第一章 MyBatis框架

-MyBatis持久层框架 2018/11/7 [泽林.王峰]

授课目标

1、 MyBatis概述

2、 开发第一个MyBatis程序-环境搭建

3、 完成MyBatis的CRUD操作

4、 优化MyBatis的配置文件

5、 解决字段名与表列名不一致方案

6、 自动增长列与模糊查询

7、 使用注解完成CRUD操作

8、 MyBatis开发Dao的两种方式

授课内容

1、 MyBatis概述

1.1）简介：

MyBatis是一个支持普通SQL查询，存储过程和高级映射的优秀**持久层框架**。MyBatis消除了几乎所有的JDBC代码和参数的手工设置以及对结果集的检索封装。MyBatis可以使用简单的**XML或注解**用于配置和原始映射，将接口和Java的POJO（Plain Old Java Objects，普通的Java对象）映射成数据库中的记录。

1.2）JDBC的不足之处：

1、数据库连接频繁的创建和关闭，缺点浪费数据库的资源，影响操作效率

设想：使用数据库连接池

2、sql语句是硬编码，如果需求变更需要修改sql，就需要修改java代码，需要重新编译，系统不易维护。

设想：将sql语句 统一配置在文件中，修改sql不需要修改java代码。

3、通过preparedStatement向占位符设置参数，存在硬编码（ 参数位置，参数）问题。系统不易维护。

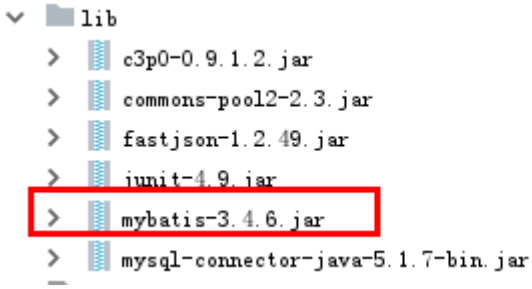
设想：将sql中的占位符及对应的参数类型配置在配置文件中，能够自动输入映射。

4、遍历查询结果集存在硬编码（列名）。

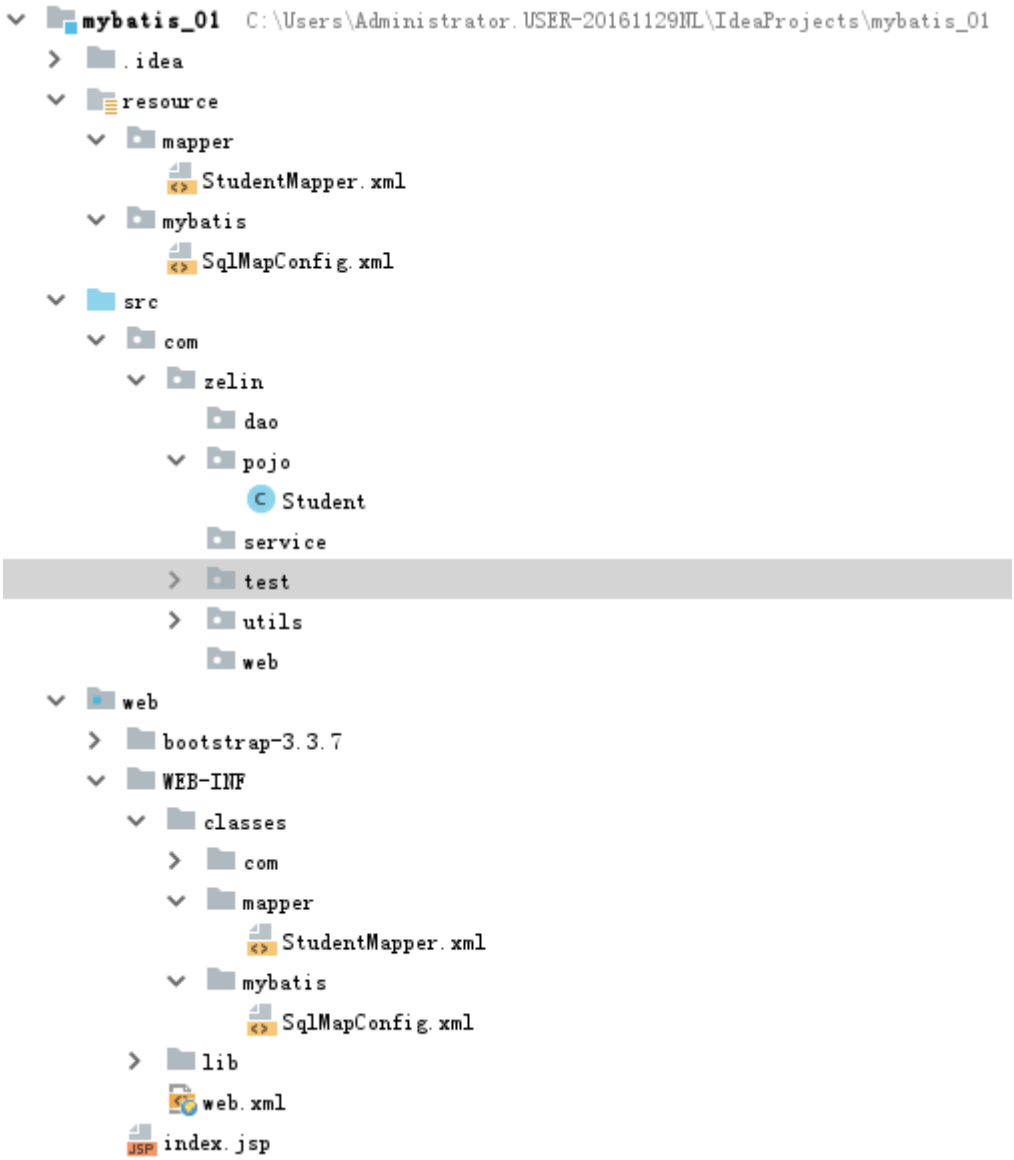
设想：自动进行sql查询结果向java对象的映射（输出映射）。

2、 开发第一个MyBatis程序-环境搭建

2.1)添加jar包：



2.2) 定义相关的配置及工程目录结构，如下图：



2.3) 在resource目录下创建mybatis目录，并创建SqlMapConfig.xml文件，内容如下：

```
1  <?xml version="1.0" encoding="UTF-8" ?>
2  <!DOCTYPE configuration
```

```
        PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
        "http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>
    <environments default="development">
        <environment id="development">
            <transactionManager type="JDBC"/>
            <dataSource type="POOLED">
                <property name="driver" value="com.mysql.jdbc.Driver"/>
                <property name="url" value="jdbc:mysql:///java1301"/>
                <property name="username" value="root"/>
                <property name="password" value="123"/>
            </dataSource>
        </environment>
    </environments>
    <mappers>
        <mapper resource="mapper/StudentMapper.xml"/>
    </mappers>
</configuration>
```

2.4) 新建com.zelin.pojo.Student这个实体类：

```
public class Student {
    private int sid;
    private String sname;
    private String sex;
    private int age;
    private String addr;
    private String birth;
    private int cid;

    public Student() {
    }

    public Student(String sname, String sex, int age, String addr, String birth, int cid) {
        this.sname = sname;
        this.sex = sex;
        this.age = age;
        this.addr = addr;
        this.birth = birth;
        this.cid = cid;
    }

