****

**设计模式实践大作业**



**学 院 国际工程师学院**

**专 业 计算机技术**

**年 级 2020 级**

**姓 名贾博森、梁怡炜、赵亚新、孙盼**

**2021年 6 月 11 日**

目 录

[一、 需求分析 1](#_Toc70789841)

[二、 项目设计 1](#_Toc70789842)

[三、 设计中体现的面向对象设计原则 1](#_Toc70789843)

[四、 设计中使用到的设计模式 1](#_Toc70789844)

# 需求分析

在线考试系统主要包括系统的注册、在线的考试系统、评分系统、试卷自由组装系统、针对不同人群考试系统、在线的管理系统和在线的阅卷系统等。  
 首先是注册功能，虽然在本系统中没有复杂的注册功能，也不需要用户的详细信息但是一定的注册功能对系统的安全还是必要的。  
 其次是在线的考试系统，这里要考虑的主要是怎样出题、怎样收集答案、怎样保存，怎样判分这些问题。  
 试卷自由组装系统，不同类型的试题可以自由组装形成一-套新的试卷。  
 针对不同人群考试系统，可以安排不同的人群考试。  
 在线的账号管理系统比较简单，就是要对数据库中的系统用户和考试用户进行管理，而管理的项目则包括浏览整个系统的用户，添加、删除用户，以及更改用户的密码。  
 在线的阅览试卷系统是对在线考试系统的补充，因为我们的在线系统是自动判分的，但是为了判分的准确性，以及对用户的答题情况进行详细地了解，对用户的试卷应该进行判定。  
**1.1目标设计**  
在线考试系统的功能如下。  
1.登录系统的功能如下:  
（1）欢迎页面功能选择页面

（2）各种功能的登录页面及密码验证

2.在线考试系统的主要实现功能要求:  
（1）让用户选择试卷类型  
（2）从试题库中随机出题给用户  
（3）将随机试卷保存在暂时表中  
（3）保存用户的答案  
（4）给用户的试卷判分  
（5）保存用户的试卷  
（6）查询用户成绩  
3.试卷自由组装系统  
（1）让用户自己选择试卷题目  
（2）系统自己组装试卷题目  
4.针对不同人群考试系统  
（1）安排不同人群考相同的试题  
5.在线账户管理系统的主要功能要求  
（1）选择更改账户属性的类别

（2）浏览整个系统中的各种用户(学生用户、教师用户、超级管理员)  
（3）添加包含某项功能的用户

（4）删除不再需要的用户  
（5）更改用户的密码  
6.在线阅卷系统的主要功能要求  
（1）浏览系统中用户答案情况。  
（2）删除某个用户的其中一个答卷或删除某种系统试卷。

