    <collection>：配置被包含集合对象的映射关系标签。
            属性：
                property 属性：被包含集合对象的变量名

                ofType 属性：集合中保存的对象数据类型-->

        <collection property="courses" ofType="course">

            <id column="cid" property="id"/>

            <result column="cname" property="name"/>

        </collection>

    </resultMap>

    <select id="selectAll" resultMap="ManyToMany">

        SELECT

            sc.sid,

            s.name sname,

            s.age sage,

            sc.cid,

            c.name cname

        FROM

            student s,course c,stu_cr sc

        WHERE

            sc.sid=s.id AND sc.cid=c.id

    </select>

</mapper>
```

处理结果

```
 //5.调用实现类的方法，接收结果
```

```
List<Student> students = mapper.selectAll();


        //6.处理结果

for (Student student : students) {

    System.out.println(student.getId() + "," + student.getName() + "," +
student.getAge());

    List<Course> courses = student.getCourses();

    for (Course cours : courses) {

        System.out.println("\t" + cours);

    }

        }
```

小结

```
<!--一对一 所有标签如下

            <resultMap>：配置字段和对象属性的映射关系标签。
               属性：
                   id 属性：唯一标识
                   type 属性：实体对象类型

            <id>：配置主键映射关系标签。
            <result>：配置非主键映射关系标签。
               属性：
                   column 属性：表中字段名称
                   property 属性：  实体对象变量名称
         (*)<association>：配置被包含对象的映射关系标签。
属性：
                   property 属性：被包含对象的变量名
                   javaType 属性：被包含对象的数据类型

-->
```

```
<!--多对多 & 一对多 所有标签如下

        <resultMap>：配置字段和对象属性的映射关系标签。
            属性：
                id 属性：唯一标识
                type 属性：实体对象类型

        <id>：配置主键映射关系标签。
        <result>：配置非主键映射关系标签。
            属性：
                column 属性：表中字段名称
                property 属性：  实体对象变量名称
      (*)<collection>：配置被包含集合对象的映射关系标签。
            属性：
                property 属性：被包含集合对象的变量名
                ofType 属性：集合中保存的对象数据类型

-->
```

扩展

针对结果集而创建对应的 javabean 类，来接收结果集数据，这种 javabean 在领域驱动模型中称之为 ： VO