一、MyBatis的核心API

- Resources Resources类，顾名思义就是资源，用于读取资源文件
- SqlSessionFactory
- SqlSession

二、使用mybatis完成crud

- ## 删除

  StudentMapper.xml中

```
<!--sql语句-->
    <!--方法  参数类型：整型-->
    <delete id="deleteStuById" parameterType="_int">
        delete from stu where id =${_parameter}
    </delete>
```

  IStudentDaoImpl中

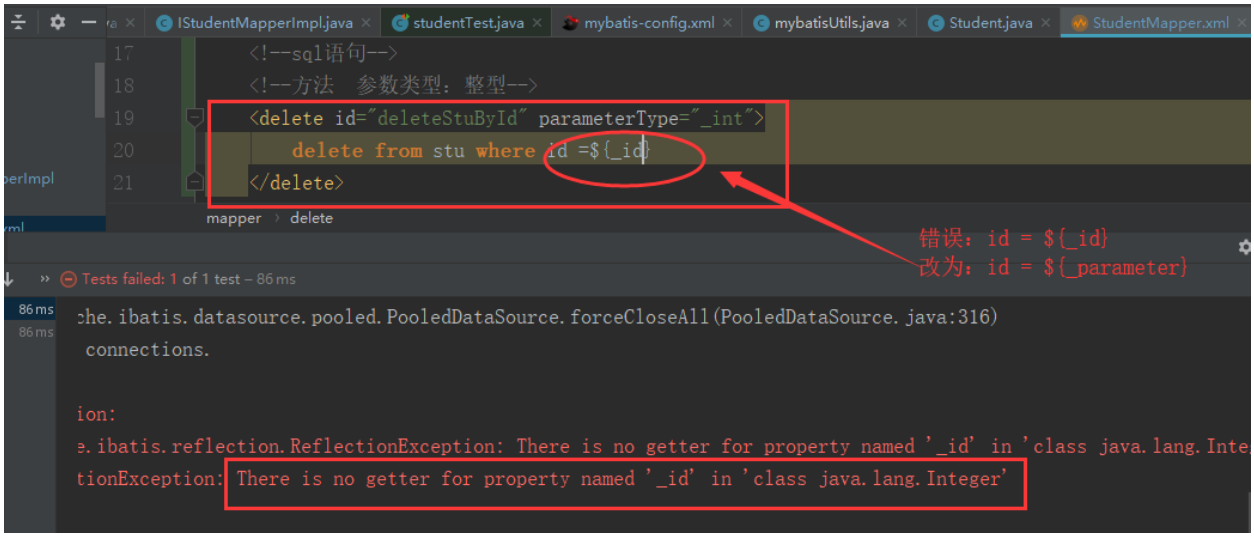
```
 @Override
    public void deleteStuById(int id) {
        //执行sql
        sqlSession.delete("deleteStuById",id);
        //提交
        sqlSession.commit();
        //关闭sqlsession
        mybatisUtils.realeaseSource(sqlSession);

    }
```

  studentTest中

```
 @Test
    public void deleteStuById(){
        studentMapper.deleteStuById(20);
    }
```

  遇到的问题

```
org.apache.ibatis.reflection.ReflectionException: There is no getter for property
named '_id' in 'class java.lang.Integer'
```



或者改为：

- ## 修改/更新

  StudentMapper.xml中

```
<!--sql语句-->
    <!--方法  参数类型：整型-->
    <update id="updateStudentById" parameterType="Student">
        update stu set sname = #{sname} where id = #{id}
    </update>
```

- ## 查询

  - ### 查询所有，返回一个Map集

    IStudentDaoImpl中

```
//查询所有，返回一个Map集
    @Override
    public Map<String, Student> findStuMap() {
        //执行sql
        Map<String,Student> studentMap=
sqlSession.selectMap("findAllStudent","sname");//第一个参数：执行的Sql语句，第二额参
数：Map中的key值
        //返回一个结果集，无需提交
        //关闭sqlsession
        mybatisUtils.realeaseSource(sqlSession);
        return studentMap;
    }
```

studentTest中

```
  //查询所有，返回一个Map集
    @Test
    public void findAllStuMap(){
      Map<String,Student> studentMap= studentMapper.findStuMap();
      Set<String> stringSet = studentMap.keySet();
      for (String setstr : stringSet){
          Student student3 =studentMap.get(setstr);
          System.out.println(student3);
      }
    }
```

- **查询一条数据**

StudentMapper.xml中

```
<!--sql语句-->
    <!--方法  参数类型：整型-->
    <select id="findOneById" resultType="Student">
        select * from stu where id = #{id}
    </select>
```

IStudentDaoImpl中

```
//查询一条数据
    @Override
    public Student findOneStuById(int id) {
        //执行sql
        Student student= sqlSession.selectOne("findOneById",id);
        //关闭sqlsession
        mybatisUtils.realeaseSource(sqlSession);
        return student;
    }
```