**系统功能需求图：**

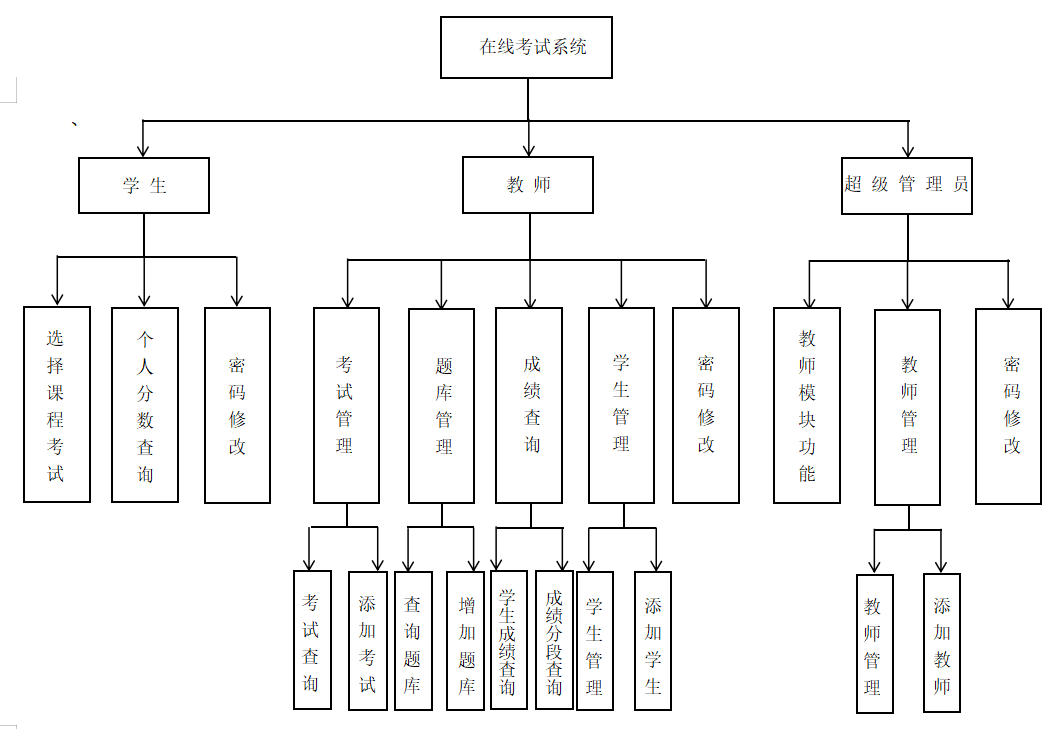


图1-1 系统功能需求

**1.2设计思想**

在线考试系统充分利用了Brower/Server结构的特点，实现了将考试系统移植到Internet的功能，该系统的优势在于极大的简化了老师出题的工作和学生考试的试题，做到了出题的多样性。  
 目前比较流行的两种网络开发模式分别是Client/Server模式与Brower/Server模式。  
 Client/Server模式主要是在以局域网为基础的环境下展开应用的，它受到地域的限制。而Brower/Server模式是通过Internet进行通信,可以不受地域的限制，但是它不能够进行联机事务处理，并县城在大量数据处理的情况下，速度较慢。从目前的开发技术来看，Brower端可以作为信息收集源，特别是大量的数据录入工作还不能完全取代Client端的用户界面。在这方面，Client端的各种开发工具的功能更加强大也更加灵活，而Browser端由于其功能结构的限定过于“瘦小”。  
 对于本在线考试系统，其功能目标是实现将现有的纸张答题式考试向基于Internet的无纸张考试系统的转变，所以它必须实现整个考试流程的进行，实现对系统的账户不同管理，以及完成阅卷、评分这些环节的处理。由于系统数据收集的实时性要求并不高，因此采用Browser/Server结构来进行设计是合理可行的。

**1.3系统功能设计**

根据系统功能的要求，在线考试系统分为上述的3个模块。即在线的考试系统，在线账户管理系统和在线的阅卷系统。可以将系统分解成几个功能模块来分别设计，功能模块之间关系如图1-2所示。

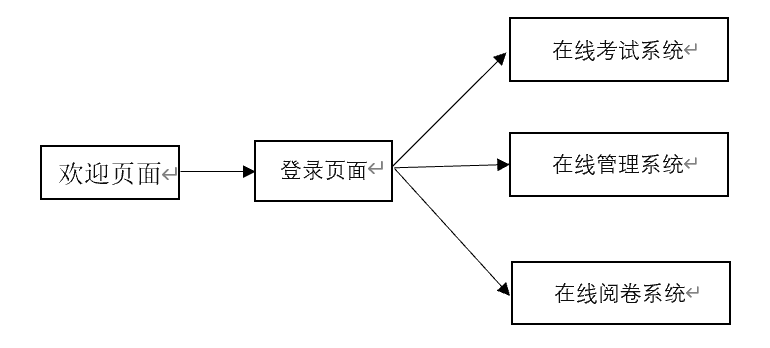


图1-2 系统功能模块

下面再来看一下整个系统的所有页面情况。

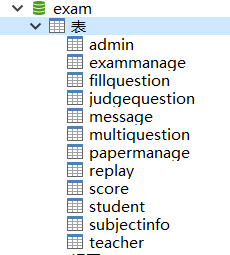
对于在线考试系统，根据系统功能分析,仅有一个单一的流程化页面。所以从学生登录页面开始，接着就进入考试页面，然后是查询成绩的页面。  
 对于在线的管理系统，根据系统功能分析，可以具体细化为浏览系统用户页面，添加用户页面以及删除用户页面。  
**1.4数据库设计**

根据系统功能设计的要求以及功能模块的划分，对于在线考试系统的数据库，可以列出以下数据项和数据结构。

（1）超级管理员表:管理员编号、姓名、密码、权限。

（2）学生信息表:主要包括学生的学号(主键)、姓名、民族、班级、年龄、专业、系别、密码、权限。  
（3）教师信息表:教师ID (主键)、姓名、性别、年龄、系别、专业、密码、权限。  
（4）课程表:课程编号(主键)、课程名称、教师ID、课程类别、备注。

（5）考试管理表:试卷ID(主键)、试卷名称、试卷分数、答题开始时间、答题时间、题型数目  
（6）试题信息表:包括填空题、判断题、选择题三个子表。分别设置专业、试题题型、难易程度、试题内容、试题分数、试题答案、教师编号。  
（7）成绩信息表:编号ID（主键)、学号、课程编号、总分、得分、是否及格(bool类型)。



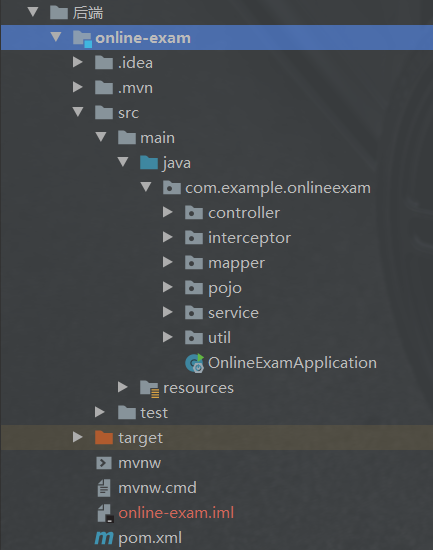
# 项目设计

项目整体采用前后端分离的架构，后端系统由SpringBoot框架继承Mybatis框架进行实现。

SpringBoot 是简化 Spring 应用开发的一个框架，他整合了 Spring 的技术栈，提供各种标准化的默认配置，使得我们可以快速开发 Spring 项目，免掉 xml 配置的麻烦，降低 Spring 项目的成本。

