****

**Web编程实践大作业**



**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系统设计**  
**1.1.1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。

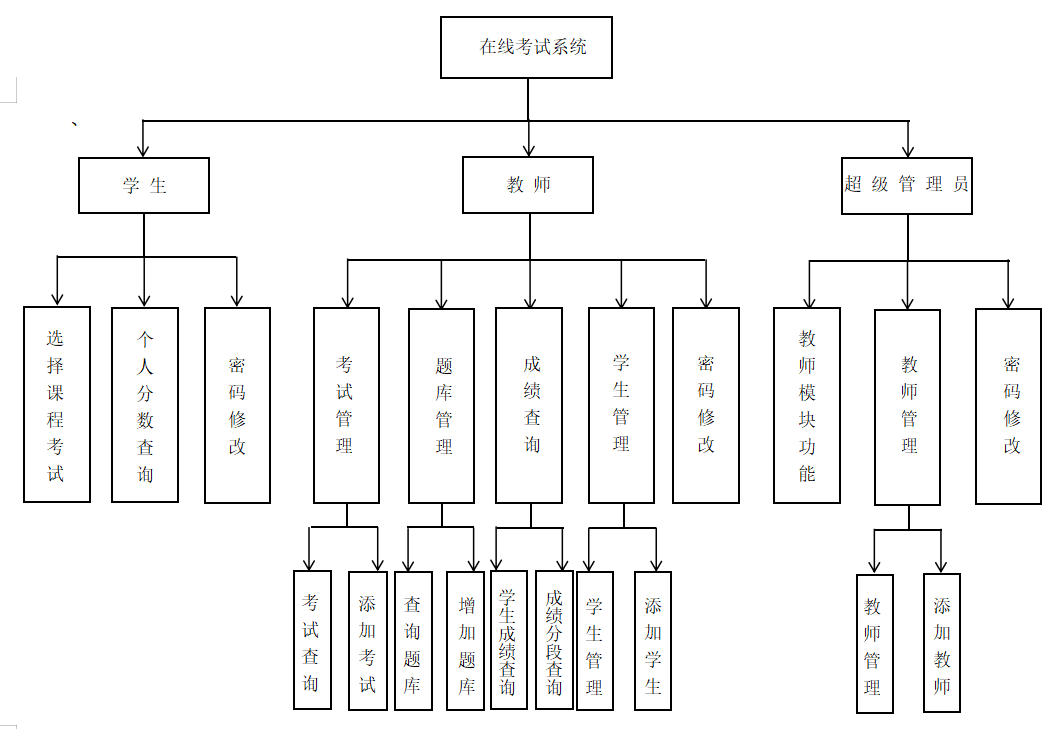


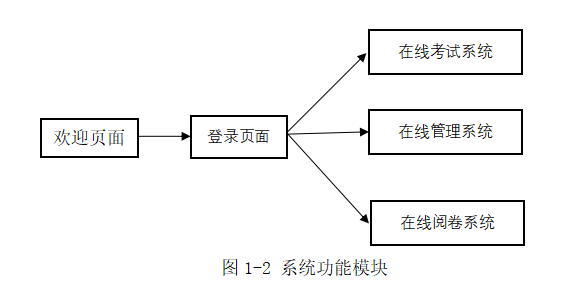
图1-1 系统功能需求图

**1.1.2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。

**1.1.3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。



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。

**1.2数据库设计与实现**  
**1.2.1数据库的需求分析**  
 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。

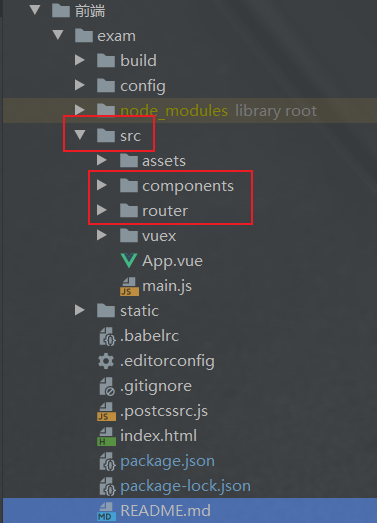
**超级管理员表**:管理员编号、姓名、密码、权限。

**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。  
**教师信息表**:教师ID (主键)、姓名、性别、年龄、系别、专业、密码、权限。  
**课程表**:课程编号(主键)、课程名称、教师ID、课程类别、备注。