- 模糊查询

StudnetDao中

```
//模糊查询
    @Test
    public void findStuByName(){
        List<Student> studentList = studentMapper.findStuByName("三");
        for (Student stu:studentList){
            System.out.println(stu);
        }
    }
```

StudentMapper.xml中

```
<!--方法 参数类型：string  结果集类型：-->
    <select id="findOneById" parameterType="String" resultType="Student">
        select * from stu where id = #{id}
    </select>
```

IStudentDaoImpl中

```
 //根据姓名，模糊查询
    @Override
    public List<Student> findStuByName(String sname) {
        //执行sql
        List<Student> studentList=
sqlSession.selectList("findStudentByName",sname);
        return studentList;
    }
```

studentTest

```
    //查询 姓名，进行模糊查询
    public List<Student> findStuByName(String sname);
```

- 解决字段名和实体的属性名不一致的情况
  - 1 在查询的时候 为字段使用别名

```
<select id="findStuBySid" parameterType="int"
resultType="org.lanqiao.pojo.Student">
   select sid id,sname name,age,gender,province,tuition from stu where
sid=#{aaa}
    </select>
```

  - 使用resultMap

```
<resultMap id="stu" type="org.lanqiao.pojo.Student">
        <!--主键列-->
        <id column="sid" property="id"></id>
       <!--其余非主键列 使用result进行映射 column 是数据库的字段名  propertiy 实体类的属性名 jdbcType对应字段的数据库中的类型  javaType 实体类中的属性的类型 -->
        <result column="sname" jdbcType="VARCHAR" property="name" javaType="string"></result>
        <result column="age" property="age"></result>
        <result column="gender" property="gender"></result>
        <result column="province" property="province"></result>
        <result column="tuition" property="tuition"></result>
    </resultMap>
```

```
<select id="findStuBySid" parameterType="int"  resultMap="stu">
        select sid ,sname ,age,gender,province,tuition from stu where sid=#{aaa}
    </select>
```

    - dd
- 当当

三、动态代理

通过之前的操作，我们发现dao的实现类其实并没有做什么实质性的工作，仅仅是通过sqlSession的相关API定位到StudentMapper映射文件中的ID中的sql语句，其实真正操作DB的是mapper中的sql。所以mybatis就抛开了dao层的实现类，可以直接定位到mapper中的sql！然后执行sql对DB进行操作！这种对dao的实现方式我们称为Mapper的动态代理方式！

1、 Mapper接口开发需要遵循以下规范：

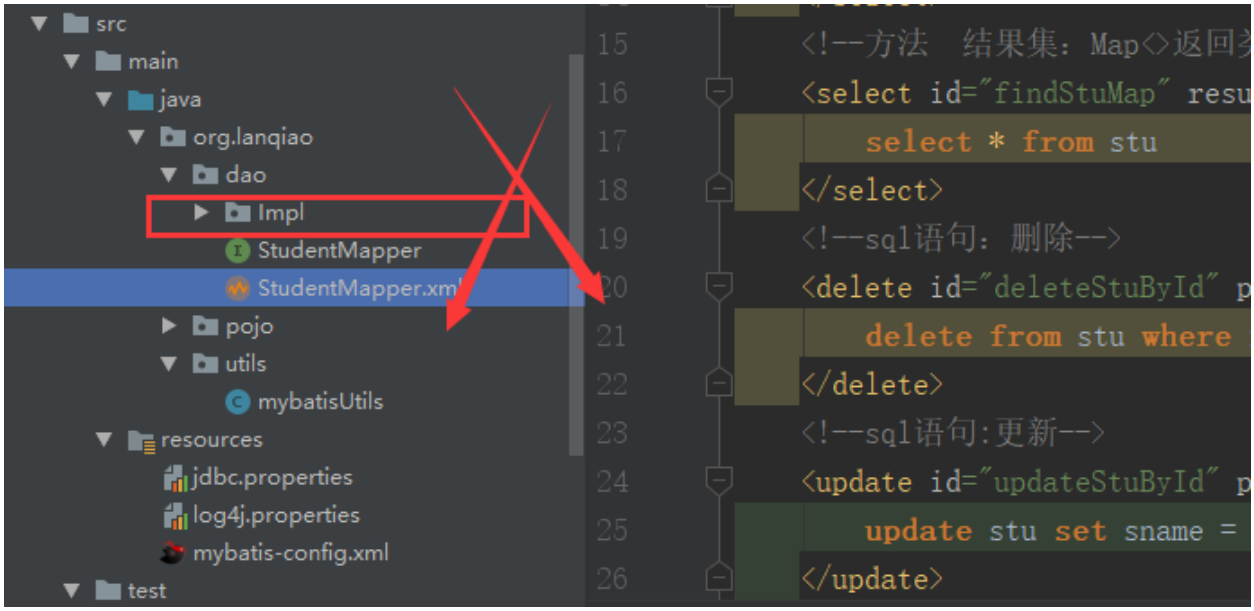
1、 Mapper.xml文件中的namespace与mapper接口的类路径相同。