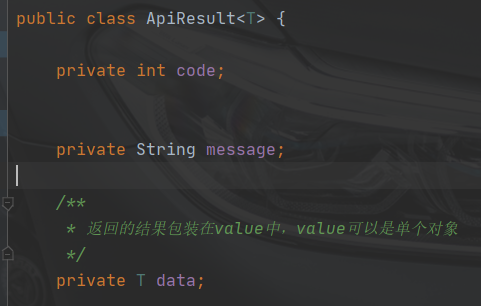
MyBatis是一个优秀的持久层框架，它对jdbc的操作数据库的过程进行封装，使开发者只需要关注SQL本身，而不需要花费精力去处理例如注册驱动、创建connection、创建statement、手动设置参数、结果集检索等jdbc繁杂的过程代码。Mybatis通过xml或注解的方式将要执行的各种statement（statement、preparedStatemnt）配置起来，并通过java对象和statement中的sql进行映射生成最终执行的sql语句，最后由mybatis框架执行sql并将结果映射成java对象并返回。

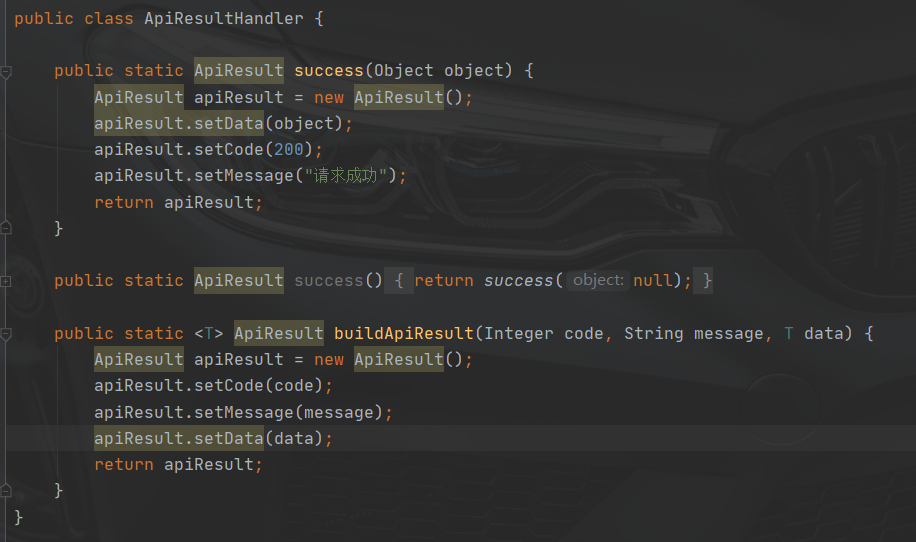
后端项目整体架构如图：

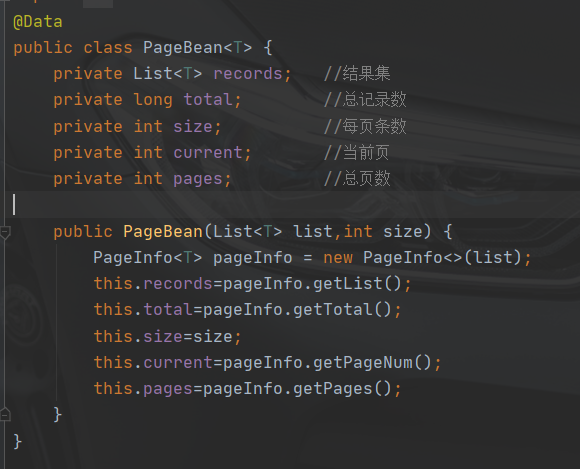


项目分为controller接口层，interceptor拦截器层，mapper映射层，pojo实体层，service业务层以及util工具层。其中，接口层负责编写接口，以实现与前端的数据交互，调用业务层方法；业务层处理业务逻辑，并调用映射层执行数据库操作；映射层中配置SQL语句，执行数据库操作，将结果封装为实体类并返回；拦截器层提供登录拦截功能；工具类则编写工具方法供其他层使用。

工具类中，我们首先定义了三个高可用类，分别是ApiResult、ApiResultHandle以及PageBean。ApiResult类来统一封装接口返回数据的类型，由状态码，信息以及数据三个字段组成；ApiResultHandle类负责生成ApiResult；PageBean负责封装分页对象。