    public Student(int sid, String sname, String sex, int age, String addr, String birth, int cid) {
        this.sid = sid;
        this.sname = sname;
        this.sex = sex;
        this.age = age;
        this.addr = addr;
        this.birth = birth;
        this.cid = cid;
```

```
30      }
31
32      public int getSid() {
33          return sid;
34      }
35
36      public void setSid(int sid) {
37          this.sid = sid;
38      }
39
40      public String getSname() {
41          return sname;
42      }
43
44      public void setSname(String sname) {
45          this.sname = sname;
46      }
47
48      public String getSex() {
49          return sex;
50      }
51
52      public void setSex(String sex) {
53          this.sex = sex;
54      }
55
56      public int getAge() {
57          return age;
58      }
59
60      public void setAge(int age) {
61          this.age = age;
62      }
63
64      public String getAddr() {
65          return addr;
66      }
67
68      public void setAddr(String addr) {
69          this.addr = addr;
70      }
71
72      public String getBirth() {
73          return birth;
74      }
75
76      public void setBirth(String birth) {
77          this.birth = birth;
78      }
79
80      public int getCid() {
81          return cid;
82      }
```

```
83
84      public void setCid(int cid) {
85          this.cid = cid;
86      }
87
88      @Override
89      public String toString() {
90          return "Student{" +
91                  "sid=" + sid +
92                  ", sname='" + sname + '\'' +
93                  ", sex='" + sex + '\'' +
94                  ", age=" + age +
95                  ", addr='" + addr + '\'' +
96                  ", birth='" + birth + '\'' +
97                  ", cid=" + cid +
98                  '}';
99      }
100 }
```

2.5) 在resource目录下新建mapper目录，并同时创建StudentMapper.xml文件

```
1  <?xml version="1.0" encoding="UTF-8" ?>
2  <!DOCTYPE mapper
3          PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
4          "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
5  <mapper namespace="test">
6      <!--1.查询所有的学生-->
7      <select id="findAll" resultType="com.zelin.pojo.Student">
8          select * from student
9      </select>
10 </mapper>
```