**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  
**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。  
**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

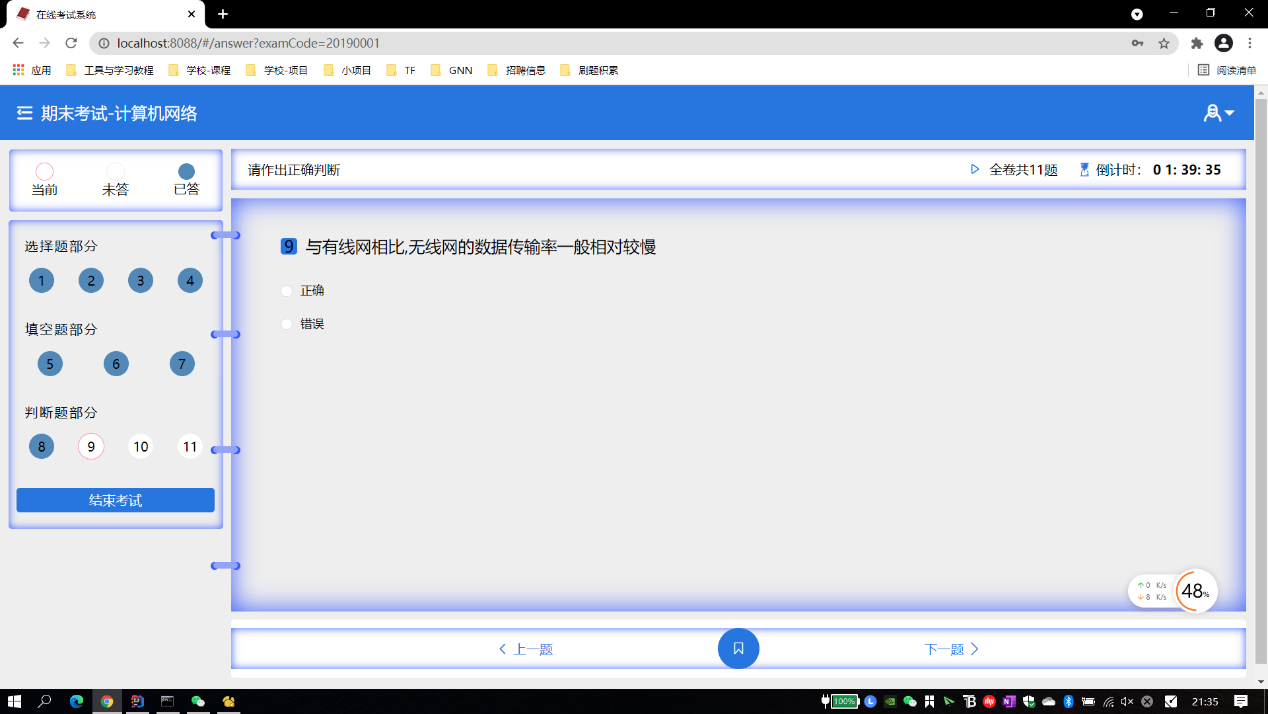
项目整体采用前后端分离的架构，前端系统采用Vue框架配合axios实现对后端接口的调用以及渲染。Vue 是一个采用 MVVM 架构, 通过数据驱动视图的形式来构建用户界面的渐进式框架，axios 是一个基于 promise 的 HTTP 库，可以用在浏览器和 node.js 中，与Vue配合可以代替Ajax实现前后端的交互，即接口的调用以及返回数据的获取。前端系统代码结构如图：