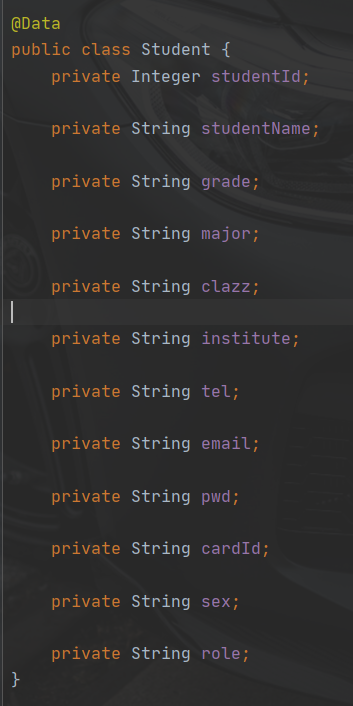






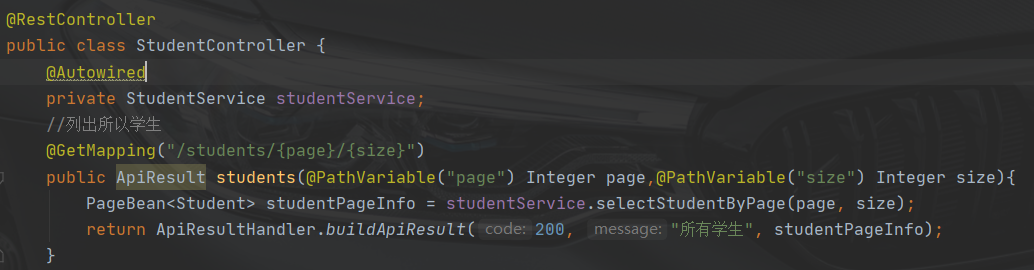
接下来介绍后端接口的具体实现，由于代码量较大，我们这里重点接收**学生相关接口**的实现。

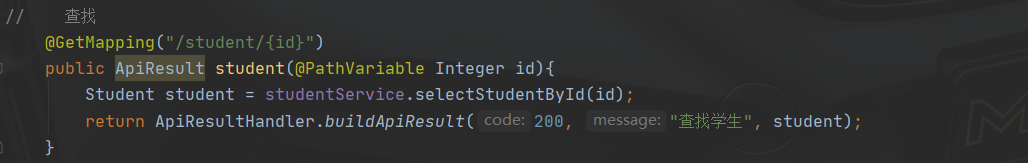
首先依据需求分析定义学生实体类并实现数据库建表。这里我们利用了lombok简化了实体类中getter和setter方法的声明。

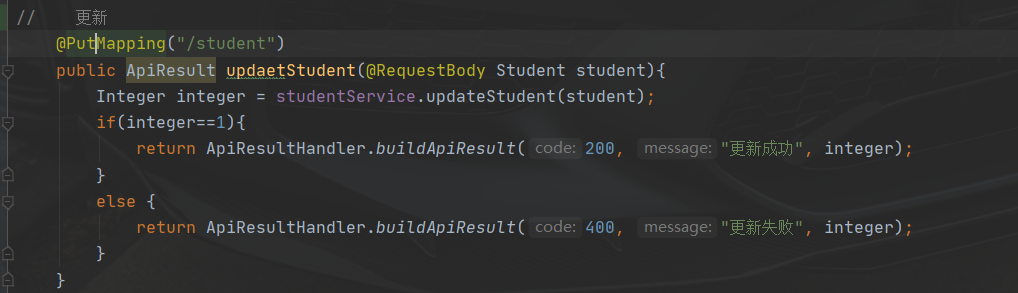


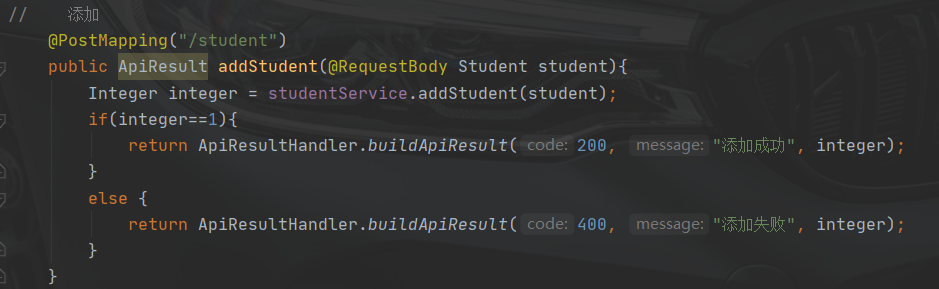
在学生接口类中，我们依次定义“查询所有学生”、“依据学号查找学生”，“更新学生信息”，“添加学生信息”，“删除学生信息”，“修改学生账号密码”以及“退出登录”这7个接口。

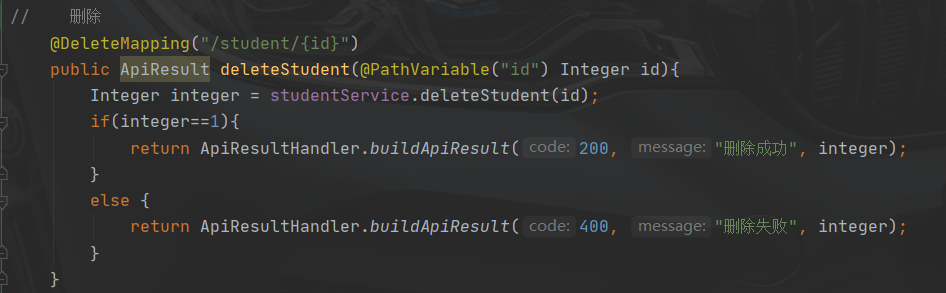
首先注入StudentService对象，供接口调用业务层方法。每个接口在调用方法完成“业务层传递-映射层数据库操作-业务层传递”的流程后返回结果后，统一将结果封装进高可用对象ApiResult返回，代码如下：

