一、什么是框架?

1.1 框架的定义

它是我们软件开发中的一套解决方案，不同的框架解决不同的问题。

1.2 框架的好处

框架封装了很多细节，是开发者可以使用极简的方式实现功能，大大提高开发效率

二、持久层技术解决方案

2.1 JDBC技术

涉及到Connection,PrepareStatement,ReaultSet对象

2.2 Spring的JdbcTemplate

是对jdbc的简单封装

2.3 Apache的DbUtils

和Spring的JdbcTemplate很像，也是对jdbc的简单封装

以上这些都不是框架，而是JDBC规范，JdbcTemplate和DBUtils都属于工具类

三、jdbc有什么问题

jdbc中步骤繁琐，程序员不需要关注太多的操作，包括数据库连接的关闭，id的增长，数据的封装等等，开发者只需关注sql语句即可，Mybatis框架是一个优秀的Java持久层框架，内部封装了jdbc，使得开发者只需要关注sql语句本身，而不需要花费精力去处理加载驱动，创建连接，创建Statement。

四、Mybatis简介

Mybatis框架是一个优秀的Java持久层框架，内部封装了jdbc，使得开发者只需要关注sql语句本身，而不需要花费精力去处理加载驱动，创建连接，创建Statement。使用了orm思想，实现了结果集的封装。

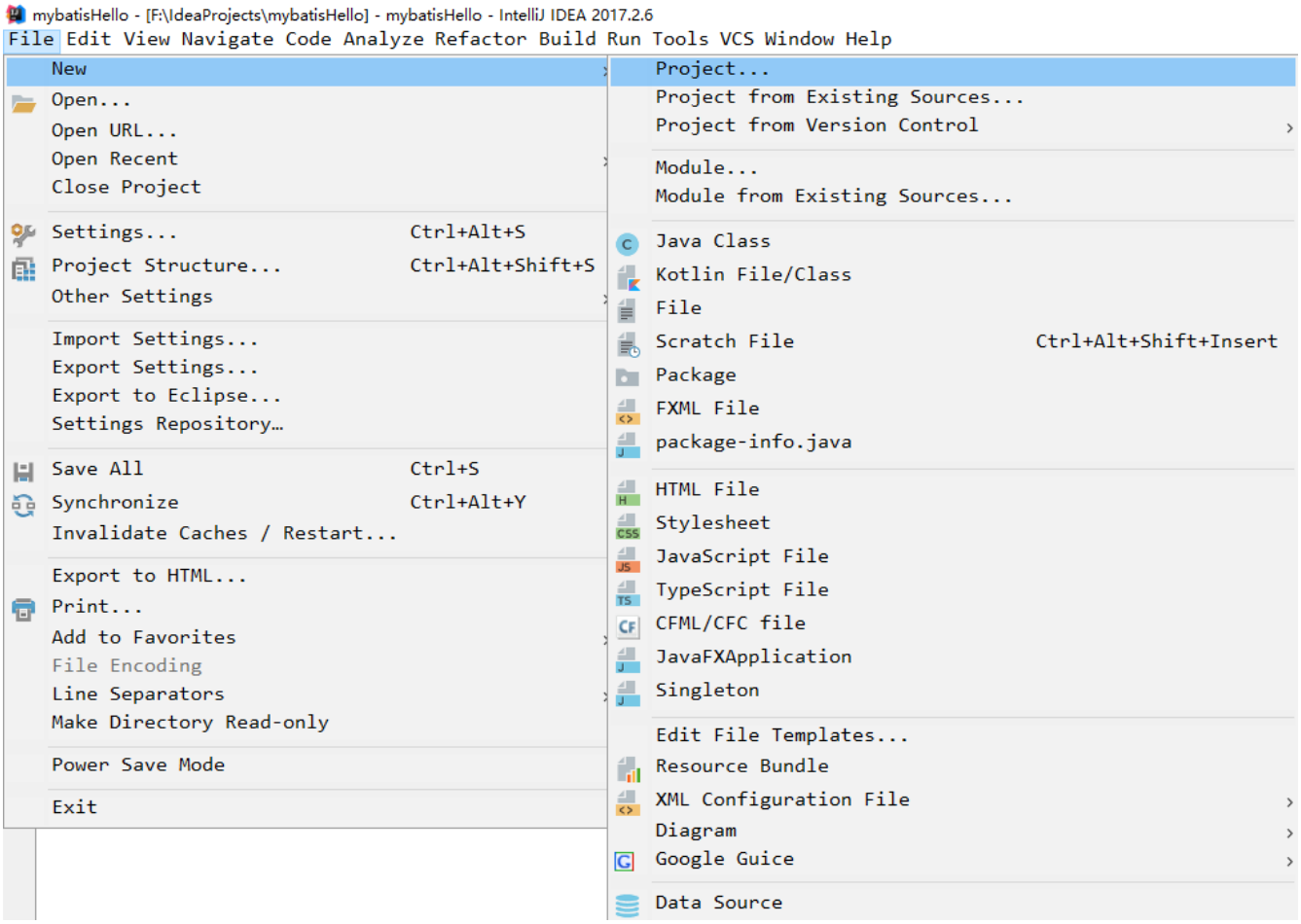
4.1 orm思想

就是把数据库和实体类的属性进行对应，让我们可以操作实体类就可以操作数据库表。

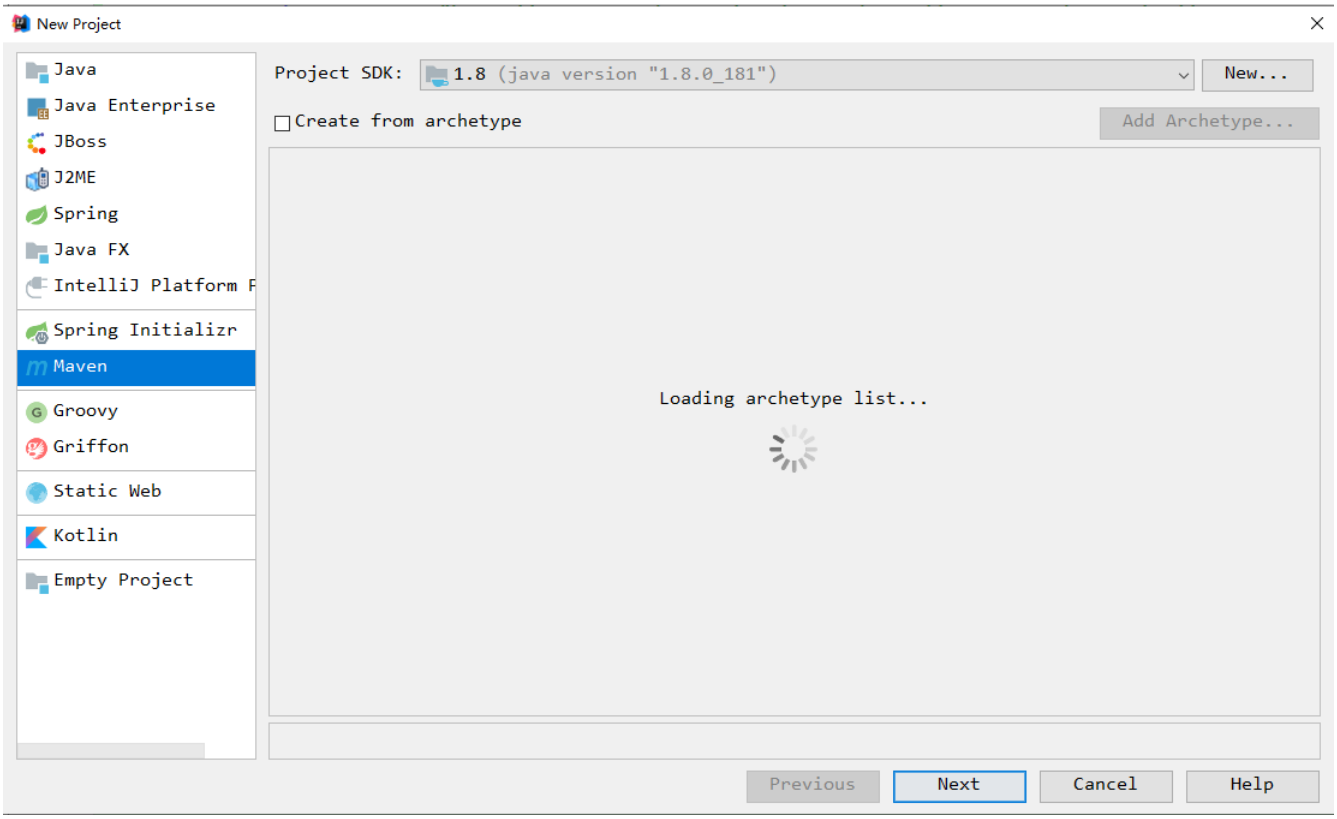
五、Myabtis的入门

5.1 创建Maven工程

在idea中选择file->new ->project



选择maven



点击下一步，填写GroupId和ArtifactId，完成

打包方式改成jar

```
<groupId>com.mybatis</groupId>
<artifactId>mybatisHello</artifactId>
<version>1.0-SNAPSHOT</version>
<packaging>jar</packaging>
```

5.2 添加maven坐标

```
<dependency>
            <groupId>org.mybatis</groupId>
            <artifactId>mybatis</artifactId>
            <version>3.4.5</version>
        </dependency>
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>5.1.6 </version>
        </dependency>
        <dependency>
            <groupId>log4j</groupId>
            <artifactId>log4j</artifactId>
            <version>1.2.12</version>
        </dependency>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.10</version>
```

```
</dependency>
```

5.3 创建实体类和接口文件

Student

```
package com.wwxy.pojo;

import java.util.Date;

public class Student {

    private Integer id;

    private String name;

    private Integer age;

    private Date birthDate;

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public Integer getAge() {
        return age;
    }

    public void setAge(Integer age) {
        this.age = age;
    }

    public Date getBirthDate() {
        return birthDate;
    }

    public void setBirthDate(Date birthDate) {
        this.birthDate = birthDate;
    }
}
```

StudentDao

```
package com.wwxy.dao;

import com.wwxy.pojo.Student;

import java.util.List;

public interface StudentDao {

    List<Student> findAll();

}
```

sqlMapConfig.xml（在resources下创建文件）Mybatis的环境配置后期和Spring整合后会配置到Spring的配置文件中。

```
<?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">
<!--Mybatis的主配置文件-->
<configuration>
    <!--配置环境 -->
    <environments default="mysql">
        <!-- 配置Mysql环境-->
        <environment id="mysql">
            <!--配置事务类型-->
            <transactionManager type="JDBC"></transactionManager>
            <!--配置数据库连接池-->
            <dataSource type="POOLED">
                <!--配置连接数据库的4个基本信息-->
                <property name="driver" value="com.mysql.jdbc.Driver"/>
                <property name="url" value="jdbc:mysql://localhost:3306/boot"/>
                <property name="username" value="root"/>
                <property name="password" value="root"/>
            </dataSource>
        </environment>
    </environments>
    <!--定义Mapper文件的位置-->
    <mappers>
        <mapper resource="com/wwxy/dao/StudentDao.xml"/>
    </mappers>
</configuration>
```

StudentDao.xml(在idea的resources下创建com/wwxy/dao，一次创建一个目录)

```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.wwxy.dao.StudentDao">
    <!--标签中的id值必须和接口文件中和其对应的方法名称一致-->
    <select id="findAll" resultType="com.wwxy.pojo.Student">
        SELECT  * from student;
    </select>
</mapper>
```

5.4 Mybatis中需要注意的问题

1. 创建的映射文件的名称和接口得到名称可以叫XXXDao.java或XXXDao.xml，在Mybatis中它把持久层的操作接口叫做Mapper，所以也可以叫XXXMapper
2. 在idea中创建目录时，它和包是不一样的，在创建包时：com.wwxy.dao是三级结构，在创建目录时：com.wwxy.dao是一级目录
3. Mybatis的映射文件的位置必须和dao接口的包接口一致，也可以将其放到同一个包下
4. 映射文件的mapper标签的namespace属性的值必须是dao包在的接口的全限定名称
5. 映射文件的操作配置标签的id值必须和dao接口对应的方法的名称一致