2.6) 创建MyBatisUtils工具类，用于得到SqlSession对象

```
1  /**
2   * 访问MyBatis的工具类
3   */
4  public class MyBatisUtils {
5
6      //1.根据配置文件得到SqlSessionFactory对象
7      public static SqlSessionFactory getSqlSessionFactory(String configName){
8          try {
9              //1.1) 根据配置文件得到输入流
10             InputStream inputStream = Resources.getResourceAsStream(configName);
11             //1.2) 根据输入流得到SqlSessionFactory对象
12             return new SqlSessionFactoryBuilder().build(inputStream);
13         } catch (IOException e) {
14             e.printStackTrace();
15             return null;
```

```
        }
    }
    //2.通过SqlSessionFactory得到SqlSession对象( 默认不开启事务)
    public static SqlSession getSqlSession(){
        SqlSessionFactory sqlSessionFactory =
getSqlSessionFactory("mybatis/SqlMapConfig.xml");
        return sqlSessionFactory.openSession();
    }
    //3.取得一个通用的SqlSession对象（可自己指定是否开启事务）
    //增、删除、改时需要指定参数为true,查询时不需要
    public static SqlSession getSqlSession(boolean isAutoCommit){
        SqlSessionFactory sqlSessionFactory =
getSqlSessionFactory("mybatis/SqlMapConfig.xml");
        return sqlSessionFactory.openSession(isAutoCommit);
    }
}
```

2.7) 进行单元测试

```
public class TestMyBatis {
    @Test
    public void test01(){
        //1.得到SqlSession对象
        SqlSession sqlSession = MyBatisUtils.getSqlSession();
        //2.查询得到学生列表
        List<Student> students = sqlSession.selectList("test.findAll");
        //3.打印学生信息
        for(Student student : students){
            System.out.println(student);
        }
    }
}
```

2.8) 运行效果演示：

```
Student{sid=1, sname='张三', sex='男', age=250, addr='广州', birth='2018-09-06 14:33:50.0', cid=2}
Student{sid=2, sname='小五', sex='男', age=28, addr='广州', birth='2010-10-27 00:00:00.0', cid=1}
Student{sid=3, sname='小红', sex='女', age=19, addr='杭州', birth='2008-08-21 00:00:00.0', cid=2}
Student{sid=4, sname='王二小', sex='男', age=12, addr='岳阳', birth='2010-02-16 00:00:00.0', cid=2}
Student{sid=5, sname='张小二', sex='男', age=28, addr='上海123', birth='2018-10-30 00:00:00.0', cid=3}
Student{sid=6, sname='黄家驹', sex='男', age=54, addr='香港', birth='1995-03-12 00:00:00.0', cid=3}
Student{sid=7, sname='罗成', sex='男', age=22, addr='邵阳', birth='1997-01-18 00:00:00.0', cid=1}
Student{sid=8, sname='魏征', sex='男', age=28, addr='洛ab阳', birth='2000-03-16 00:00:00.0', cid=1}
Student{sid=9, sname='徐达', sex='男', age=29, addr='江苏无锡', birth='2010-02-23 00:00:00.0', cid=2}
Student{sid=13, sname='孙宾', sex='男', age=35, addr='上海浦东', birth='1987-06-23 00:00:00.0', cid=2}
Student{sid=15, sname='赵本山', sex='男', age=65, addr='东北大街', birth='1954-11-17 00:00:00.0', cid=1}
Student{sid=16, sname='宋小宝', sex='男', age=54, addr='东北', birth='1963-07-18 00:00:00.0', cid=2}
Student{sid=19, sname='常遇春', sex='男', age=100, addr='河南', birth='1918-08-28 00:00:00.0', cid=1}
Student{sid=20, sname='常遇春', sex='男', age=100, addr='河南', birth='1918-08-28 00:00:00.0', cid=1}
Student{sid=22, sname='胡大海', sex='男', age=110, addr='山东', birth='1908-02-28 00:00:00.0', cid=2}
```

3、 完成MyBatis的CRUD操作

3.1)在StudentMapper.xml文件中定义sql语句：

```
<?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="test">
    <!--1.查询所有的学生-->
    <select id="findAll" resultType="com.zelin.pojo.Student">
        select * from student
    </select>
    <!--2.添加学生-->
    <insert id="insert" parameterType="com.zelin.pojo.Student">
        insert into student values(null,#{sname},#{sex},#{age},#{addr},#{cid},#{birth})
    </insert>
    <!--3.修改学生-->
    <update id="update" parameterType="com.zelin.pojo.Student">
        update student
        set sname=#{sname},sex=#{sex},age=#{age},addr=#{addr},
            cid=#{cid},birth=#{birth}
        where sid=#{sid}
    </update>
    <!--4.删除学生-->
    <delete id="delete" parameterType="int">
        delete from student where sid=#{value}
    </delete>
</mapper>
```

3.2)进行单元测试：

```
@Test
    //添加学生
    public void test02(){
        //1.得到SqlSession对象
        SqlSession sqlSession = MyBatisUtils.getSqlSession(true);
        //2.执行添加方法
        //2.1)构造要添加的学生对象
        Student student = new Student("余小明","男",20,"上海","2000-1-1",1);
        //2.2)执行添加学生操作
        sqlSession.insert("test.insert",student);
        System.out.println("添加学生成功！");
    }
    @Test
    //修改学生
    public void test03(){
        //1.得到SqlSession对象

        SqlSession sqlSession = MyBatisUtils.getSqlSession(true);
```

```
        //2.执行修改方法
        //2.1)构造要修改的学生对象
        Student student = new Student(36,"张小远","男",22,"广州","2001-9-10",2);
        //2.2)修改学生
        sqlSession.update("test.update",student);
        System.out.println("修改成功！");

    }
    @Test
    public void test04(){
        //1.得到SqlSession对象
        SqlSession sqlSession = MyBatisUtils.getSqlSession(true);
        //2.执行删除学生
        sqlSession.delete("test.delete",36);
        System.out.println("删除成功！");
    }
```