2、 Mapper接口方法名和Mapper.xml中定义的每个statement的id相同

3、 Mapper接口方法的输入参数类型和mapper.xml中定义的每个sql 的parameterType的类型相同 4、 Mapper接口方法的输出参数类型和mapper.xml中定义的每个sql的resultType的类型相同

2、 实现步骤

- 删除之前：StudnetDao的实现类



- 修改StudnetMapper.xml文件中的namespace必须是StudentDao的完整限定名
- 修改StudentMapper.xml文件中所有的id必须和StudentDao接口中的方法名称完全一致
- 修改测试代码

```
public class studentTest {
    /*
      StudentMapper studentMapper = new IStudentMapperImpl();
    * 删除了：dao层的实现类：IStudentDaoImpl
    * 但在Test方法中需要 studentMapper这个对象
    * 如何得到studentMapper这个对象
    * */
//    StudentMapper studentMapper = new IStudentMapperImpl();
    StudentMapper studentMapper;
    SqlSession sqlSession;
    @Before
    public void init(){
       //获取sqlSession对象
        sqlSession = mybatisUtils.getSqlSeesion("mybatis-config.xml");
       //为dao创建对象(通过动态代理生成实体类)
        studentMapper = sqlSession.getMapper(StudentMapper.class);
    }

    ...
    ...
}
```

- 在查询的时候 为字段使用别名

```
 <select id="findOneStuById" parameterType="int"
resultType="org.lanqiao.pojo.Student">
        select id id,sname name,sage age,ssex sex from stu where sid=#{aaa}
    </select>
```

- 使用resultMap

```
 <resultMap id="stu" type="org.lanqiao.pojo.Student">
        <!--主键列-->
        <id column="sid" property="id"></id>
      <!--其余非主键列 使用result进行映射 column 是数据库的字段名  propertiy 实体类的属性名
jdbcType对应字段的数据库中的类型  javaType 实体类中的属性的类型 -->
        <result column="sname" jdbcType="VARCHAR" property="name"
javaType="string"></result>
        <result column="age" property="age"></result>
        <result column="gender" property="gender"></result>
        <result column="province" property="province"></result>
        <result column="tuition" property="tuition"></result>
    </resultMap>
```

- **多条件查询**

  将查询的参数封装为一个Map 封装成map的时候 map的key 必须和sql语句中的#{名称保持一致}

  - StudentMapper

    ```
    //多条件查询
        public List<Student> fingStuByNameAndAge(Map<String,Object> parameter);
    ```

  - StudentMapper.xml

    ```
    <!--: 多条件查询-->
        <select id="fingStuByNameAndAge" resultType="Student">
            select * from stu where sage > #{sage} and sname like '%' #{sname} '%'
        </select>
    ```

  - Test

    ```
     //多条件查询
        @Test
        public void fingStuByNameAndAgeTest(){
            Map<String,Object> parameter  = new HashMap<>();
            parameter.put("sage",12);
            parameter.put("sname","李");
            List<Student> studentList =
    studentMapper.fingStuByNameAndAge(parameter);
            for (Student stu:studentList){
                System.out.println(stu);
            }
        }
    ```

  - 当当

- **将查询参数封装成一个对象**

  - StudentMapper

    ```
    //多条件查询: 根据对象
        public List<Student> fingStuByStudent(Student student);
    ```

- StudentMapper.xml

```
<!--多条件查询：根据一个对象-->
    <select id="fingStuByStudent" resultType="Student">
        select * from stu where sage > #{sage} and sname like '%' #{sname} '%'
    </select>
```

- Test

```
 //多条件查询：根据一个对象
    @Test
    public  void fingStuByStudentTest(){
        Student student = new Student();
        student.setSage(10);
        student.setSname("李");
        List<Student> studentList = studentMapper.fingStuByStudent(student);
        for (Student stu:studentList){
            System.out.println(stu);
        }
    }
```

- dd

- **使用占位符**
  - StudentMapper

```
//多条件查询:根据年龄和姓名,并将结果封装为一个对象
    public List<Student> findStuByNameAndAge(String sname,int sage);
```

  - StudentMapper.xml

```
<!--多条件查询：根据年龄和姓名，并将结果封装为一个对象-->
    <select id="findStuByNameAndAge" resultType="Student">
        select * from stu where sage > #{arg1} and sname like '%' #{arg0} "%"
    </select>
```

  - Test

```
//多条件查询：根据年龄和姓名，并将结果封装为一个对象
    @Test
    public  void fingStuByNameAndAge(){
        List<Student> studentList = studentMapper.findStuByNameAndAge("李",10);
        for (Student stu:studentList){
            System.out.println(stu);
        }
    }
```

  - dd

- DD

- dd
- dd
- 当当
- 当当
- 当当
- 当当