   当遵循3、4、5之后，我们在开发中就无需再写dao方法的实现类

5.5 测试

```java
public static void main(String[] args) throws IOException {
        //1. 读取配置文件
        InputStream in = Resources.getResourceAsStream("sqlMapConfig.xml");
        //2. 创建sqlSessionFactory
        SqlSessionFactoryBuilder builder = new SqlSessionFactoryBuilder();
        SqlSessionFactory sqlSessionFactory = builder.build(in);
        //3. 使用工厂生产sqlsession对象
        SqlSession sqlSession = sqlSessionFactory.openSession();
        //4. sqlsession创建dao接口的代理对象
        StudentDao studentDao = sqlSession.getMapper(StudentDao.class);
        //5. 使用代理对象执行方法
        List<Student> list = studentDao.findAll();
        for (int i = 0; i < list.size(); i++) {
            System.out.println(list.get(i).toString());
        }
        //6. 释放资源
        sqlSession.close();
        in.close();
    }
```

结果：

```
Student{id=1, name='小红', age=6, birthDate=null}
```

六、注解入门

6.1 删除映射文件

删除com/wwxy/dao下的StudentDao.xml

6.2 修改主配置文件的Mapper属性的值

```
<?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">
<!--Mybatis的主配置文件-->
<configuration>
    <!--配置环境 -->
    <environments default="mysql">
        <!-- 配置Mysql环境-->
        <environment id="mysql">
            <!--配置事务类型-->
            <transactionManager type="JDBC"></transactionManager>
            <!--配置数据库连接池-->
            <dataSource type="POOLED">
                <!--配置连接数据库的4个基本信息-->
                <property name="driver" value="com.mysql.jdbc.Driver"/>
                <property name="url" value="jdbc:mysql://localhost:3306/boot"/>
                <property name="username" value="root"/>
                <property name="password" value="root"/>
            </dataSource>
        </environment>
    </environments>
    <!--定义Mapper文件的位置-->
    <mappers>
        <!--如果采用注解配置的话，mapper标签的属性修改为class，值为dao接口的全限定类名-->
        <mapper class="com.wwxy.dao.StudentDao"/>
    </mappers>
</configuration>
```

6.3 在dao接口中添加注解

在dao接口对应的方法上面添加：@Select("select * from student")

注意：在实际开发中，都是越简单越好，Mybatis支持以实现类的方式进行开发，也支持使用代理方式进行开发，但实际开发中，一般不使用实现类方式进行开发。

七、以实现类的方式使用MyBatis

7.1 保持xml入门的配置不变

7.2 创建dao接口的实现类

```
package com.wwxy.dao.impl;

import com.wwxy.dao.StudentDao;
import com.wwxy.pojo.Student;
import org.apache.ibatis.session.SqlSession;
import org.apache.ibatis.session.SqlSessionFactory;

import java.util.List;

public class StudentDaoImpl implements StudentDao {

    private SqlSessionFactory factory;

    public StudentDaoImpl(SqlSessionFactory factory) {
        this.factory = factory;
    }

    public List<Student> findAll() {
        SqlSession sqlSession = factory.openSession();
        List<Student> list = sqlSession.selectList("com.wwxy.dao.StudentDao.findAll");//参数指定了使用
哪个映射文件下的哪个方法
        sqlSession.close();
        return list;
    }
}
```

7.2 修改测试类

```
public static void main(String[] args) throws IOException {
        //1. 读取配置文件
        InputStream in = Resources.getResourceAsStream("sqlMapConfig.xml");
        //2. 创建sqlSessionFactory
        SqlSessionFactoryBuilder builder = new SqlSessionFactoryBuilder();
        SqlSessionFactory sqlSessionFactory = builder.build(in);
        //3. 创建dao实现类，传入sqlSessionFactory
        StudentDao studentDao = new StudentDaoImpl(sqlSessionFactory);
        //4. 调用实现类的方法
        List<Student> list = studentDao.findAll();
        for (int i = 0; i < list.size(); i++) {
            System.out.println(list.get(i).toString());
        }
        in.close();
    }
```

八、入门案例中的设计模式分析

8.1 入门代码

```
public static void main(String[] args) throws IOException {
        //1. 读取配置文件
        InputStream in = Resources.getResourceAsStream("sqlMapConfig.xml");
        //2. 创建sqlSessionFactory
        SqlSessionFactoryBuilder builder = new SqlSessionFactoryBuilder();
        SqlSessionFactory sqlSessionFactory = builder.build(in);
        //3. 使用工厂生产sqlsession对象
        SqlSession sqlSession = sqlSessionFactory.openSession();
        //4. sqlsession创建dao接口的代理对象
        StudentDao studentDao = sqlSession.getMapper(StudentDao.class);
        //5. 使用代理对象执行方法
        List<Student> list = studentDao.findAll();
        for (int i = 0; i < list.size(); i++) {
            System.out.println(list.get(i).toString());
        }
        //6. 释放资源
        sqlSession.close();
        in.close();
    }
```

8.2 读取配置文件

```
InputStream in = Resources.getResourceAsStream("sqlMapConfig.xml");
```

在读取配置文件时，使用绝对路径会因为服务的移动而导致配置文件找不到，使用相对路径，如/src/java/main/xxx.xml，有时也会因为部署项目后找不到配置文件，通常使用：

1. 使用类加载器，它只能读取类路径下的配置文件
2. 使用ServletContext对象的getRealPath()

8.3 创建sqlSessionFactoryBuilder

```
InputStream in = Resources.getResourceAsStream("sqlMapConfig.xml");
SqlSessionFactoryBuilder builder = new SqlSessionFactoryBuilder();
SqlSessionFactory sqlSessionFactory = builder.build(in);
```

这里使用了建造者设计模式。这里记录一个常见的建造者模式的代码

8.4 建造者设计模式

Course的建造者

```
package com.design.mode.builder.v2;

/**
 * 演进版建造者模式
 */
public class Course {

    private String courseName;
```

```
private String courseVideo;

private String courseNote;

private String courseQA;

public Course(CourseBuilder courseBuilder) {
    this.courseName=courseBuilder.courseName;
    this.courseVideo=courseBuilder.courseVideo;
    this.courseNote=courseBuilder.courseNote;
    this.courseQA=courseBuilder.courseQA;
}

public String getCourseName() {
    return courseName;
}

public void setCourseName(String courseName) {
    this.courseName = courseName;
}

public void setCourseVideo(String courseVideo) {
    this.courseVideo = courseVideo;
}

public void setCourseNote(String courseNote) {
    this.courseNote = courseNote;
}

public void setCourseQA(String courseQA) {
    this.courseQA = courseQA;
}

public String getCourseVideo() {
    return courseVideo;
}

public String getCourseNote() {
    return courseNote;
}

public String getCourseQA() {
    return courseQA;
}

@Override
public String toString() {
    return "Course{" +
            "courseName='" + courseName + '\'' +
            ", courseVideo='" + courseVideo + '\'' +
            ", courseNote='" + courseNote + '\'' +
            ", courseQA='" + courseQA + '\'' +
```

```
                '}';
    }


    public static class  CourseBuilder{
        private String courseName;

        private String courseVideo;

        private String courseNote;

        private String courseQA;


        public CourseBuilder setCourseName(String courseName){
            this.courseName=courseName;
            return this;
        }

        public CourseBuilder setCourseVideo(String courseVideo) {
            this.courseVideo=courseVideo;
            return this;
        }
        public CourseBuilder setCourseNote(String courseNote) {
            this.courseNote=courseNote;
            return this;
        }

        public CourseBuilder setCourseQA(String courseQA) {
            this.courseQA=courseQA;
            return this;
        }

        public Course  build(){
            return new Course(this);
        }


    }


}
```

Test

```
package com.design.mode.builder.v2;

public class Test {

    public static void main(String[] args) {
        Course course = new Course.CourseBuilder().setCourseName("JavaV2").setCourseNote("JavaV2手
记").build();
        System.out.println(course);
    }
}
```

建造者模式的优势：隐藏了对象的创建细节，使得调用者只关注调用的方法就可以获取对象。

8.5 创建sqlSession对象

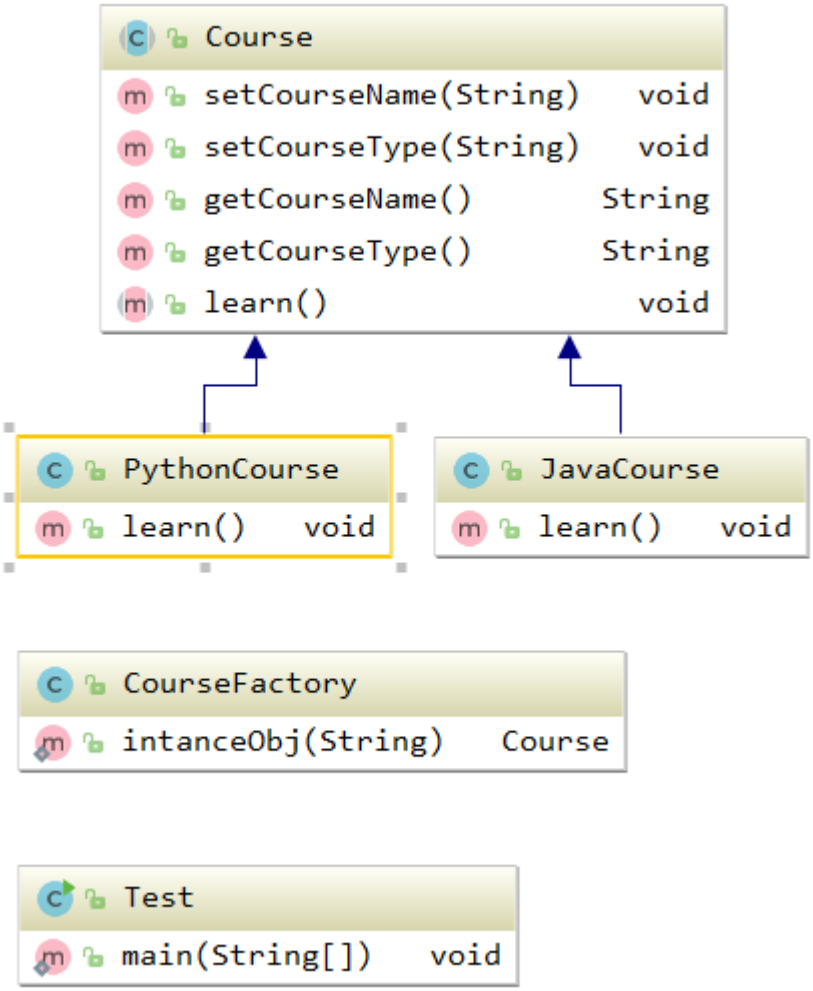
```
SqlSession sqlSession = sqlSessionFactory.openSession();
```

使用了工厂模式，好处是解耦合，降低类与类之间的依赖关系。

8.6 工厂模式

8.6.1 简单工厂

类图如下



Course

```
package com.design.mode.simpleFactory;

public abstract class Course {
    private String CourseName;

    public void setCourseName(String courseName) {
        CourseName = courseName;
    }

    public void setCourseType(String courseType) {
        CourseType = courseType;
    }

    private String CourseType;

    public String getCourseName() {
        return CourseName;
    }

    public String getCourseType() {
```

```
        return CourseType;
    }

    public abstract void  learn();
}
```

CourseFactory

```
package com.design.mode.simpleFactory;

public class CourseFactory {

    public static Course  intanceObj(String type){
        if(type.equalsIgnoreCase("java")){
            return new JavaCourse();
        }
        else if(type.equalsIgnoreCase("python")){
            return new PythonCourse();
        }
        return null;
    }

}
```

JavaCourse

```
package com.design.mode.simpleFactory;

public class JavaCourse extends Course {
    @Override
    public void learn() {
        System.out.println("学习Java");

    }
}
```