> **注意：**
>
> 查询时得到的getSqlSession()方法，可以不带参数，即不需要事务。但是在增、删、改操作时必须使用getSqlSession(true)方法来得到带有事务功能的SqlSession对象。

4、 优化MyBatis的配置文件

4.1) 别名的优化配置-在SqlMapConfig.xml全局配置文件中定义如下配置：

```
<configuration>
<typeAliases>
        <!--下面这种配置只能指定一个类的别名（开发中不常用）-->
        <!--<typeAlias type="com.zelin.pojo.Student" alias="student"/>-->
        <!--只指定要映射类的包名即可，规则：就是根据类名作为别名，类名首字母可以是小写，
           如：student即是com.zelin.pojo.Student的别名，或者：Student
        -->
        <package name="com.zelin.pojo"/>
</typeAliases>
...
</configuration>
```

另外，系统也为我们提供了一些类的别名，如下图所示：

别名	映射的类型
_byte	byte
_long	long
_short	short
_int	int
_integer	int
_double	double
_float	float
_boolean	boolean
string	String
byte	Byte
long	Long
short	Short
int	Integer
integer	Integer
double	Double
float	Float
boolean	Boolean
date	Date
decimal	BigDecimal
bigdecimal	BigDecimal
object	Object
map	Map
hashmap	HashMap
list	List
arraylist	ArrayList
collection	Collection
iterator	Iterator

4.2) 引入数据库访问的属性文件：

4.2.1) 在resource/db下添加db.properties文件：

```
jdbc.driver=com.mysql.jdbc.Driver
jdbc.url=jdbc:mysql:///java1301
jdbc.user=root
jdbc.password=123
```

4.2.2) 在mybatis/SqlMapConfig.xml修改配置如下：

```
<environments default="development">
       <environment id="development">
           <transactionManager type="JDBC"/>
           <dataSource type="POOLED">
               <property name="driver" value="${jdbc.driver}"/>
               <property name="url" value="${jdbc.url}"/>
               <property name="username" value="${jdbc.user}"/>
               <property name="password" value="${jdbc.password}"/>
           </dataSource>
       </environment>
   </environments>
```

5、 解决字段名与表列名不一致方案

5.1)优化方式一（使用as别名）→修改StudentMapp.xml

```
<!--1.1处理表的列名与java对象的字段名不一致的方式-->
    <select id="findAll2" resultType="student">
        select sid,sex,sname,age,addr,cid,birth as birthday from student
    </select>
```

5.2)优化方式二（使用自定义结果集映射）→修改StudentMapp.xml

```
<!--结果集映射-->
    <resultMap id="studentMap" type="student">
        <id column="sid" property="sid"/>
        <result column="sname" property="sname"/>
        <result column="sex" property="sex"/>
        <result column="age" property="age"/>
        <result column="addr" property="addr"/>
        <result column="cid" property="cid"/>
        <result column="birth" property="birthday"/>
    </resultMap>
<!--1.2使用结果集映射来处理java对象的字段名与表的列名不一致的情况-->
    <select id="findAll3" resultMap="studentMap">
        select * from student
    </select>
```

6、 自动增长列与模糊查询

6.1）自动增长列的处理：（MySql版）

```
 <insert id="insert" parameterType="student">
        <!--解决mysql插入数据后，得到的自动增长列字段值-->
        <selectKey keyColumn="sid" keyProperty="sid" order="AFTER" resultType="int">
            Select LAST_INSERT_ID()
        </selectKey>
        insert into student values(null,#{sname},#{sex},#{age},#{addr},#{cid},#{birth})
    </insert>
```

> **说明：**
>
> 1. 当添加完此学生后，就会将当前新添加记录的sid（主键值）传入参数对象student的sid中。这个在实际开发中使用非常多，
> 2. 比如：订单与订单项，当我们添加完一条订单后，又要在订单项中添加多条记录，而每一条订单记录就需要订单表的主键（订单id），此时，就需要在添加完订单后，从订单对象中获取到订单id，从而将其添加到订单项中。

6.2) 模糊查询

6.2.1) 方案一：(使用#{}加上%)

```
<!--5.条件查询(方式一)-->
    <select id="findStudentsByWords" resultType="student" parameterType="string">
        select * from student where addr like #{addr}
    </select>
```

查询时(单元测试)：

```
@Test
    public void test05(){
        //1.得到SqlSession对象
        SqlSession sqlSession = MyBatisUtils.getSqlSession(true);
        //2.进行条件查询
        String keywords = "州";
        List<Student> students = sqlSession.selectList("test.findStudentsByWords","%"+keywords+"%");
        //3.遍历集合
        for(Student student :students){
            System.out.println(student);
        }
    }
```