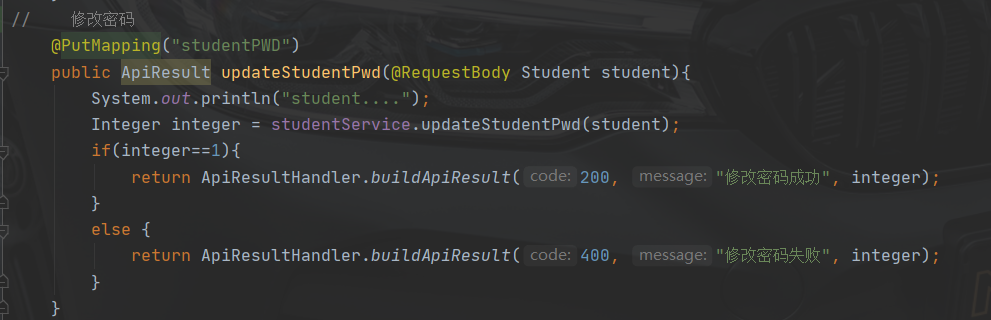


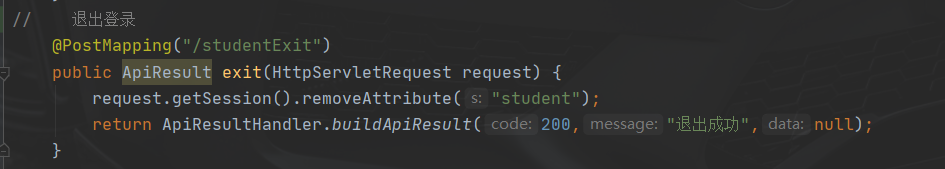




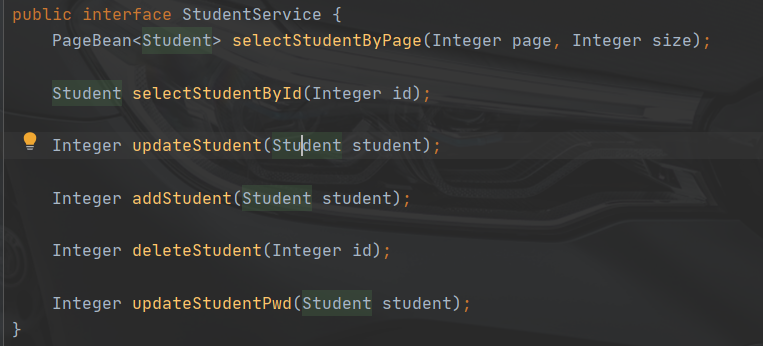


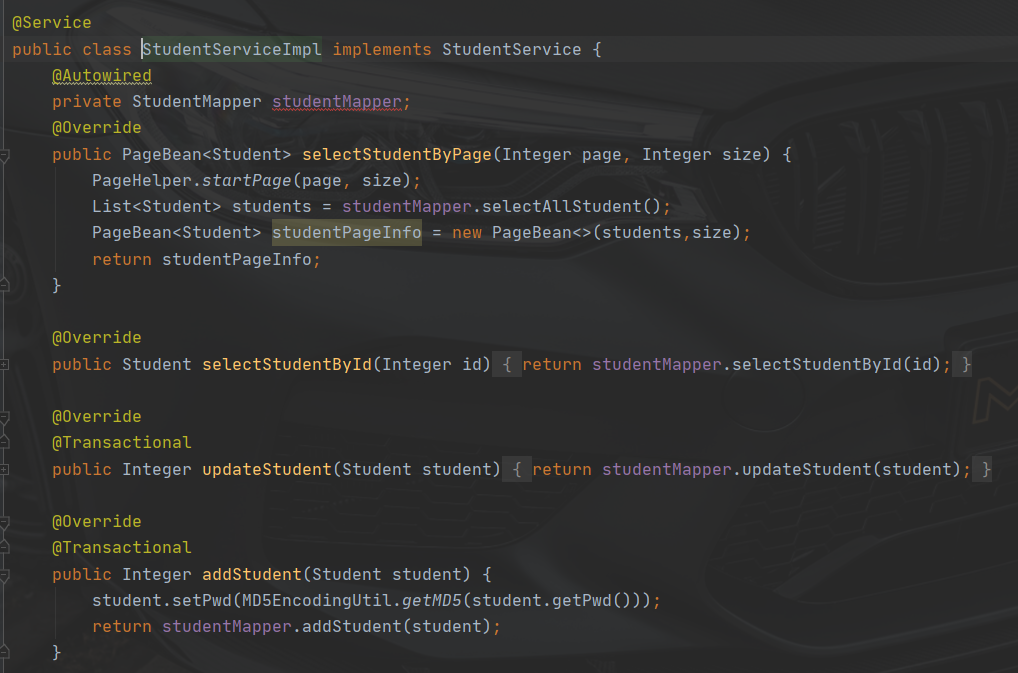


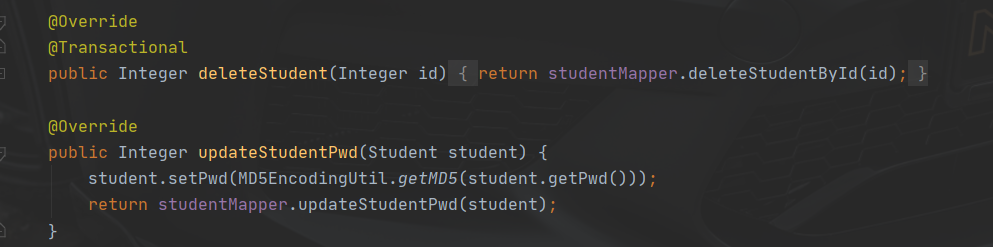




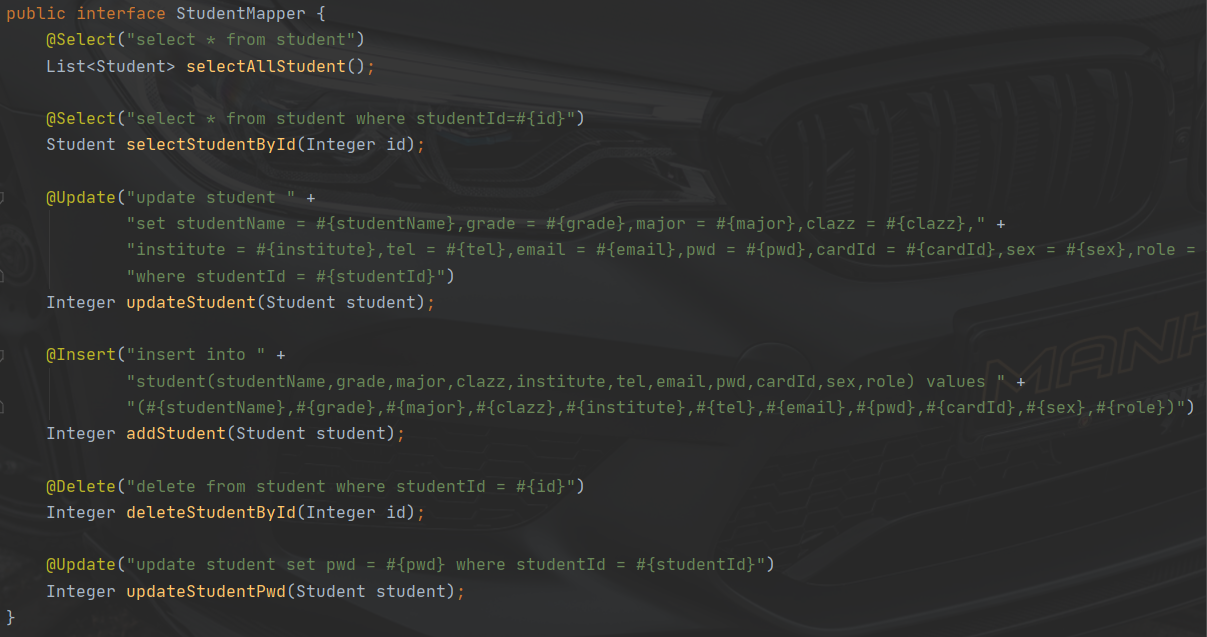
对于业务层，我们分别实现业务层接口及其实现，进一步降低耦合，在实现类中，通过注入StudentMapper对象供业务层方法调用映射层方法：







对于映射层，我们使用Mybatis框架基于注解的方式实现数据库操作：



# 三、设计中体现的面向对象设计原则

**3.1 依赖倒转原则**

高层模块不应该依赖低层模块，它们都应该依赖抽象。抽象不应该依赖于细节，细节应该依赖于抽象。

针对抽象层编程，将具体类的对象通过依赖注入(Dependency Injection, DI)的方式注入到其他对象。