PythonCourse

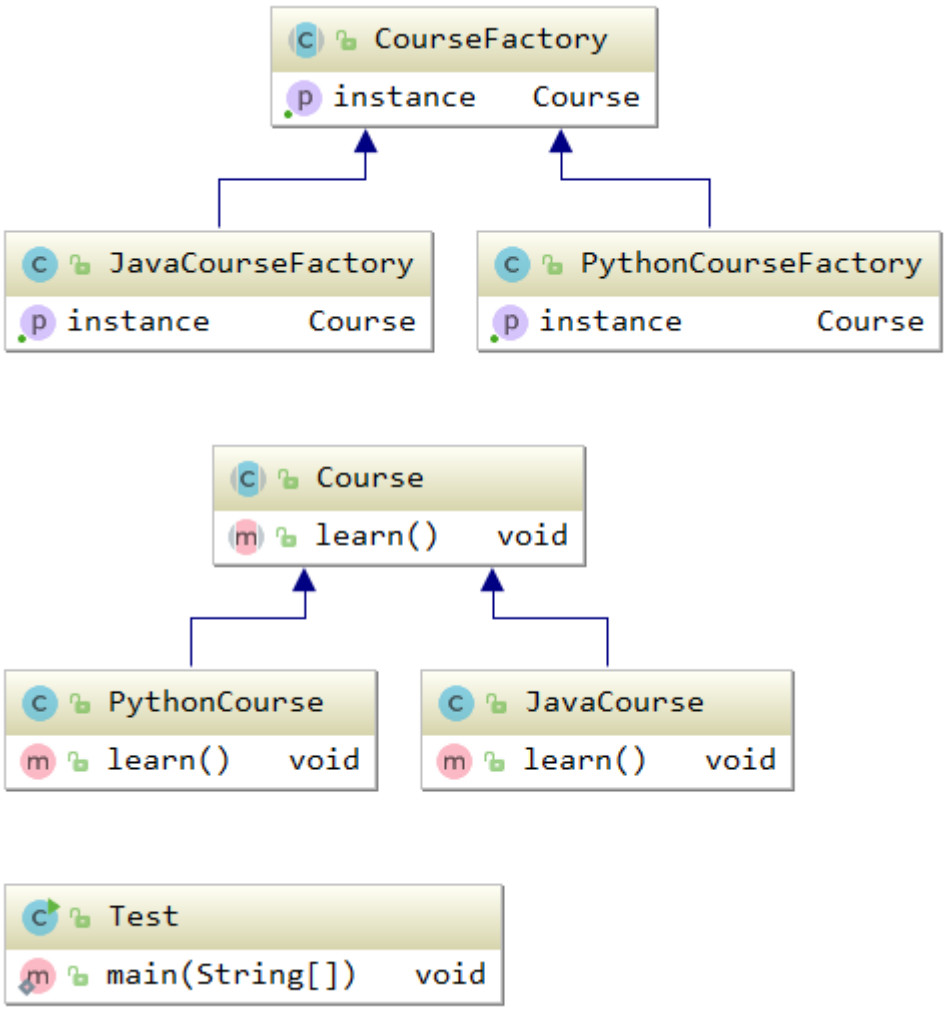
```
package com.design.mode.simpleFactory;

public class PythonCourse extends  Course {
    @Override
    public void learn() {
        System.out.println("学习python");
    }
}
```

Test

```
package com.design.mode.simpleFactory;

/**
 * 测试简单工厂模式
 */
public class Test {

    public static void main(String[] args) {

        Course course = CourseFactory.intanceObj("Java");
        course.learn();

    }

}
```

8.5.2 工厂方法

类图



Course

```
package com.design.mode.factoryMethod;

public abstract class Course {
    public abstract void  learn();
}
```

JavaCourse

```
package com.design.mode.factoryMethod;

public class JavaCourse extends  Course{

    @Override
    public void learn() {
        System.out.println("学习Java课程");
    }
}
```

PythonCourse

```
package com.design.mode.factoryMethod;

public class PythonCourse extends  Course {
    @Override
    public void learn() {
        System.out.println("学习Python课程");
    }
}
```

CourseFactory

```
package com.design.mode.factoryMethod;

public abstract class CourseFactory {
    public abstract Course getInstance();
}
```

JavaCourseFactory

```
package com.design.mode.factoryMethod;

public class JavaCourseFactory extends CourseFactory {
    @Override
    public Course getInstance() {
        return new JavaCourse();
    }
}
```

PythonCourseFactory

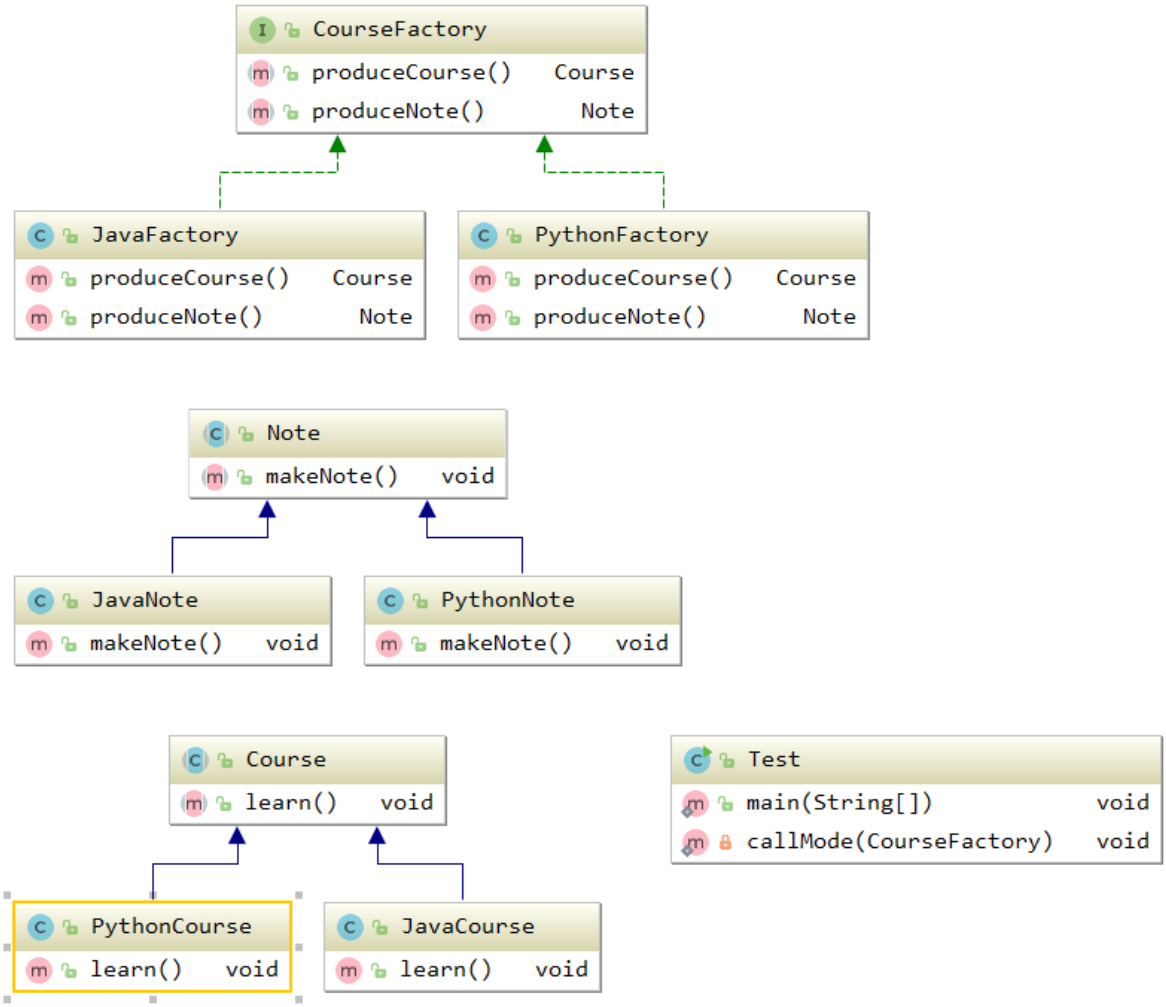
```
package com.design.mode.factoryMethod;

public class PythonCourseFactory extends CourseFactory {
    @Override
    public Course getInstance() {
        return new PythonCourse();
    }
}
```

Test

```
package com.design.mode.factoryMethod;

/**
 * 测试工厂方法设计模式
 */
public class Test {
    public static void main(String[] args) {
        CourseFactory courseFactory = new PythonCourseFactory();  //创建工厂对象
        Course course = courseFactory.getInstance();  //通过工厂对象获取对象
        course.learn();
    }
}
```

8.5.3 抽象工厂

类图



Course

```
package com.design.mode.abstractFactory;

public abstract class Course {

    public abstract void learn();

}
```

JavaCourse

```
package com.design.mode.abstractFactory;


public class JavaCourse extends Course{

    @Override
    public void learn() {
        System.out.println("学习Java课程");
    }
}
```

CourseFactory

```
package com.design.mode.abstractFactory;

public interface CourseFactory {


    public Course  produceCourse();

    public Note   produceNote();

}
```

JavaFactory

```
package com.design.mode.abstractFactory;

public class JavaFactory implements CourseFactory {
    @Override
    public Course produceCourse() {
        return new JavaCourse();
    }

    @Override
    public Note produceNote() {
        return new JavaNote();
    }
}
```

JavaNote

```
package com.design.mode.abstractFactory;

public class JavaNote extends Note {
    @Override
    public void makeNote() {
        System.out.println("做Java课程笔记");
        }
}
```

Note

```
package com.design.mode.abstractFactory;

public abstract class  Note {

    public abstract void  makeNote();
}
```

PythonCourse

```
package com.design.mode.abstractFactory;

public class PythonCourse extends Course {
    @Override
    public void learn() {
        System.out.println("学习Python课程");
    }
}
```

PythonFactory

```
package com.design.mode.abstractFactory;

public class PythonFactory implements   CourseFactory {
    @Override
    public Course produceCourse() {
        return new PythonCourse();
    }

    @Override
    public Note produceNote() {
        return new PythonNote();
    }
}
```

PythonNote

```
package com.design.mode.abstractFactory;

public class PythonNote extends Note{

    @Override
    public void makeNote() {
        System.out.println("做Python课程笔记");
    }
}
```

Test

```
package com.design.mode.abstractFactory;

/**
 * 测试抽象工厂
 */
public class Test {

    public static void main(String[] args) {
//        CourseFactory factory = new PythonFactory();
        CourseFactory factory = new JavaFactory();
        callMode(factory);
    }

    private static void callMode(CourseFactory factory) {
        Course course = factory.produceCourse();
        Note note = factory.produceNote();
        course.learn();
        note.makeNote();
    }
}
```

8.7 通过代理获得dao的代理接口

```
  StudentDao studentDao = sqlSession.getMapper(StudentDao.class);
```

这里使用了代理模式，好处是在不修改源码的基础上对已有的方法进行增强

8.9 动态代理

（后续补充）

九、Mybatis的CRUD

9.1 code

StudentDao

```
package com.wwxy.dao;

import com.wwxy.pojo.Student;
import org.apache.ibatis.annotations.Select;

import java.util.List;

public interface StudentDao {

    List<Student> findAll();

    void update(Student student);

    void insert(Student student);

    void delete(Integer id);

    Student listLikeName(String name);
}
```

StudentDao.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.wwxy.dao.StudentDao">

    <select id="findAll" resultType="com.wwxy.pojo.Student">
        SELECT  * from student;
    </select>

    <update id="update" parameterType="com.wwxy.pojo.Student">
        update student set name =#{name},age=#{age},school=#{school} where id=#{id}
    </update>

    <insert id="insert" parameterType="com.wwxy.pojo.Student">
        <!--设置新增后返回id-->
        <selectKey keyProperty="id" keyColumn="id" resultType="int" order="AFTER">
            SELECT  last_insert_id();
        </selectKey>
        insert into student (name,age,school) VALUES (#{name},#{age},#{school});
    </insert>