6.2.1) 方案二：（使用${}）

```
<!--5.1条件查询（方式二）-->
    <select id="findStudentsByWords2" resultType="student" parameterType="string">
        select * from student where addr like '%${value}%'
    </select>
```

查询时(单元测试)：

```
//条件查询方式二
    @Test
    public void test06(){
        //1.得到SqlSession对象
        SqlSession sqlSession = MyBatisUtils.getSqlSession(true);
        //2.进行条件查询
        String keywords = "州";
        List<Student> students =
 sqlSession.selectList("test.findStudentsByWords2",keywords);
        //3.遍历集合
        for(Student student :students){
            System.out.println(student);
        }
    }
```

> ### #{}和${}**小结：**
>
> （1）#{}表示一个占位符号，通过#{}可以实现preparedStatement向占位符中设置值，自动进行java类型和jdbc类型转换。#{}可以有效防止sql注入。 #{}可以接收简单类型值或pojo属性值。 如果parameterType传输单个简单类型值，#{}括号中可以是value或其它名称。
>
> （2）表示拼接sql串，通过{}可以将parameterType 传入的内容拼接在sql中且不进行jdbc类型转换，可以接收简单类型值或$pojo$属性值，如果$parameterType$传输单个简单类型值，{}括号中名称只能是value。

7、 使用注解完成CRUD操作

7.1)定义Classes.java实体类

```
public class Classes {
    private int cid;
    private String cname;

    public Classes() {
    }

    public Classes(int cid, String cname) {
        this.cid = cid;
        this.cname = cname;
    }

    public int getCid() {
        return cid;
    }

    public void setCid(int cid) {
        this.cid = cid;
    }

    public String getCname() {
```

```
        return cname;
    }

    public void setCname(String cname) {
        this.cname = cname;
    }

    @Override
    public String toString() {
        return "Classes{" +
                "cid=" + cid +
                ", cname='" + cname + '\'' +
                '}';
    }
}

```

7.2)定义com.zelin.dao.ClassesMapper接口

```
public interface ClassesMapper {
    //查询所有班级
    @Select("select * from classes")
    public List<Classes> findAll() throws Exception;
    //添加班级
    @Insert("insert into classes values(null,#{cname})")
    public void insert(Classes classes) throws  Exception;
    //修改班级
    @Update("update classes set cname=#{cname} where cid=#{cid}")
    public void update(Classes classes) throws Exception;
    //删除班级
    @Delete("delete from classes where cid=${value}")
    public void delete(String cid) throws Exception;
}
```

7.3)测试

```
**
 * 测试注解版的MyBatis
 */
public class TestMyBatisAnno {
    @Test
    public void test01() throws Exception {
        //1.得到SqlSession对象
        SqlSession sqlSession = MyBatisUtils.getSqlSession(true);
        //2.得到mapper对象
        ClassesMapper classesMapper = sqlSession.getMapper(ClassesMapper.class);
        //3.调用接口中的方法
        List<Classes> classes = classesMapper.findAll();
        //4.打印
        for(Classes cls : classes){
            System.out.println(cls);
```

```
        }
    }
    //测试添加
    @Test
    public void test02() throws  Exception{
        //1.得到SqlSession对象
        SqlSession sqlSession = MyBatisUtils.getSqlSession(true);
        //2.添加班级
        //2.1)构造班级对象
        Classes classes = new Classes();
        classes.setCname("1304班");
        //2.2)添加班级
        ClassesMapper mapper = sqlSession.getMapper(ClassesMapper.class);
        mapper.insert(classes);
        System.out.println("班级添加成功!");
    }
    //测试修改
    @Test
    public void test03() throws  Exception{
        //1.得到SqlSession对象
        SqlSession sqlSession = MyBatisUtils.getSqlSession(true);
        //2.修改班级
        //2.1)构造班级对象
        Classes classes = new Classes(4,"1304");
        //2.2)修改班级
        ClassesMapper mapper = sqlSession.getMapper(ClassesMapper.class);
        mapper.update(classes);
        System.out.println("班级修改成功!");
    }
    //测试删除
    @Test
    public void test04() throws  Exception{
        //1.得到SqlSession对象
        SqlSession sqlSession = MyBatisUtils.getSqlSession(true);
        //2.删除班级
        //2.2)删除班级
        ClassesMapper mapper = sqlSession.getMapper(ClassesMapper.class);
        mapper.delete("5");
        System.out.println("班级删除成功!");
    }
}
```

8、 MyBatis开发Dao的两种方式

8.1)第一种整合方式：

8.1) 原始DAO开发方式：(较麻烦)

8.1.1) 定义StudentDao.java接口：

```
public interface StudentDao {
```

```
    public List<Student> findAll(String id);

    public void insert(String id,Student student);

    public void update(String id,Student student);

    public void delete(String id,int sid);

    public Student findStudentBySid(String id,int sid);

}
```

8.1.2) 定义StudentDaoImpl.java实现类：

```
public class StudentDaoImpl implements StudentDao{

    //查询所有学生

    @Override

    public List<Student> findAll(String id) {

        SqlSession sqlSession = null;

        try {

            sqlSession = MyBatisUtils.getSqlSession();

            sqlSession.selectList(id);

            return sqlSession.selectList(id);



        } catch (Exception e) {

            e.printStackTrace();

        }finally {

            sqlSession.close();

        }

        return null;

    }