在 Spring 框架中实现 IoC 容器的方法就是依赖注入，其作用是在使用 Spring 框架创建对象时动态地将其所依赖的底下注入 Bean 组件中，其目标是让调用者不要主动去使用被调用者，而是让被调用者向调用者提供服务。IoC 和 bean对象 的配合完美实现了这个过程，我们可以使用 @Component 、@Service、@Controller、@Repository 等注解添加一个 bean 到 IoC 容器，然后通过 @autowired 注解就可以让 IoC 容器自动找到对应的类注入进来。

**3.2 接口隔离原则**

客户端不应该依赖那些它不需要的接口。

Spring 的 IoC 容器不仅仅可以通过一个类的类型来实现依赖注入，也可以通过接口类型来实现依赖注入，我们只需要让接口更加专门化，然后将其放入 IoC 容器中，需要的时候使用 Spring 进行注入即可。

**3.3 迪米特法则**

每一个软件单位对其他的单位都只有最少的知识，而且局限于那些与本单位密切相关的软件单位

迪米特法则要求一个软件实体应当尽可能少地与其他实体发生相互作用。

从Spring IoC 的机制上来说，天然就降低了不同实体之间的依赖关系，因为我们不是通过 new 的方式来创建对象，而且通过依赖注入的方式，所以很充分地体现了迪米特法则。

**3.4 开闭原则**

软件实体应当对扩展开放，对修改关闭。

至于开闭原则那就更加明显了，我们基于 Spring 框架做开发，在构建类时大多数都是按照该原则 —— 新功能构建新的类来实现，而并非在原有的类的基础上添加新的方法。