    <delete id="delete" parameterType="int">
        DELETE from student where id=#{id};
    </delete>

    <select id="listLikeName" parameterType="java.lang.String" resultType="com.wwxy.pojo.Student">
        <!--select * from student where name like '%${value}%' 使用字符串拼接的方式进行模糊查询-->
        SELECT  * from student where name like #{name} <!--使用占位符的方式进行模糊查询-->
```

```
    </select>
</mapper>
```

Test

```
package com.wwxy.test;

import com.wwxy.dao.StudentDao;
import com.wwxy.pojo.Student;
import org.apache.ibatis.io.Resources;
import org.apache.ibatis.session.SqlSession;
import org.apache.ibatis.session.SqlSessionFactory;
import org.apache.ibatis.session.SqlSessionFactoryBuilder;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;

import java.io.IOException;
import java.io.InputStream;
import java.util.List;

public class MybatisTest {

    private SqlSession sqlSession;

    private InputStream in;

    private StudentDao studentDao;

    @Before
    public void init() throws IOException {
        if (sqlSession == null) {
            //1. 读取配置文件
            in = Resources.getResourceAsStream("sqlMapConfig.xml");
            //2. 创建sqlSessionFactory
            SqlSessionFactoryBuilder builder = new SqlSessionFactoryBuilder();
            SqlSessionFactory sqlSessionFactory = builder.build(in);
            sqlSession = sqlSessionFactory.openSession();
            studentDao = sqlSession.getMapper(StudentDao.class);
        }

    }

    @After
    public void close() throws IOException {
        sqlSession.commit();
        sqlSession.close();
        in.close();
    }

    @Test
    public void list() {
        List<Student> all = studentDao.findAll();
    }
```

```
    @Test
    public void save() {
        Student student = new Student();
        student.setName("wangwu");
        student.setAge(20);
        System.out.println(student.toString());
        studentDao.insert(student);
        System.out.println(student.toString());
    }

    @Test
    public void update(){
        Student student = new Student();
        student.setId(3);
        student.setName("hehe");
        student.setAge(20);
        studentDao.update(student);
    }

    @Test
    public void delete() {
        studentDao.delete(2);
    }

    @Test
    public void selectByName() {
        Student student = studentDao.listLikeName("%三%");
        System.out.println(student.toString());
    }


}
```

此外selectKey标签用于在数据库设置id自增长情况下，在插入数据时返回插入数据的id，并将其set到对象的id属性中

```
<selectKey keyProperty="id" keyColumn="id" resultType="int" order="AFTER">
            SELECT  last_insert_id();
</selectKey>
```

十、OGNL表达式

10.1 定义

OGNL表示对象（Object）、图（Graphic）、导航（Naviation）、语言（Language）

10.2 作用

它是通过对象的取值方法来获取数据。写法上吧get给省略了。

如：获取用户的名称：

类中的写法：user.getName();

OGNL：user.name;

在MyBatis中为什么写sql语句时#{}里的值不用写'user.',而是直接写属性名称呢？因为在parameterType中已经指定了属性所属的类，所以不需要再写了对象名了

十一、参数传递

11.1 传递自定义包装对象

创建StudentQuery对象用于存放查询条件，然后传递StudentQuery对象，实现根据学生名称查询学生对象

11.2 代码

StudentDao.xml

```
<select id="queryLikeName" parameterType="com.wwxy.pojo.StudentQuery"
resultType="com.wwxy.pojo.Student">
SELECT  * from student where name like #{student.name} <!--使用占位符的方式进行模糊查询-->
</select>
```

StudentQuery

```
package com.wwxy.pojo;

public class StudentQuery {

    private Student student;

    public Student getStudent() {
        return student;
    }

    public void setStudent(Student student) {
        this.student = student;
    }
}
```

StudentDao

```
 Student queryLikeName(StudentQuery studentQuery);
```

Test

```
@Test
    public void queryByName(){
        StudentQuery studentQuery = new StudentQuery();
        Student student = new Student();
        student.setName("%三%");
        studentQuery.setStudent(student);
        Student student1 = studentDao.queryLikeName(studentQuery);
        System.out.println(student1.toString());
    }
```

11.3 注意

当参数传参是基本数据类型时，#{}里的名称可以随意定义，如果是Map或者是Pojo，则需要和Map或POJO里的属性名称保持一致。

11.4 #{}和${}的区别

#{}是占位符，可以防止sql注入，而${}是字符串拼接符，不能防止sql注入，

十二、结果封装

12.1 ResultType

当数据库字段名称和pojo对象的属性名称一致时，采用ResultType属性来接收查询的数据，并将其封装到pojo对象中或者基本数据类型中。

12.2 ResultMap

用于解决数据库字段和实体类属性名称不一致时的情况，用于将其进行一一映射

```
    <resultMap id="studentMap" type="com.wwxy.pojo.Student">
        <!--主键字段的对应-->
        <id property="sid" column="id"></id>
        <!--其他字段的对应-->
        <result property="sname" column="name"></result>
        <result property="sage" column="age"></result>
        <result property="sschool" column="school"></result>

    </resultMap>

    <select id="findAll" resultMap="studentMap">
        SELECT  * from student;
    </select>
```

十三、sqlMapConfig.xml的一些标签

13.1 properties标签

通过properties标签，可以将datasource标签里的属性提取出来，同时也支持读取外部配置文件

13.1.1 resource属性

其resource属性用于指定配置文件的路径，需要按照类路径解析的，所以配置文件必须放到类路径下

```
<?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">
<!--Mybatis的主配置文件-->
<configuration>

    <properties resource="jdbc.properties">

    </properties>
    <!--配置环境 -->
    <environments default="mysql">
        <!-- 配置Mysql环境-->
        <environment id="mysql">
            <!--配置事务类型-->
            <transactionManager type="JDBC"></transactionManager>
            <!--配置数据库连接池-->
            <dataSource type="POOLED">
                <!--配置连接数据库的4个基本信息-->
                <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>
    <!--定义Mapper文件的位置-->
    <mappers>
        <mapper resource="com/wwxy/dao/StudentDao.xml"/>
    </mappers>
</configuration>
```

jdbc.properties

```
jdbc.driver=com.mysql.jdbc.Driver
jdbc.url=jdbc:mysql://localhost:3306/boot
jdbc.user=root
jdbc.password=root
```

13.1.2 url属性

要求配置文件的路径按照URL的格式编写

什么是url?

URL：Uniform Resource Locator 统一资源定位符，它可以唯一标识一个资源的位置，

写法如下：

http://localhost:8080/student/list

协议 ip 端口 URI

什么是URI?

Uniform Resource Identifier:统一资源标识符，它是在应用中唯一定位一个资源

url属性支持使用file协议，Windows系统的文件系统默认实现了file协议，所以url的值可以是window下的某个文件的绝对路径。

```
<?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">
<!--Mybatis的主配置文件-->
<configuration>

    <properties url="c:/jdbc.properties">

    </properties>
    <!--配置环境 -->
    <environments default="mysql">
        <!-- 配置Mysql环境-->
        <environment id="mysql">
            <!--配置事务类型-->
            <transactionManager type="JDBC"></transactionManager>
            <!--配置数据库连接池-->
            <dataSource type="POOLED">
                <!--配置连接数据库的4个基本信息-->
                <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>
    <!--定义Mapper文件的位置-->
    <mappers>
        <mapper resource="com/wwxy/dao/StudentDao.xml"/>
    </mappers>
</configuration>
```

13.2 typeAliases标签和package标签

typeAliases标签的typeAlias用于配置一个类的别名，type属性是实体类的全限定名称，alias属性是指定的别名，别名不区分大小写.

typeAliases标签的package用于指定某个包来配置别名，指定后，该包下的所有实体类均有别名，实体类的类名就是别名，不区分大小写

mappers标签里的package标签用于指定dao接口所在的包，会自动去找对应的dao接口。

```
<?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">
<!--Mybatis的主配置文件-->
<configuration>


    <properties resource="jdbc.properties"></properties>
    <typeAliases>
        <!--
            typeAlias用于配置一个类的别名，type属性是实体类的全限定名称，alias属性是指定的别名，别名不区分大小写
        -->
        <typeAlias type="com.wwxy.pojo.Student" alias="student"></typeAlias>

        <!--
            用于指定某个包来配置别名，指定后，该包下的所有实体类均有别名，实体类的类名就是别名，不区分大小写
        -->
        <package name="com.wwxy.pojo"></package>

    </typeAliases>
    <!--配置环境 -->
    <environments default="mysql">
        <!-- 配置Mysql环境-->
        <environment id="mysql">
            <!--配置事务类型-->
            <transactionManager type="JDBC"></transactionManager>
            <!--配置数据库连接池-->
            <dataSource type="POOLED">
                <!--配置连接数据库的4个基本信息-->
                <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>
    <!--定义Mapper文件的位置-->
    <mappers>
        <!--mapper标签用于指定一个xml文件的路径-->
        <!--<mapper resource="com/wwxy/dao/StudentDao.xml"/> -->

        <!--package用于指定dao接口所在的包，指定后就不需要写mapper标签了-->
        <package name="com.wwxy.dao"></package>
    </mappers>
</configuration>
```

十四、MyBatis中的连接池

Mybatis的连接池提供了三种方式进行配置

14.1 配置的位置

在sqlMapConfig.xml中的dataSource标签中，type属性是表示采用何种连接池方式

取值: POOLED, UNPOOLED, JNDI

1. POOLED: 采用传统的javax.sql.DataSource规范中的连接池，mybatis中有针对规范的实现
2. UNPOOLED:采用传统的获取连接的方式，虽然也实现了javax.sql.DataSource接口，但并没有使用连接池的思想。
3. JNDI： 采用服务器提供的JNDI技术获取DataSource,不同的服务器所能获取的DataSource是不一样的，注意：如果不是web工程或war包，无法使用，tomcat服务器获取的连接池是dbcp

14.2 使用POLLED

使用POOLED的数据源，会使用连接池

PooledDatasource

```
public class PooledDataSource implements DataSource {
    ...
    private final PoolState state = new PoolState(this);
    ...
    @Override
  public Connection getConnection() throws SQLException {
    return popConnection(dataSource.getUsername(),
dataSource.getPassword()).getProxyConnection();
  }

    private PooledConnection popConnection(String username, String password) throws SQLException {
    boolean countedWait = false;
    PooledConnection conn = null;
    long t = System.currentTimeMillis();
    int localBadConnectionCount = 0;

    while (conn == null) {
      synchronized (state) {
        if (!state.idleConnections.isEmpty()) {    //如果存放空闲连接的集合不是空的
          // Pool has available connection
          conn = state.idleConnections.remove(0); //将集合中第一个元素删除，并将其返回给conn
          if (log.isDebugEnabled()) {
            log.debug("Checked out connection " + conn.getRealHashCode() + " from pool.");
          }
        } else {
          // 在没有空闲连接的情况下，先判断活动的连接数是否小于允许活动的最大连接数
          if (state.activeConnections.size() < poolMaximumActiveConnections) {
            // 创建一个新的数据库连接
            conn = new PooledConnection(dataSource.getConnection(), this);
            if (log.isDebugEnabled()) {
              log.debug("Created connection " + conn.getRealHashCode() + ".");
            }
          } else {
```