```

```
35
36
37      //添加学生
38
39      @Override
40
41      public void insert(String id, Student student) {
42
43          SqlSession sqlSession = null;
44
45          try {
46
47              sqlSession = MyBatisUtils.getSqlSession(true);
48
49              sqlSession.insert(id, student);
50
51
52
53          } catch (Exception e) {
54
55              e.printStackTrace();
56
57          }finally {
58
59              sqlSession.close();
60
61          }
62
63      }
64
65      //修改学生
66
67      @Override
68
69      public void update(String id, Student student) {
70
71          SqlSession sqlSession = null;
72
73          try {
74
75              sqlSession = MyBatisUtils.getSqlSession(true);
76
77              sqlSession.update(id, student);
78
79
80
81          } catch (Exception e) {
82
83              e.printStackTrace();
84
85          }finally {
86

87              sqlSession.close();
```

```
88
89          }
90
91      }
92
93      //删除学生
94
95      @Override
96
97      public void delete(String id, int sid) {
98
99          SqlSession sqlSession = null;
100
101         try {
102
103             sqlSession = MyBatisUtils.getSqlSession(true);
104
105             sqlSession.delete(id, sid);
106
107
108
109         } catch (Exception e) {
110
111             e.printStackTrace();
112
113         }finally {
114
115             sqlSession.close();
116
117         }
118
119     }
120
121     //查询单个学生
122
123     @Override
124
125     public Student findStudentBySid(String id, int sid) {
126
127         SqlSession sqlSession = null;
128
129         try {
130
131             sqlSession = MyBatisUtils.getSqlSession();
132
133             return sqlSession.selectOne(id,sid);
134
135
136
137         } catch (Exception e) {
138
139             e.printStackTrace();

140
```

```
        }finally {

            sqlSession.close();

        }

        return null;

    }

}

```

8.1.3) 测试代码：

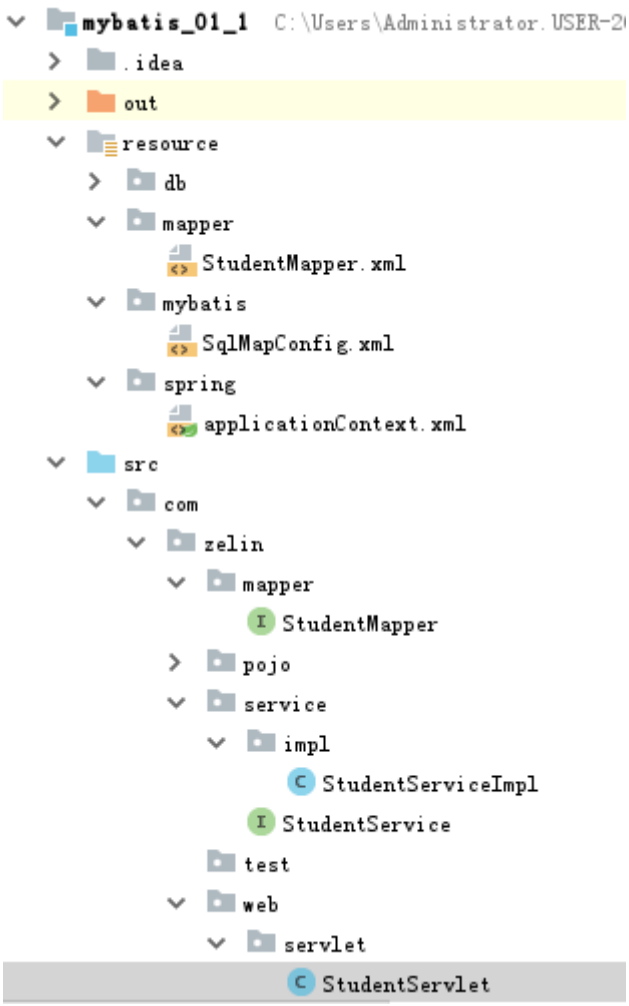
```
public class TestStudentDao {

    private StudentDao studentDao;

    @Before

    public void init(){

        studentDao = new StudentDaoImpl();

    }

    //查询所有学生

    @Test

    public void testFindAll(){

        List<Student> students = studentDao.findAll("student.findAll");

        for(Student student : students){

            System.out.println(student);

        }

    }

    //查询单个学生

    @Test

    public void testFindStudent(){

        Student student =
studentDao.findStudentBySid("student.findStudentBySid",1031);

```

```
        System.out.println(student);

    }

    //添加学生

    @Test

    public void testInsertStudent(){

        Student student = new Student("李世民", "男", 100, "北京");

        student.setCid(3);

        studentDao.insert("student.insertStudent", student );

        System.out.println("添加成功！");

    }

    //添加学生

    @Test

    public void testUpdateStudent(){

        Student student = new Student(1033,"世民", "男", 200, "广州");

        student.setCid(2);

        studentDao.update("student.updateStudent", student);

        System.out.println("修改成功！");

    }

    //添加学生

    @Test

    public void testDeleteStudent(){

        studentDao.delete("student.deleteStudent", 1033);

        System.out.println("删除成功!");

    }

}
```

8.1.4) 运行效果同上.