**3.5 Mapper动态代理开发遵循四个原则**

1. 接口方法名 = User.xml中id名

2. 返回值类型与Mapper.xml文件中返回值类型一致

3. 方法入参类型与Mapper.xml中入参类型一致

4. 命名空间绑定此接口

**3.6 Mapper接口开发遵循以下规范**

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

# 四、设计中使用到的设计模式

**4.1 工厂方法**

实现方式：FactoryBean接口。

实现原理：实现了FactoryBean接口的bean是一类叫做factory的bean。其特点是spring会在使用getBean()调用获得该bean时，会自动调用该bean的getObject()方法，所以返回的不是factory这个bean，而是这个bean.getOjbect()方法的返回值。

在Mybatis中比如SqlSessionFactory使用的是工厂模式，该工厂没有那么复杂的逻辑，是一个简单工厂模式。

**4.2 单例模式**

Spring中依赖注入的Bean实例默认是单例的。

Spring的依赖注入（包括lazy-init方式）都是发生在AbstractBeanFactory的getBean里。getBean的doGetBean方法调用getSingleton进行bean的创建。

单例模式定义：保证一个类仅有一个实例，并提供一个访问它的全局访问点。

spring对单例的实现：spring中的单例模式完成了后半句话，即提供了全局的访问点BeanFactory。但没有从构造器级别去控制单例，这是因为spring管理的是任意的java对象。

在Mybatis中有两个地方用到单例模式，ErrorContext和LogFactory，其中ErrorContext是用在每个线程范围内的单例，用于记录该线程的执行环境错误信息。

**4.3 适配器模式**

实现方式：SpringMVC中的适配器HandlerAdatper。

实现原理：HandlerAdatper根据Handler规则执行不同的Handler。

实现过程：DispatcherServlet根据HandlerMapping返回的handler，向HandlerAdatper发起请求，处理Handler。

HandlerAdapter根据规则找到对应的Handler并让其执行，执行完毕后Handler会向HandlerAdapter返回一个ModelAndView，最后由HandlerAdapter向DispatchServelet返回一个ModelAndView。

实现意义：HandlerAdatper使得Handler的扩展变得容易，只需要增加一个新的Handler和一个对应的HandlerAdapter即可。

因此Spring定义了一个适配接口，使得每一种Controller有一种对应的适配器实现类，让适配器代替controller执行相应的方法。这样在扩展Controller时，只需要增加一个适配器类就完成了SpringMVC的扩展了。

Mybatis提供了多种日志框架的实现，这些实现都匹配这个Log接口所定义的接口方法，最终实现了所有外部日志框架到Mybatis日志包的适配。

**4.4 装饰器模式**

实现方式：Spring中用到的包装器模式在类名上有两种表现：一种是类名中含有Wrapper，另一种是类名中含有Decorator。

实质：动态地给一个对象添加一些额外的职责。就增加功能来说，Decorator模式比生成子类更为灵活。

在mybatis中，缓存的功能由根接口Cache（org.apache.ibatis.cache.Cache）定义。整个体系采用装饰器设计模式，数据存储和缓存的基本功能由PerpetualCache（org.apache.ibatis.cache.impl.PerpetualCache）永久缓存实现，然后通过一系列的装饰器来对PerpetualCache永久缓存进行缓存策略等方便的控制。

**4.5 代理模式**

实现方式：AOP底层，就是动态代理模式的实现。

动态代理：在内存中构建的，不需要手动编写代理类。

静态代理：需要手工编写代理类，代理类引用被代理对象。

实现原理：切面在应用运行的时刻被织入。一般情况下，在织入切面时，AOP容器会为目标对象动态的创建一个代理对象。SpringAOP就是以这种方式织入切面的。

织入：把切面应用到目标对象并创建新的代理对象的过程。

代理模式可以认为是Mybatis的核心使用的模式，正是由于这个模式，我们只需要编写Mapper.java接口，不需要实现，由Mybatis后台帮我们完成具体SQL的执行。

**4.6 观察者模式**

实现方式：spring的事件驱动模型使用的是 观察者模式 ，Spring中Observer模式常用的地方是listener的实现。

具体实现：事件机制的实现需要三个部分：事件源、事件、事件监听器。

ApplicationEvent抽象类(事件)

继承自jdk的EventObject，所有的事件都需要继承ApplicationEvent，并且通过构造器参数source得到事件源。该类的实现类ApplicationContextEvent表示ApplicaitonContext的容器事件。

ApplicationListener接口（事件监听器）

继承自jdk的EventListener，所有的监听器都要实现这个接口。

这个接口只有一个onApplicationEvent()方法，该方法接受一个ApplicationEvent或其子类对象作为参数，在方法体中，可以通过不同对Event类的判断来进行相应的处理。

当事件触发时所有的监听器都会收到消息。

ApplicationContext接口(事件源)