```
            //若已经大于了最大数限制，就从活动连接集合中去第一个元素
            PooledConnection oldestActiveConnection = state.activeConnections.get(0);
            long longestCheckoutTime = oldestActiveConnection.getCheckoutTime();
            if (longestCheckoutTime > poolMaximumCheckoutTime) {
              // Can claim overdue connection
              state.claimedOverdueConnectionCount++;
              state.accumulatedCheckoutTimeOfOverdueConnections += longestCheckoutTime;
              state.accumulatedCheckoutTime += longestCheckoutTime;
              state.activeConnections.remove(oldestActiveConnection);
              if (!oldestActiveConnection.getRealConnection().getAutoCommit()) {
                try {
                  oldestActiveConnection.getRealConnection().rollback();
                } catch (SQLException e) {
                  /*
                     Just log a message for debug and continue to execute the following
                     statement like nothing happend.
                     Wrap the bad connection with a new PooledConnection, this will help
                     to not intterupt current executing thread and give current thread a
                     chance to join the next competion for another valid/good database
                     connection. At the end of this loop, bad {@link @conn} will be set as null.
                   */
                  log.debug("Bad connection. Could not roll back");
                }
              }
              conn = new PooledConnection(oldestActiveConnection.getRealConnection(), this);
              conn.setCreatedTimestamp(oldestActiveConnection.getCreatedTimestamp());
              conn.setLastUsedTimestamp(oldestActiveConnection.getLastUsedTimestamp());
              oldestActiveConnection.invalidate();
              if (log.isDebugEnabled()) {
                log.debug("Claimed overdue connection " + conn.getRealHashCode() + ".");
              }
            } else {
              // Must wait
              try {
                if (!countedWait) {
                  state.hadToWaitCount++;
                  countedWait = true;
                }
                if (log.isDebugEnabled()) {
                  log.debug("Waiting as long as " + poolTimeToWait + " milliseconds for
connection.");
                }
                long wt = System.currentTimeMillis();
                state.wait(poolTimeToWait);
                state.accumulatedWaitTime += System.currentTimeMillis() - wt;
              } catch (InterruptedException e) {
                break;
              }
            }
          }
        }
        if (conn != null) {
          // ping to server and check the connection is valid or not
```

```
          if (conn.isValid()) {
            if (!conn.getRealConnection().getAutoCommit()) {
              conn.getRealConnection().rollback();
            }
            conn.setConnectionTypeCode(assembleConnectionTypeCode(dataSource.getUrl(), username,
password));
            conn.setCheckoutTimestamp(System.currentTimeMillis());
            conn.setLastUsedTimestamp(System.currentTimeMillis());
            state.activeConnections.add(conn);
            state.requestCount++;
            state.accumulatedRequestTime += System.currentTimeMillis() - t;
          } else {
            if (log.isDebugEnabled()) {
              log.debug("A bad connection (" + conn.getRealHashCode() + ") was returned from the
pool, getting another connection.");
            }
            state.badConnectionCount++;
            localBadConnectionCount++;
            conn = null;
            if (localBadConnectionCount > (poolMaximumIdleConnections +
poolMaximumLocalBadConnectionTolerance)) {
              if (log.isDebugEnabled()) {
                log.debug("PooledDataSource: Could not get a good connection to the database.");
              }
              throw new SQLException("PooledDataSource: Could not get a good connection to the
database.");
            }
          }
        }
      }

    }

    if (conn == null) {
      if (log.isDebugEnabled()) {
        log.debug("PooledDataSource: Unknown severe error condition.  The connection pool returned
a null connection.");
      }
      throw new SQLException("PooledDataSource: Unknown severe error condition.  The connection
pool returned a null connection.");
    }

    return conn;
  }

}
```

```
 matches criteria [is assignable to Object]
2020-03-30 15:37:04,177 430    [           main] DEBUG ansaction.jdbc.JdbcTransaction  - Opening JDBC Connection
2020-03-30 15:37:04,411 664    [           main] DEBUG source.pooled.PooledDataSource  - Created connection 699780352.
2020-03-30 15:37:04,412 665    [           main] DEBUG ansaction.jdbc.JdbcTransaction  - Setting autocommit to false on JDBC Connection
[com.mysql.jdbc.JDBC4Connection@29b5cd00]
2020-03-30 15:37:04,414 667    [           main] DEBUG om.wwxy.dao.StudentDao.findAll  - ==>  Preparing: SELECT * from student;
2020-03-30 15:37:04,444 697    [           main] DEBUG om.wwxy.dao.StudentDao.findAll  - ==> Parameters:
2020-03-30 15:37:04,475 728    [           main] DEBUG om.wwxy.dao.StudentDao.findAll  - <==      Total: 3
[Student{id=1, name='张三', age=20, school='null'}, Student{id=3, name='hehe', age=20, school='null'}, Student{id=4, name='wangwu',
 age=20, school='null'}]
2020-03-30 15:37:04,476 729    [           main] DEBUG ansaction.jdbc.JdbcTransaction  - Resetting autocommit to true on JDBC Connection
 [com.mysql.jdbc.JDBC4Connection@29b5cd00]
2020-03-30 15:37:04,476 729    [           main] DEBUG ansaction.jdbc.JdbcTransaction  - Closing JDBC Connection [com.mysql.jdbc
 .JDBC4Connection@29b5cd00]
2020-03-30 15:37:04,477 730    [           main] DEBUG source.pooled.PooledDataSource  - Returned connection 699780352 to pool.
```

将连接归还给连接池

14.3 使用UNPOOLED连接池类型

当使用UNPOOLED类型时，Mybatis后台通过UnpooledDataSource对象创建连接，该对象每次调用都会使用jdbc创建连接，用完关闭连接

UnpooledDataSource

```
public class UnpooledDataSource implements DataSource {
    ...
  @Override
  public Connection getConnection() throws SQLException {
    return doGetConnection(username, password);
  }

private Connection doGetConnection(String username, String password) throws SQLException {
    Properties props = new Properties();
    if (driverProperties != null) {
      props.putAll(driverProperties);
    }
    if (username != null) {
      props.setProperty("user", username);
    }
    if (password != null) {
      props.setProperty("password", password);
    }
    return doGetConnection(props);
  }

  private Connection doGetConnection(Properties properties) throws SQLException {
    initializeDriver();
    Connection connection = DriverManager.getConnection(url, properties);
    configureConnection(connection);
    return connection;
  }

   private synchronized void initializeDriver() throws SQLException {
    if (!registeredDrivers.containsKey(driver)) {
      Class<?> driverType;
      try {
        if (driverClassLoader != null) {
          driverType = Class.forName(driver, true, driverClassLoader);
        } else {
          driverType = Resources.classForName(driver);
        }
```

```
        // DriverManager requires the driver to be loaded via the system ClassLoader.
        // http://www.kfu.com/~nsayer/Java/dyn-jdbc.html
        Driver driverInstance = (Driver)driverType.newInstance();
        DriverManager.registerDriver(new DriverProxy(driverInstance));
        registeredDrivers.put(driver, driverInstance);
      } catch (Exception e) {
        throw new SQLException("Error setting driver on UnpooledDataSource. Cause: " + e);
      }
    }
  }


}
```

```
2020-03-30 15:43:01,274 229    [           main] DEBUG ansaction.jdbc.JdbcTransaction  - Opening JDBC Connection
2020-03-30 15:43:01,451 406    [           main] DEBUG ansaction.jdbc.JdbcTransaction  - Setting autocommit to false on JDBC Connection
 [com.mysql.jdbc.JDBC4Connection@7a9273a8]
2020-03-30 15:43:01,458 413    [           main] DEBUG om.wwxy.dao.StudentDao.findAll  - ==>  Preparing: SELECT * from student;
2020-03-30 15:43:01,489 444    [           main] DEBUG om.wwxy.dao.StudentDao.findAll  - ==> Parameters:
2020-03-30 15:43:01,502 457    [           main] DEBUG om.wwxy.dao.StudentDao.findAll  - <==      Total: 3
[Student{id=1, name='张三', age=20, school='null'}, Student{id=3, name='hehe', age=20, school='null'}, Student{id=4, name='wangwu',
 age=20, school='null'}]
2020-03-30 15:43:01,502 457    [           main] DEBUG ansaction.jdbc.JdbcTransaction  - Resetting autocommit to true on JDBC Connection
 [com.mysql.jdbc.JDBC4Connection@7a9273a8]
2020-03-30 15:43:01,503 458    [           main] DEBUG ansaction.jdbc.JdbcTransaction  - Closing JDBC Connection [com.mysql.jdbc
 .JDBC4Connection@7a9273a8]
```

每次都会创建链接，用完归还链接

十五、事务

15.1 什么是事务?

15.2 事务的四大特性ACID

15.3 不考虑隔离会产生什么问题

15.4 事务的隔离级别

十六、动态Sql

16.1 if标签

当传入的参数是JavaBean对象或者Map，如果其含有id属性，就可以在where语句后面使用if标签。

写法:

```
<select id="findById" parameterType="Student" resultType="Student">
       SELECT  * from student where 1=1
    <if text="id != null">
       and id=#{id}
    </if>
 </select>
```

16.2 where标签

where标签可以去掉where语句的第一个and

```
<select id="findById" parameterType="Student" resultType="Student">
       SELECT  * from student
    <where>
     <if text="id != null">
       and id=#{id}
    </if>
    </where>

 </select>
```

16.3 foreach标签

foreach标签用于传入多个参数时通过循环构造成sql语句的，通常用于in语句后面的列表数据的封装和insert语句的批量插入

标签中：

collection属性是集合的名称，指的是pojo对象或map集合中，值为List的属性的名称，如果传参是List集合时，值就是list

open:指的是循环以**开始

close:指的是循环以**结束

item：循环中每一个元素的名称别名，跟标签里的#{}一致

separator:元素拼接好后以什么符号分割

xml

```
<select id="findListInIds" parameterType="list" resultType="com.wwxy.pojo.Student">
       SELECT * from student
       <where>
           <if test="list != null">
               <foreach collection="list" open=" id in (" close=")" item="id" separator=",">
                   #{id}
               </foreach>
           </if>

       </where>

   </select>
```

Dao

```
List<Student> findListInIds(List<Integer> idList);
```

Test

```
@Test
    public void queryByIds(){
        List<Integer> list = new ArrayList<Integer>();
        list.add(1);
        list.add(2);
        list.add(4);
        List<Student> listInIds = studentDao.findListInIds(list);
        System.out.println(listInIds);
    }
```

注意：当传入的参数是List，而不是pojo对象或者Map时，if标签的test属性的值不是参数的名称，foreach标签里的collection属性的值也不是参数的名称，这里怀疑是参数的类型。

当传入的参数是Map集合时，跟pojo对象一样，Mybatis内部使用了OGNL表达式，在sql中只需要写属性名称即可

如下代码：

Test

```
@Test
    public void test(){
        Map<String,String> map = new HashMap();
        map.put("userName","张三");
        List<Student> listInIds = studentDao.findByName(map);
        System.out.println(listInIds);
    }
```

Dao

```
List<Student> findByName(Map<String,String> map);
```

xml

```
  <select id="findByName" parameterType="map" resultType="com.wwxy.pojo.Student">
        SELECT * from student
        <where>
            <if test="userName != null and userName !=''">
                name = #{userName}
            </if>
        </where>
    </select>
```

16.4 sql标签

sql标签用于将相同的代码抽取出来，在需要用到的地方使用include标签引用

```
<sql id="selectSql" >
       SELECT * from student
</sql>

<select id="findAll" resultType="Student">
     <include refid="selectSql"></include>
 </select>

<select id="listLikeName" parameterType="java.lang.String" resultType="Student">
   <include refid="selectSql"></include>
   where name like #{name} <!--使用占位符的方式进行模糊查询-->
 </select>
```

十七、多表查询

17.1 多对一查询

创建学生表和教师表，学生表关联教师表的id

建表语句

```
/*
Navicat MySQL Data Transfer

Source Server         : local
Source Server Version : 50718
Source Host           : localhost:3306
Source Database       : boot

Target Server Type    : MYSQL
Target Server Version : 50718
File Encoding         : 65001

Date: 2020-03-31 10:37:08
*/

SET FOREIGN_KEY_CHECKS=0;

-- ----------------------------
-- Table structure for student
-- ----------------------------
DROP TABLE IF EXISTS `student`;
CREATE TABLE `student` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `name` varchar(255) COLLATE utf8_bin DEFAULT NULL,
  `age` int(11) DEFAULT NULL,
  `school` varchar(255) COLLATE utf8_bin DEFAULT NULL,
  `teacher_id` int(11) DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `t_key` (`teacher_id`),
  CONSTRAINT `t_key` FOREIGN KEY (`teacher_id`) REFERENCES `teacher` (`id`)
```