8.2）Mapper动态代理开发方式：（经常使用）**

8.2.1)完整的目录结构如下：



8.2.2)定义resource/mapper/studentMapper.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">
<!--这里的namespace一定是接口的完整限定名-->
<mapper namespace="com.zelin.mapper.StudentMapper">
    <!--1.查询所有的学生-->
    <!--
        mapper映射文件+接口这种开发模式要求：
        ① 接口的方法名与映射文件中的id名称一致。
        ② 接口中的返回值类型（如果是数组或集合则是元素的类型）与映射文件中的resultType的类型一致
        ③ 接口中的参数类型与映射文件中的parameterType类型一致。
        ④ 这里的namespace一定是接口的完整限定名
    -->
    <select id="findAll" resultType="student" parameterType="student">
        select * from student
    </select>

```

```
</mapper>
```

8.2.3)定义resource/spring/applicationContext.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd">
    <!--1.读取属性文件-->
    <context:property-placeholder location="classpath*:db/db.properties"/>
    <!--2.读取spring注解-->
    <context:component-scan base-package="com.zelin"/>
    <!--3.配置数据源-->
    <bean id="dataSource" class="com.alibaba.druid.pool.DruidDataSource">
        <property name="driverClassName" value="${jdbc.driver}"/>
        <property name="url" value="${jdbc.url}"/>
        <property name="username" value="${jdbc.user}"/>
        <property name="password" value="${jdbc.password}"/>
    </bean>
    <!--4.配置sqlSessionFactory-->
    <bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
        <property name="dataSource" ref="dataSource"/>
        <!--4.1) 配置mapper映射文件的位置-->
        <property name="mapperLocations" value="classpath*:mapper/*.xml"/>
        <!--4.2) 配置别名包-->
        <property name="typeAliasesPackage" value="com.zelin.pojo"/>
        <!--4.3) 指定mybatis全局配置文件的位置及文件名-->
        <!--<property name="configLocation"
value="classpath:mybatis/SqlMapConfig.xml"/>-->
    </bean>

    <!--5.配置mapperScannerConfigurer-->
    <bean id="mapperScannerConfigurer"
class="org.mybatis.spring.mapper.MapperScannerConfigurer">
        <!--配置接口的扫描包-->
        <property name="basePackage" value="com.zelin.mapper"/>
        <!--注入sqlSessionFactory-->
        <property name="sqlSessionFactoryBeanName" value="sqlSessionFactory"/>
    </bean>
</beans>
```

8.2.4)定义resource/mybatis/SqlMapConfig.xml(可选)

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE configuration
        PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
        "http://mybatis.org/dtd/mybatis-3-config.dtd">
```

```
<configuration>
    <!--引入数据库访问的属性文件-->
    <!--<properties resource="db/db.properties"/>-->
    <!--配置别名-->
    <!--<typeAliases>-->
        <!--下面这种配置只能指定一个类的别名（开发中不常用）-->
        <!--<typeAlias type="com.zelin.pojo.Student" alias="student"/>-->
        <!--只指定要映射类的包名即可，规则：就是根据类名作为别名，类名首字母可以是小写，
           如：student即是com.zelin.pojo.Student的别名，或者：Student
        -->
        <!--<package name="com.zelin.pojo"/>-->
    <!--</typeAliases>-->

    <!--<environments default="development">-->
        <!--<environment id="development">-->
            <!--<transactionManager type="JDBC"/>-->
            <!--<dataSource type="POOLED">-->
                <!--<property name="driver" value="${jdbc.driver}"/>-->
                <!--<property name="url" value="${jdbc.url}"/>-->
                <!--<property name="username" value="${jdbc.user}"/>-->
                <!--<property name="password" value="${jdbc.password}"/>-->
            <!--</dataSource>-->
        <!--</environment>-->
    <!--</environments>-->
    <!--<mappers>-->
        <!--<mapper resource="mapper/StudentMapper.xml"/>-->
        <!--<mapper class="com.zelin.dao.ClassesMapper"/>-->
    <!--</mappers>-->
</configuration>
```

8.2.5)定义com.zelin.mapper.StudentMapper接口

```
public interface StudentMapper {
    public List<Student> findAll() throws Exception;
}
```

8.2.6)定义com.zelin.service.StudentService接口：

```
public interface StudentService {
    public List<Student> findAll() throws Exception;
}

```

8.2.7)定义com.zelin.service.impl.StudentServiceImpl实现

```java
@Service("studentService")
public class StudentServiceImpl implements StudentService {
    @Autowired
    private StudentMapper studentMapper;
    @Override
    public List<Student> findAll() throws Exception {
        return studentMapper.findAll();
    }
}

```

8.2.8)定义com.zelin.web.servlet.StudentServlet类