ApplicationContext是spring中的全局容器，翻译过来是”应用上下文”。

实现了ApplicationEventPublisher接口。

职责：负责读取bean的配置文档，管理bean的加载，维护bean之间的依赖关系，可以说是负责bean的整个生命周期，再通俗一点就是我们平时所说的IOC容器。

ApplicationEventMulticaster抽象类[事件源中publishEvent方法需要调用其方法getApplicationEventMulticaster]属于事件广播器，它的作用是把Applicationcontext发布的Event广播给所有的监听器。

**4.7 策略模式**

实现方式：Spring框架的资源访问Resource接口。该接口提供了更强的资源访问能力，Spring 框架本身大量使用了 Resource 接口来访问底层资源。

Resource 接口介绍：Resource接口是具体资源访问策略的抽象，也是所有资源访问类所实现的接口。

Resource 接口主要提供了如下几个方法：

getInputStream()：定位并打开资源，返回资源对应的输入流。每次调用都返回新的输入流。调用者必须负责关闭输入流。

exists()：返回 Resource 所指向的资源是否存在。

isOpen()：返回资源文件是否打开，如果资源文件不能多次读取，每次读取结束应该显式关闭，以防止资源泄漏。

getDescription()：返回资源的描述信息，通常用于资源处理出错时输出该信息，通常是全限定文件名或实际 URL。

getFile：返回资源对应的 File 对象（通常不用）。

getURL：返回资源对应的 URL 对象（通常不用）。

Resource 接口本身没有提供访问任何底层资源的实现逻辑，针对不同的底层资源，Spring 将会提供不同的 Resource 实现类，不同的实现类负责不同的资源访问逻辑。

Spring 为 Resource 接口提供了如下实现类：

UrlResource：访问网络资源的实现类。

ClassPathResource：访问类加载路径里资源的实现类。

FileSystemResource：访问文件系统里资源的实现类。

ServletContextResource：访问相对于 ServletContext 路径里的资源的实现类

InputStreamResource：访问输入流资源的实现类。

ByteArrayResource：访问字节数组资源的实现类。

这些 Resource 实现类，针对不同的的底层资源，提供了相应的资源访问逻辑，并提供便捷的包装，以利于客户端程序的资源访问。

**4.8 模版方法模式**

经典模板方法定义：父类定义了骨架（调用哪些方法及顺序），某些特定方法由子类实现。

最大的好处：代码复用。除了子类要实现的特定方法，其他方法及方法调用顺序都在父类中预先写好了。

所以父类模板方法中有两类方法：

1.共同的方法：所有子类都会用到的代码

2.不同的方法：子类要覆盖的方法，分为两种：

(1)抽象方法：父类中的是抽象方法，子类必须覆盖

(2)钩子方法：父类中是一个空方法，子类继承了默认也是空的

注：为什么叫钩子，子类可以通过这个钩子（方法），控制父类，因为这个钩子实际是父类的方法（空方法）！

Spring模板方法模式实质：是模板方法模式和回调模式的结合，是Template Method不需要继承的另一种实现方式。Spring几乎所有的外接扩展都采用这种模式。

具体实现：JDBC的抽象和对Hibernate的集成，都采用了一种理念或者处理方式，那就是模板方法模式与相应的Callback接口相结合。

采用模板方法模式是为了以一种统一而集中的方式来处理资源的获取和释放。

在Mybatis中，sqlSession的SQL执行，都是委托给Executor实现的。

**4.9 Builder模式**

Builder模式的定义是“将一个复杂对象的构建与它的表示分离，使得同样的构建过程可以创建不同的表示。”，它属于创建类模式。

一般来说，如果一个对象的构建比较复杂，超出了构造函数所能包含的范围，就可以使用工厂模式和Builder模式。

相对于工厂模式会产出一个完整的产品，Builder应用于更加复杂的对象的构建，甚至只会构建产品的一个部分。

在Mybatis环境的初始化过程中，SqlSessionFactoryBuilder会调用XMLConfigBuilder读取所有的MybatisMapConfig.xml和所有的\*Mapper.xml文件，构建Mybatis运行的核心对象Configuration对象，然后将该Configuration对象作为参数构建一个SqlSessionFactory对象。

其中XMLConfigBuilder在构建Configuration对象时，也会调用XMLMapperBuilder用于读取\*Mapper文件，而XMLMapperBuilder会使用XMLStatementBuilder来读取和build所有的SQL语句。

在这个过程中，有一个相似的特点，就是这些Builder会读取文件或者配置，然后做大量的XpathParser解析、配置或语法的解析、反射生成对象、存入结果缓存等步骤，这么多的工作都不是一个构造函数所能包括的，因此大量采用了Builder模式来解决。

**4.10 组合模式**

组合模式组合多个对象形成树形结构以表示“整体-部分”的结构层次。

组合模式对单个对象(叶子对象)和组合对象(组合对象)具有一致性，它将对象组织到树结构中，可以用来描述整体与部分的关系。

同时它也模糊了简单元素(叶子对象)和复杂元素(容器对象)的概念，使得客户能够像处理简单元素一样来处理复杂元素，从而使客户程序能够与复杂元素的内部结构解耦。

在使用组合模式中需要注意一点也是组合模式最关键的地方：叶子对象和组合对象实现相同的接口。这就是组合模式能够将叶子节点和对象节点进行一致处理的原因。

Mybatis支持动态SQL的强大功能。