```
) ENGINE=InnoDB AUTO_INCREMENT=5 DEFAULT CHARSET=utf8 COLLATE=utf8_bin;

-- ----------------------------
-- Table structure for teacher
-- ----------------------------
DROP TABLE IF EXISTS `teacher`;
CREATE TABLE `teacher` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `name` varchar(10) COLLATE utf8_bin DEFAULT NULL,
  `age` int(11) DEFAULT NULL,
  `className` varchar(20) COLLATE utf8_bin DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=3 DEFAULT CHARSET=utf8 COLLATE=utf8_bin;
```

student

```
package com.wwxy.pojo;

import java.util.Date;

public class Student {

    private Integer id;

    private String name;

    private Integer age;

    private String school;

    private int teacher_id;

    public int getTeacher_id() {
        return teacher_id;
    }

    public void setTeacher_id(int teacher_id) {
        this.teacher_id = teacher_id;
    }

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
```

```
        this.name = name;
    }

    public Integer getAge() {
        return age;
    }

    public void setAge(Integer age) {
        this.age = age;
    }

    public String getSchool() {
        return school;
    }

    public void setSchool(String school) {
        this.school = school;
    }

    @Override
    public String toString() {
        return "Student{" +
                "id=" + id +
                ", name='" + name + '\'' +
                ", age=" + age +
                ", school='" + school + '\'' +
                ", teacher_id=" + teacher_id +
                '}';
    }
}
```

Teacher

```
package com.wwxy.pojo;

public class Teacher {

    private int id;

    private String name;

    private int age;

    public String getClassName() {
        return className;
    }

    public void setClassName(String className) {
        this.className = className;
    }
```

```
    private String className;

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    @Override
    public String toString() {
        return "Teacher{" +
                "id=" + id +
                ", name='" + name + '\'' +
                ", age=" + age +
                ", className='" + className + '\'' +
                '}';
    }
}
```

17.1.1 采用继承进行多对一查询

新建实体类StudentMess

```
package com.wwxy.pojo;

public class StudentMess extends Student {

    private String teacherName;

    public String getTeacherName() {
        return teacherName;
    }

    public void setTeacherName(String teacherName) {
        this.teacherName = teacherName;
```

```
    }

    @Override
    public String toString() {
        return super.toString()+"   StudentMess{" +
                "teacherName='" + teacherName + '\'' +
                '}';
    }
}
```

在StudentDao中新建方法

```
List<StudentMess> findAllStudent();
```

在StudentDao.xml里新建查询标签

```
<select id="findAllStudent" resultType="studentmess">
       select s.*,t.`name` as teacherName from student s,teacher t where s.teacher_id=t.id;
   </select>
```

17.1.2 采用Mybatis建议的方式查询

在Student对象中创建Teacher类的引用

```
package com.wwxy.pojo;

import java.util.Date;

public class Student {

    private Integer id;

    private String name;

    private Integer age;

    private String school;

    private int teacher_id;

    private Teacher teachers;

    public Teacher getTeachers() {
        return teachers;
    }

    public void setTeachers(Teacher teachers) {
        this.teachers = teachers;
    }

    public int getTeacher_id() {
        return teacher_id;
    }
```

```
    public void setTeacher_id(int teacher_id) {
        this.teacher_id = teacher_id;
    }

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public Integer getAge() {
        return age;
    }

    public void setAge(Integer age) {
        this.age = age;
    }

    public String getSchool() {
        return school;
    }

    public void setSchool(String school) {
        this.school = school;
    }

    @Override
    public String toString() {
        return "Student{" +
                "id=" + id +
                ", name='" + name + '\'' +
                ", age=" + age +
                ", school='" + school + '\'' +
                ", teacher_id=" + teacher_id +
                ", teachers=" + teachers +
                '}';
    }
}
```

在StudentDao.xml编写查询方法

```
<resultMap id="stuTeach" type="student">
        <id column="id" property="id"></id>
        <result column="name" property="name"></result>
        <result column="age" property="age"></result>
        <result column="school" property="school"></result>
        <result column="teacher_id" property="teacher_id"></result>
        <!--一对一的对象关系映射-->
        <association property="teachers" javaType="teacher">
            <result column="teacherName" property="name"></result>
        </association>
</resultMap>


    <select id="findAllStudent2" resultMap="stuTeach">
        select s.*,t.`tName` as teacherName from student s,teacher t where s.teacher_id=t.id;
    </select>
```

StudentDao

```
List<StudentMess> findAllStudent();
```

17.1.3 一对多查询

在Teacher类中添加Student类的集合

Teacher

```
package com.wwxy.pojo;

import java.util.List;

public class Teacher {

    private int id;

    private String name;

    private int age;

    private String className;

    private List<Student> students;

    public List<Student> getStudents() {
        return students;
    }

    public void setStudents(List<Student> students) {
        this.students = students;
    }

    public String getClassName() {
        return className;
```

```
    }

    public void setClassName(String className) {
        this.className = className;
    }


    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    @Override
    public String toString() {
        return "Teacher{" +
                "id=" + id +
                ", name='" + name + '\'' +
                ", age=" + age +
                ", className='" + className + '\'' +
                ", students=" + students +
                '}';
    }
}
```

TeacherDao.xml、

```
<resultMap id="tMap" type="Teacher">
        <id column="id" property="id"></id>
        <result property="name" column="tName"></result>
        <result property="age" column="age"></result>
        <result property="className" column="className"></result>
        <collection property="students" ofType="student"><!--ofType的值是集合中元素的类型-->
            <result column="sAge" property="age"></result>
            <result column="sName" property="name"></result>
```

```
            <result column="school" property="school"></result>
        </collection>
    </resultMap>

    <select id="findAllTeacher" resultMap="tMap">
        SELECT
            t.*, s.age AS sAge,
            s.`name` AS sName,
            s.school
        FROM
            teacher t,
            student s
        WHERE
            t.id = s.teacher_id;
    </select>
```

Test

```
@Test
    public void test2(){
        List<Teacher> all = teacherDao.findAllTeacher();
        for (int i = 0; i < all.size(); i++) {
            Teacher teacher = all.get(i);
            List<Student> students = teacher.getStudents();
            System.out.println(teacher);
        }
    }
```

17.1.4 多对多查询

使用用户和角色模拟

```
select u.*,r.id as rid,r.role_name,r.role_desc from user u
left outer join user_role ur  on u.id = ur.uid
left outer join role on u.id = ur.uid
```

写法跟一对多是一样的，主要是sql语句不通，根据哪一方查询，就将哪一方放到左边，使用两次左连接查询数据，封装即可

十八、MyBatis的延迟加载

18.1 延迟加载

在真正使用数据时才发起查询，不使用不查询，也称之为按需加载或者懒加载

18.2 立即加载

不管是否使用，只要一调用方法，就会发起查询。

18.3 Mybatis的数据加载时机

在查询一个对象时，与该对象关联的数据夹杂时机关键需要看该对象与关联数据的对应关系。

一对多，多对多：采用延迟加载

多对一，一对一：采用立即加载

18.4 在多对一或一对多的情况下如何实现延迟加载

一对一或多对一在默认情况下是立即加载，

在默认情况下查询学生，会将教师的信息也一块查询出来

在查询学生的所有信息的xml中进行修改

```
 <resultMap id="stuTeach" type="student">
        <id column="id" property="id"></id>
        <result column="name" property="name"></result>
        <result column="age" property="age"></result>
        <result column="school" property="school"></result>
        <result column="teacher_id" property="teacher_id"></result>
        <!--一对一的对象关系映射-->
        <association property="teachers" column="teacher_id"
select="com.wwxy.dao.TeacherDao.selectById">
        </association>
    </resultMap>


    <select id="findAllStudent2" resultMap="stuTeach">
        SELECT * from  student;
    </select>
```

其中标签的select属性是指定在进行teacher对象的映射时，调用的是哪个Mapper下的哪个方法。此时在TeacherDao.xml创建对应的方法

```
<sql id="selectAll">
         SELECT * from teacher
    </sql>
<select id="selectById" parameterType="int" resultType="Teacher">
        <include refid="selectAll"></include> where  id=#{id}
 </select>
```

Test

```
@Test
    public void test(){
        List<Student> allStudent = studentDao.findAllStudent2();
        System.out.println();
    }
```

打印结果如下：

```
2020-04-01 10:50:38,160 291    [           main] DEBUG ansaction.jdbc.JdbcTransaction  - Opening JDBC Connection
2020-04-01 10:50:38,371 502    [           main] DEBUG source.pooled.PooledDataSource  - Created connection 110431793.
2020-04-01 10:50:38,371 502    [           main] DEBUG ansaction.jdbc.JdbcTransaction  - Setting autocommit to false on JDBC Connection
 [com.mysql.jdbc.JDBC4Connection@6950e31]
2020-04-01 10:50:38,373 504    [           main] DEBUG dao.StudentDao.findAllStudent2  - ==>  Preparing: SELECT * from student;
2020-04-01 10:50:38,399 530    [           main] DEBUG dao.StudentDao.findAllStudent2  - ==> Parameters:
2020-04-01 10:50:38,415 546    [           main] DEBUG wwxy.dao.TeacherDao.selectById  - ====>  Preparing: SELECT * from teacher where
 id=?
2020-04-01 10:50:38,415 546    [           main] DEBUG wwxy.dao.TeacherDao.selectById  - ====> Parameters: 1(Integer)
2020-04-01 10:50:38,419 550    [           main] DEBUG wwxy.dao.TeacherDao.selectById  - <====      Total: 1
2020-04-01 10:50:38,421 552    [           main] DEBUG wwxy.dao.TeacherDao.selectById  - ====>  Preparing: SELECT * from teacher where
 id=?
2020-04-01 10:50:38,421 552    [           main] DEBUG wwxy.dao.TeacherDao.selectById  - ====> Parameters: 2(Integer)
2020-04-01 10:50:38,422 553    [           main] DEBUG wwxy.dao.TeacherDao.selectById  - <====      Total: 1
2020-04-01 10:50:38,422 553    [           main] DEBUG dao.StudentDao.findAllStudent2  - <==      Total: 3

2020-04-01 10:50:38,422 553    [           main] DEBUG ansaction.jdbc.JdbcTransaction  - Resetting autocommit to true on JDBC Connection
 [com.mysql.jdbc.JDBC4Connection@6950e31]
2020-04-01 10:50:38,422 553    [           main] DEBUG ansaction.jdbc.JdbcTransaction  - Closing JDBC Connection [com.mysql.jdbc
 .JDBC4Connection@6950e31]
2020-04-01 10:50:38,422 553    [           main] DEBUG source.pooled.PooledDataSource  - Returned connection 110431793 to pool.
```

显示两个sql语句，这意味着数据是立即加载的。

如何修改为延迟加载呢?

在SqlMapConfig.xml里添加setting标签配置，

```
<settings>
        <!--开启MyBatis全局延迟加载开关-->
        <setting name="lazyLoadingEnabled" value="true"/>
        <!--允许开启全局立即加载，否则就按需加载-->
        <setting name="aggressiveLazyLoading" value="false"></setting>
 </settings>
```

效果如下：

```
<?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">
<!--Mybatis的主配置文件-->
<configuration>


    <properties resource="jdbc.properties"></properties>

    <settings>
        <!--开启MyBatis全局延迟加载开关-->
        <setting name="lazyLoadingEnabled" value="true"/>
        <!--允许开启全局立即加载，否则就按需加载-->
        <setting name="aggressiveLazyLoading" value="false"></setting>
    </settings>


    <typeAliases>
        <!--
            typeAlias用于配置一个类的别名，type属性是实体类的全限定名称，alias属性是指定的别名，别名不区分大小写
        -->
        <typeAlias type="com.wwxy.pojo.Student" alias="student"></typeAlias>