```java
@WebServlet("/student")
public class StudentServlet extends HttpServlet {
    private StudentService studentService;
    @Override
    public void init() throws ServletException {
        WebApplicationContext webApplicationContext = WebApplicationContextUtils.getWebApplicationContext(getServletContext());
        studentService = webApplicationContext.getBean(StudentService.class);
    }

    protected void service(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        response.setContentType("text/html;charset=utf-8");
        String method = request.getParameter("method");
        if(!StringUtils.isEmpty(method)){
            if("list".equals(method)){
                list(request,response);
            }
        }
    }

    private void list(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        try {
            List<Student> students = studentService.findAll();
            System.out.println(students);
            response.getWriter().print(JSON.toJSONString(students));
            response.getWriter().flush();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

}
```

8.2.9)修改web.xml文件

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
         version="3.1">
    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>classpath*:spring/applicationContext.xml</param-value>
    </context-param>
    <listener>
        <listener-
class>org.springframework.web.context.ContextLoaderListener</listener-class>
    </listener>
</web-app>
```

8.2.10)在/web目录下新建index.html文件

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
    <link rel="stylesheet" href="bootstrap-3.3.7/css/bootstrap.min.css">
    <script src="bootstrap-3.3.7/js/jquery.min.js"></script>
    <script src="bootstrap-3.3.7/js/bootstrap.min.js"></script>
    <style>
        .table {
            text-align: center;
        }

        .container {
            margin-top: 20px;
        }
    </style>
</head>
<body>
<div class="container">
    <div class="panel panel-primary">
        <div class="panel-heading ">
            <h4 class="glyphicon glyphicon-user">学生管理系统</h4>
        </div>
        <table class="table table-bordered table-hover">
            <tr class="bg-info">
                <td>学号</td>
                <td>姓名</td>
                <td>性别</td>
                <td>年龄</td>
                <td>住址</td>
                <td>生日</td>
                <td>所在班级</td>
                <td>操作</td>
```

```
            </tr>
            <tbody id="tb"></tbody>

        </table>
        <div class="panel-footer clearfix" style="text-align:right;padding:10px;">
            <a href="javascript:add()" class="btn btn-sm btn-warning pull-left glyphicon glyphicon-plus-sign"
               id="btn_add">添加学生</a>
            泽林信息版权所有 2000-2018.
        </div>
    </div>

</div>

</body>
</html>
<script>
    $(function () {
        //刷新列表
        reloadList();
    })

    //刷新列表
    function reloadList() {
        //1、发出异步请求加载所有的学生列表
        $.post(
            'student?method=list',
            function (data) {
                var info = "";
                $.each(data, function (i, v) {
                    info += "<tr>";
                    info += "<td>" + v.sid + "</td>";
                    info += "<td>" + v.sname + "</td>";
                    info += "<td>" + v.sex + "</td>";
                    info += "<td>" + v.age + "</td>";
                    info += "<td>" + v.addr + "</td>";
                    info += "<td>" + v.birth + "</td>";
                    info += "<td>" + v.cname + "</td>";
                    info += "<td>";
                    info += "<a href='javascript:;' onclick='updateStudent("+v.sid+")' class='btn btn-sm btn-info glyphicon glyphicon-pencil'>修改</a> "
                    info += "<a href='javascript:;' class='btn btn-sm btn-danger glyphicon glyphicon-trash' onclick='deleStudent("+v.sid+")'>删除</a>"
                    info += "</td>";
                    info += "</tr>";
                })
                //放上面的字符串放到tbody中
                $("#tb").html(info);
            }, "json"
        );
        //2.发出异步请求，得到所有的班级列表

        $.post(
```

```
            'classes?method=list',
            function (data) {
                var classesInfo = "";
                $(data).each(function (i, v) {
                    classesInfo += "<option value='" + v.cid + "'>" + v.cname + "
</option>";
                })
                $("#cid").html(classesInfo);
            }, "json")
    }

</script>
```

8.2.10)运行效果如下：

学生管理系统

学号	姓名	性别	年龄	住址	生日	所在班级	操作
1	张三	男	250	广州	2018-09-06 14:33:50.0	undefined	修改 删除
2	小五	男	28	广州	2010-10-27 00:00:00.0	undefined	修改 删除
3	小红	女	19	杭州	2008-08-21 00:00:00.0	undefined	修改 删除
4	王二小	男	12	岳阳	2010-02-16 00:00:00.0	undefined	修改 删除
5	张小二	男	28	上海123	2018-10-30 00:00:00.0	undefined	修改 删除
6	黄家驹	男	54	香港	1995-03-12 00:00:00.0	undefined	修改 删除
7	罗成	男	22	邵阳	1997-01-18 00:00:00.0	undefined	修改 删除
8	魏征	男	28	洛ab阳	2000-03-16 00:00:00.0	undefined	修改 删除
9	徐达	男	29	江苏无锡	2010-02-23 00:00:00.0	undefined	修改 删除
13	孙宾	男	35	上海浦东	1987-06-23 00:00:00.0	undefined	修改 删除
15	赵本山	男	65	东北大街	1954-11-17 00:00:00.0	undefined	修改 删除