        <!--
            用于指定某个包来配置别名，指定后，该包下的所有实体类均有别名，实体类的类名就是别名，不区分大小写
```

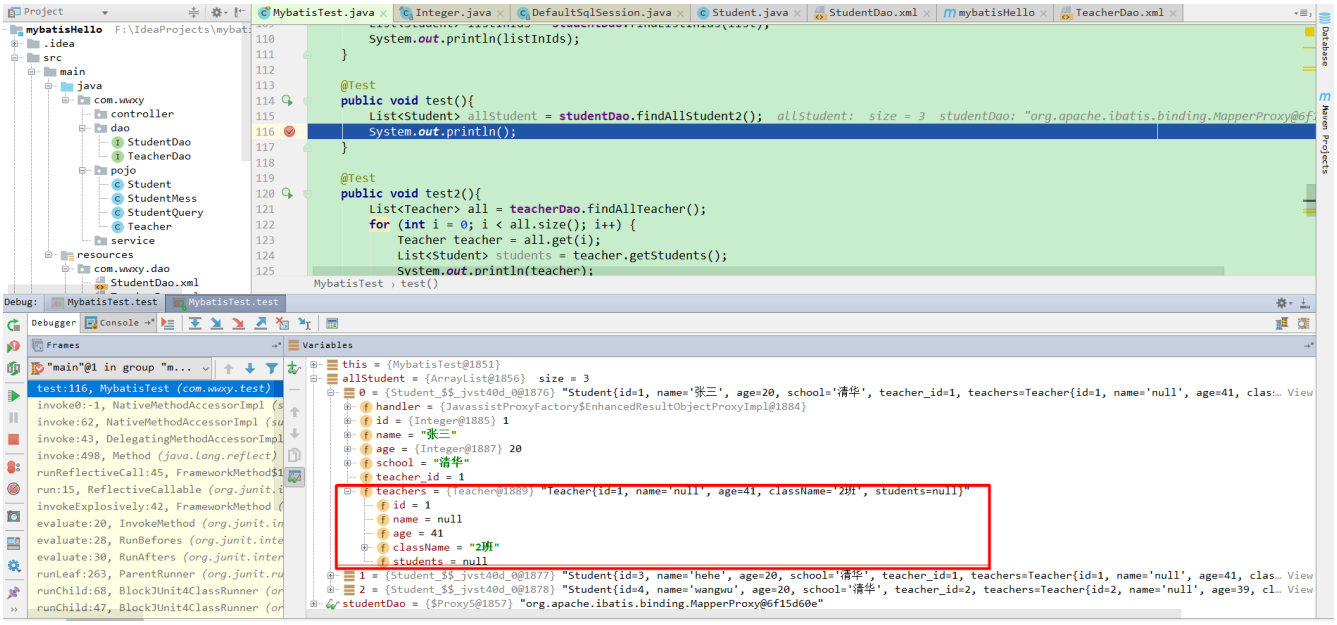
```
        -->
        <package name="com.wwxy.pojo"></package>

    </typeAliases>
    <!--配置环境 -->
    <environments default="mysql">
        <!-- 配置Mysql环境-->
        <environment id="mysql">
            <!--配置事务类型-->
            <transactionManager type="JDBC"></transactionManager>
            <!--配置数据库连接池-->
            <dataSource type="POOLED">
                <!--配置连接数据库的4个基本信息-->
                <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>
    <!--定义Mapper文件的位置-->
    <mappers>
        <!--mapper标签用于指定一个xml文件的路径-->
        <!--<mapper resource="com/wwxy/dao/StudentDao.xml"/> -->

        <!--package用于指定dao接口所在的包，指定后就不需要写mapper标签了-->
        <package name="com.wwxy.dao"></package>
    </mappers>
</configuration>
```

执行测试：结果如下

```
2020-04-01 11:04:13,937 260    [           main] DEBUG ansaction.jdbc.JdbcTransaction  - Opening JDBC Connection
2020-04-01 11:04:14,159 482    [           main] DEBUG source.pooled.PooledDataSource  - Created connection 110431793.
2020-04-01 11:04:14,160 483    [           main] DEBUG ansaction.jdbc.JdbcTransaction  - Setting autocommit to false on JDBC Connection
 [com.mysql.jdbc.JDBC4Connection@6950e31]
2020-04-01 11:04:14,162 485    [           main] DEBUG dao.StudentDao.findAllStudent2  - ==>  Preparing: SELECT * from student;
2020-04-01 11:04:14,190 513    [           main] DEBUG dao.StudentDao.findAllStudent2  - ==> Parameters:
2020-04-01 11:04:14,237 560    [           main] DEBUG dao.StudentDao.findAllStudent2  - <==      Total: 3

2020-04-01 11:04:14,237 560    [           main] DEBUG ansaction.jdbc.JdbcTransaction  - Resetting autocommit to true on JDBC Connection
 [com.mysql.jdbc.JDBC4Connection@6950e31]
2020-04-01 11:04:14,238 561    [           main] DEBUG ansaction.jdbc.JdbcTransaction  - Closing JDBC Connection [com.mysql.jdbc
 .JDBC4Connection@6950e31]
2020-04-01 11:04:14,241 564    [           main] DEBUG source.pooled.PooledDataSource  - Returned connection 110431793 to pool.

Process finished with exit code 0
```

发现现在只执行了一个sql语句，实现了延迟加载。

但是在这种情况下进行debug，发现尽管查询的是student,但teacher的信息也查询出来了，按理应该是不查到信息才对，不知道什么原因



十九、Mybatis的缓存

19.1 什么是缓存

缓存指的是存放于内存中的临时数据

19.2 为什么使用缓存

为了减少和数据库的交互次数，提高程序的执行效率

19.3 缓存的适用场景

适用于缓存：

1. 经常查询且不经常改变的数据
2. 数据的正确与否对最后的结果影响不大的数据

不适用缓存：

1. 经常改变的数据
2. 数据的正确与否对最终结果影响很大的
    1. 商品的库存
    2. 银行的汇率
    3. 股市的股价

19.4 MyBatis的一级缓存

指的是MyBatis中SqlSession对象中的缓存， 当执行查询之后，查询的结果会同时存入到sqlSession对象提供的一块区域中，该区域是一个Map，当再次查询数据时，mybatis会先去sqlsession中查询缓存里是否有该数据，有的话直接拿出来用，当sqlsession对象消失时，MyBatis的一级缓存也消失了

当使用Mapper接口的方法在同一个方法中调用两次，只要SqlSession对象不销毁，查出来的对象是同一个。

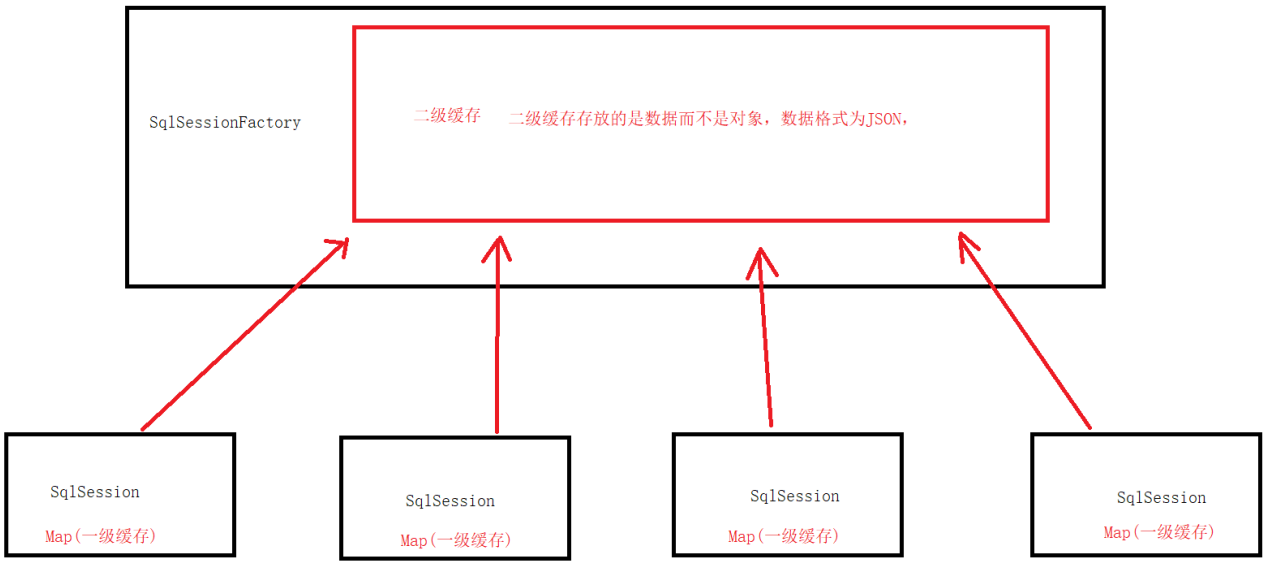
重新获取对象，查询出来的对象不是同一个。一级缓存是默认开启的，

19.5 触发清空 Mybatis一级缓存的情况

当调用SqlSession的修改、添加、删除、commit(),close()等方法时，就会清空一级缓存

19.6 二级缓存

它指的是MyBatis中的SqlSessionFactory对象的缓存，由同一个SqlSessionFactory对象创建的Sqlsession共享其缓存。



19.7 二级缓存使用步骤

使用分类：

- 让MyBatis框架支持二级缓存，需要在SqlMapConfig.xml中配置。
- 在当前的映射文件中使用二级缓存，在mapper.xml中配置
- 在当前操作中使用二级缓存，在select标签中配置

在SqlMapConfig.xml中配置cacheEnabled

```
<!--开启二级缓存，默认为true,也可以不配置-->
<setting name="cacheEnabled" value="true"/>
```

在mapper.xml中添加，在操作语句的标签上加上useCache属性，值为true

```
<cache/>

<select id="findAll" resultType="Student" useCache="true">
   <include refid="selectSql"></include>
</select>
```

二十、MyBatis的注解开发

20.1 环境搭建

pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-
4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.wwxy.mybatis.annotation</groupId>
    <artifactId>mybatis_annotation</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>jar</packaging>

    <dependencies>
        <dependency>
            <groupId>org.mybatis</groupId>
            <artifactId>mybatis</artifactId>
            <version>3.4.5</version>
        </dependency>

        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>5.1.6</version>
        </dependency>

        <dependency>
            <groupId>log4j</groupId>
            <artifactId>log4j</artifactId>
            <version>1.2.12</version>
        </dependency>

        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.10</version>
        </dependency>
    </dependencies>

</project>
```

SqlMapConfig.xml

```
<?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>
    <!--配置数据库连接信息文件位置-->
    <properties resource="jdbc.properties"></properties>

    <!--配置别名-->
    <typeAliases>
        <!--指定实体类所在的包名称-->
        <package name="com.wwxy.pojo"></package>
    </typeAliases>

    <environments default="mysql">
        <environment id="mysql">
            <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>
        <!--配置dao接口所在的包-->
        <package name="com.wwxy.dao"></package>
    </mappers>
</configuration>
```

Student

```
package com.wwxy.pojo;

import java.io.Serializable;

public class Student implements Serializable{

    private Integer id;

    private String name;

    private int age;

    private String school;

    private int teacher_id;

    public Integer getId() {
        return id;
```

```
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    public String getSchool() {
        return school;
    }

    public void setSchool(String school) {
        this.school = school;
    }

    public int getTeacher_id() {
        return teacher_id;
    }

    public void setTeacher_id(int teacher_id) {
        this.teacher_id = teacher_id;
    }

    @Override
    public String toString() {
        return "Student{" +
                "id=" + id +
                ", name='" + name + '\'' +
                ", age=" + age +
                ", school='" + school + '\'' +
                ", teacher_id=" + teacher_id +
                '}';
    }
}
```

StudentDao

```
package com.wwxy.dao;
```

```
import com.wwxy.pojo.Student;
import org.apache.ibatis.annotations.Delete;
import org.apache.ibatis.annotations.Insert;
import org.apache.ibatis.annotations.Select;
import org.apache.ibatis.annotations.Update;

import java.util.List;

public interface StudentDao {

    @Select("select * from student")
    List<Student> findAll();


    @Select("select * from student where name like #{name}")
    List<Student> findLikeByName(String name);


    @Insert("insert into student(name,age,school,teacher_id) values(#{name},#{age},#{school},#
{teacher_id})")
    void insert(Student student);

    @Update("update student set name=#{name},age=#{age} where id=#{id}")
    void update(Student student);


    @Delete("delete from student where id=#{id}")
    void delete(int id);

    @Select("select * from student where id=#{id}")
    Student selectById(Integer id);



}
```

Test

```
package com.wwxy.test;

import com.wwxy.dao.StudentDao;
import com.wwxy.pojo.Student;
import org.apache.ibatis.io.Resources;
import org.apache.ibatis.session.SqlSession;
import org.apache.ibatis.session.SqlSessionFactory;
import org.apache.ibatis.session.SqlSessionFactoryBuilder;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;

import java.io.IOException;
import java.io.InputStream;
import java.util.List;
```

```
public class TestAnnotation {

    private InputStream in;

    private SqlSessionFactory factory;

    private SqlSession session;

    private StudentDao studentDao;


    @Before
    public void init() throws Exception {
        in = Resources.getResourceAsStream("SqlMapConfig.xml");
        SqlSessionFactoryBuilder builder = new SqlSessionFactoryBuilder();
        factory = builder.build(in);
        session = factory.openSession();
        studentDao= session.getMapper(StudentDao.class);
    }

    @After
    public void destroy() throws IOException {
        session.commit();
        session.close();
        in.close();
    }


    @Test
    public void selectAll(){
        List<Student> all = studentDao.findAll();
        for (int i = 0; i < all.size(); i++) {
            Student student = all.get(i);
            System.out.println(student);
        }

    }

    @Test
    public void selectLikeByName(){
        List<Student> all = studentDao.findLikeByName("%三%");
        for (int i = 0; i < all.size(); i++) {
            Student student = all.get(i);
            System.out.println(student);
        }

    }


    @Test
    public void insert(){
        Student student = new Student();
```

```
        student.setName("张无忌");
        student.setAge(23);
        student.setSchool("武当");
        student.setTeacher_id(1);
        studentDao.insert(student);
    }

    @Test
    public void update(){
        Student student = new Student();
        student.setId(4);
        student.setName("王五");
        student.setAge(23);
        studentDao.update(student);
    }


    @Test
    public void delete(){
        studentDao.delete(4);
    }


    @Test
    public void queryById(){
        Student student = studentDao.selectById(6);
        System.out.println(student.toString());
    }
}
```

注意：

Mybatis允许同时使用注解和xml配置，但是不允许同一个方法即使用注解，又使用xml配置，会报错

```
### Cause: org.apache.ibatis.builder.BuilderException: Error parsing SQL Mapper Configuration.
Cause: java.lang.IllegalArgumentException: Mapped Statements collection already contains value for
com.wwxy.dao.StudentDao.selectById

    at org.apache.ibatis.exceptions.ExceptionFactory.wrapException(ExceptionFactory.java:30)
    at org.apache.ibatis.session.SqlSessionFactoryBuilder.build(SqlSessionFactoryBuilder.java:80)
    at org.apache.ibatis.session.SqlSessionFactoryBuilder.build(SqlSessionFactoryBuilder.java:64)
    at com.wwxy.test.TestAnnotation.init(TestAnnotation.java:32)
    at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
    at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
    at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
    at java.lang.reflect.Method.invoke(Method.java:498)
    at org.junit.runners.model.FrameworkMethod$1.runReflectiveCall(FrameworkMethod.java:45)
    at org.junit.internal.runners.model.ReflectiveCallable.run(ReflectiveCallable.java:15)
    at org.junit.runners.model.FrameworkMethod.invokeExplosively(FrameworkMethod.java:42)
    at org.junit.internal.runners.statements.RunBefores.evaluate(RunBefores.java:27)
    at org.junit.internal.runners.statements.RunAfters.evaluate(RunAfters.java:30)
    at org.junit.runners.ParentRunner.runLeaf(ParentRunner.java:263)
    at org.junit.runners.BlockJUnit4ClassRunner.runChild(BlockJUnit4ClassRunner.java:68)
    at org.junit.runners.BlockJUnit4ClassRunner.runChild(BlockJUnit4ClassRunner.java:47)
```

```
    at org.junit.runners.ParentRunner$3.run(ParentRunner.java:231)
    at org.junit.runners.ParentRunner$1.schedule(ParentRunner.java:60)
    at org.junit.runners.ParentRunner.runChildren(ParentRunner.java:229)
    at org.junit.runners.ParentRunner.access$000(ParentRunner.java:50)
    at org.junit.runners.ParentRunner$2.evaluate(ParentRunner.java:222)
    at org.junit.runners.ParentRunner.run(ParentRunner.java:300)
    at org.junit.runner.JUnitCore.run(JUnitCore.java:157)
    at com.intellij.junit4.JUnit4IdeaTestRunner.startRunnerWithArgs(JUnit4IdeaTestRunner.java:68)
    at
com.intellij.rt.execution.junit.IdeaTestRunner$Repeater.startRunnerWithArgs(IdeaTestRunner.java:47)
    at com.intellij.rt.execution.junit.JUnitStarter.prepareStreamsAndStart(JUnitStarter.java:242)
    at com.intellij.rt.execution.junit.JUnitStarter.main(JUnitStarter.java:70)
Caused by: org.apache.ibatis.builder.BuilderException: Error parsing SQL Mapper Configuration.
Cause: java.lang.IllegalArgumentException: Mapped Statements collection already contains value for
com.wwxy.dao.StudentDao.selectById
    at org.apache.ibatis.builder.xml.XMLConfigBuilder.parseConfiguration(XMLConfigBuilder.java:121)
    at org.apache.ibatis.builder.xml.XMLConfigBuilder.parse(XMLConfigBuilder.java:99)
    at org.apache.ibatis.session.SqlSessionFactoryBuilder.build(SqlSessionFactoryBuilder.java:78)
    ... 25 more
Caused by: java.lang.IllegalArgumentException: Mapped Statements collection already contains value
for com.wwxy.dao.StudentDao.selectById
    at org.apache.ibatis.session.Configuration$StrictMap.put(Configuration.java:872)
    at org.apache.ibatis.session.Configuration$StrictMap.put(Configuration.java:844)
    at org.apache.ibatis.session.Configuration.addMappedStatement(Configuration.java:668)
    at
org.apache.ibatis.builder.MapperBuilderAssistant.addMappedStatement(MapperBuilderAssistant.java:302)

    at
org.apache.ibatis.builder.annotation.MapperAnnotationBuilder.parseStatement(MapperAnnotationBuilder.java:351)

    at
org.apache.ibatis.builder.annotation.MapperAnnotationBuilder.parse(MapperAnnotationBuilder.java:134)

    at org.apache.ibatis.binding.MapperRegistry.addMapper(MapperRegistry.java:72)
    at org.apache.ibatis.binding.MapperRegistry.addMappers(MapperRegistry.java:97)
    at org.apache.ibatis.binding.MapperRegistry.addMappers(MapperRegistry.java:105)
    at org.apache.ibatis.session.Configuration.addMappers(Configuration.java:737)
    at org.apache.ibatis.builder.xml.XMLConfigBuilder.mapperElement(XMLConfigBuilder.java:364)
    at org.apache.ibatis.builder.xml.XMLConfigBuilder.parseConfiguration(XMLConfigBuilder.java:119)
    ... 27 more


java.lang.NullPointerException
    at com.wwxy.test.TestAnnotation.destroy(TestAnnotation.java:39)
    at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
    at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
    at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
    at java.lang.reflect.Method.invoke(Method.java:498)
    at org.junit.runners.model.FrameworkMethod$1.runReflectiveCall(FrameworkMethod.java:45)
    at org.junit.internal.runners.model.ReflectiveCallable.run(ReflectiveCallable.java:15)
    at org.junit.runners.model.FrameworkMethod.invokeExplosively(FrameworkMethod.java:42)
    at org.junit.internal.runners.statements.RunAfters.evaluate(RunAfters.java:36)
    at org.junit.runners.ParentRunner.runLeaf(ParentRunner.java:263)
```

```
    at org.junit.runners.BlockJUnit4ClassRunner.runChild(BlockJUnit4ClassRunner.java:68)
    at org.junit.runners.BlockJUnit4ClassRunner.runChild(BlockJUnit4ClassRunner.java:47)
    at org.junit.runners.ParentRunner$3.run(ParentRunner.java:231)
    at org.junit.runners.ParentRunner$1.schedule(ParentRunner.java:60)
    at org.junit.runners.ParentRunner.runChildren(ParentRunner.java:229)
    at org.junit.runners.ParentRunner.access$000(ParentRunner.java:50)
    at org.junit.runners.ParentRunner$2.evaluate(ParentRunner.java:222)
    at org.junit.runners.ParentRunner.run(ParentRunner.java:300)
    at org.junit.runner.JUnitCore.run(JUnitCore.java:157)
    at com.intellij.junit4.JUnit4IdeaTestRunner.startRunnerWithArgs(JUnit4IdeaTestRunner.java:68)
    at
com.intellij.rt.execution.junit.IdeaTestRunner$Repeater.startRunnerWithArgs(IdeaTestRunner.java:47)
    at com.intellij.rt.execution.junit.JUnitStarter.prepareStreamsAndStart(JUnitStarter.java:242)
    at com.intellij.rt.execution.junit.JUnitStarter.main(JUnitStarter.java:70)
```

StudentDao.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.wwxy.dao.StudentDao">

    <sql id="selectSql" >
        SELECT * from student
    </sql>

    <select id="selectById" parameterType="int" resultType="student">
        <include refid="selectSql"></include> where id =#{id}
    </select>


</mapper>
```

20.2 MyBatis中注解多表查询

20.2.1 当实体类字段和数据库字段不一致时，使用注解开发

使用@Results和@ResultMap注解来代替原来的标签和resultMap属性

修改Student对象的name属性的名称，使其和数据的字段名称不一致，

Student

```
package com.wwxy.pojo;

import java.io.Serializable;

public class Student implements Serializable{

    private Integer id;

    private String sname;
```

```
public String getSname() {
    return sname;
}

public void setSname(String sname) {
    this.sname = sname;
}

private int age;

private String school;

private int teacher_id;

public Integer getId() {
    return id;
}

public void setId(Integer id) {
    this.id = id;
}

public int getAge() {
    return age;
}

public void setAge(int age) {
    this.age = age;
}

public String getSchool() {
    return school;
}

public void setSchool(String school) {
    this.school = school;
}

public int getTeacher_id() {
    return teacher_id;
}

public void setTeacher_id(int teacher_id) {
    this.teacher_id = teacher_id;
}

@Override
public String toString() {
    return "Student{" +
            "id=" + id +
            ", sname='" + sname + '\'' +
            ", age=" + age +
```

```
                ", school='" + school + '\'' +
                ", teacher_id=" + teacher_id +
                '}';
    }
}
```

StudentDao

```
package com.wwxy.dao;

import com.wwxy.pojo.Student;
import org.apache.ibatis.annotations.*;

import java.util.List;

public interface StudentDao {

    @Select("select * from student")
    @Results(id = "studentResultMap",
            value = {
            @Result(column = "name",property = "sname")
            }
    )
    List<Student> findAll();


    @Select("select * from student where name like #{name}")
    @ResultMap(value = {"studentResultMap"})
    List<Student> findLikeByName(String name);


    @Insert("insert into student(name,age,school,teacher_id) values(#{name},#{age},#{school},#
{teacher_id})")
    void insert(Student student);

    @Update("update student set name=#{name},age=#{age} where id=#{id}")
    void update(Student student);


    @Delete("delete from student where id=#{id}")
    void delete(int id);

    @Select("select * from student where id=#{id}")
    @ResultMap(value = {"studentResultMap"})
    Student selectById(Integer id);



}
```

如上所示，其实@Results注解相当于标签。而@result注解相当于该标签里的标签和标签

而@ResultMap标签则相当于查询标签如 ▼