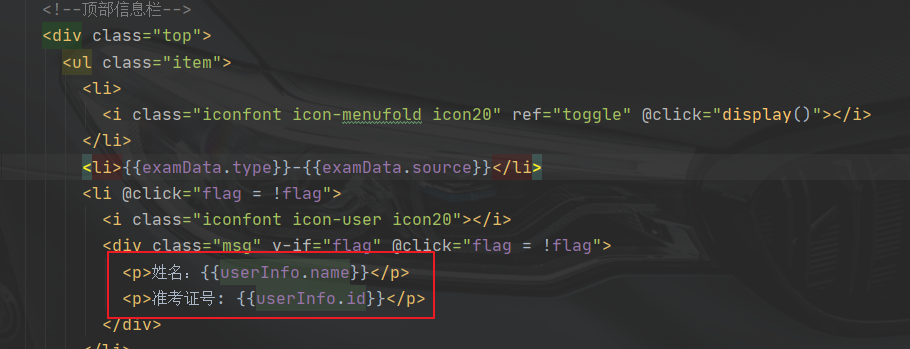
Src为项目的根目录，其中assert放置静态资源，包括公共的 css 文件、js文件、iconfont 字体文件、img 图片文件以及其他资源类文件；Components指项目开发的Vue组件；router放置路由设置文件，指定路由对应的组件；App.vue为Vue项目的根组件；main.js为Vue项目入口文件。

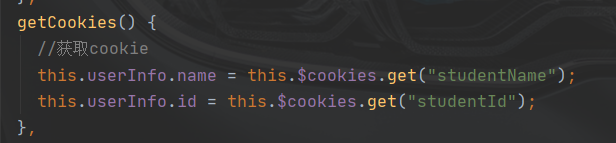
项目前端代码主要放置于components中，分为admin（超级管理员页面），chart（图表页面），common（通用页面），student（学生页面），teacher（教师页面），由于代码量较大，这里分别重点展示**学生考试界面**以及**教师考试管理页面**的前端具体实现。

1、学生考试页面：

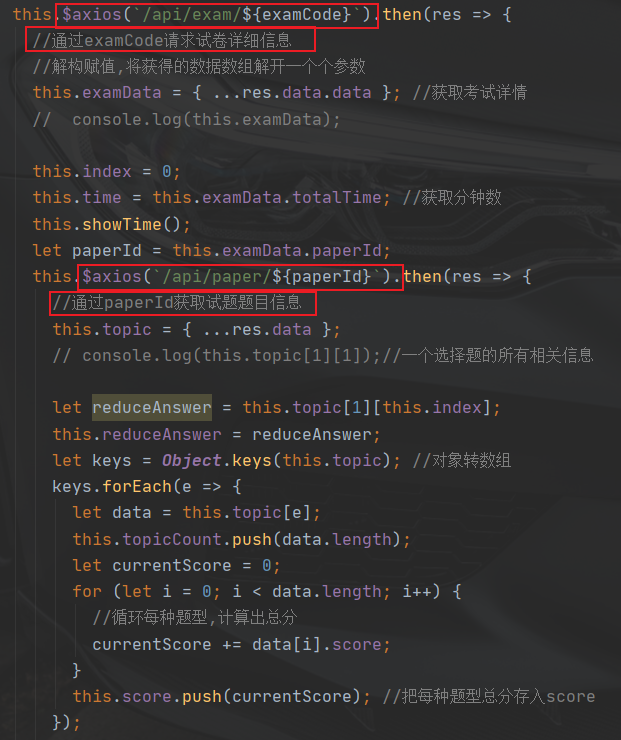


1. 页面右上方的考试信息栏。我们利用cookie获取：





1. 获取试卷信息：

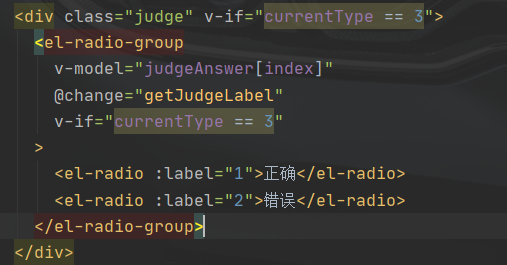


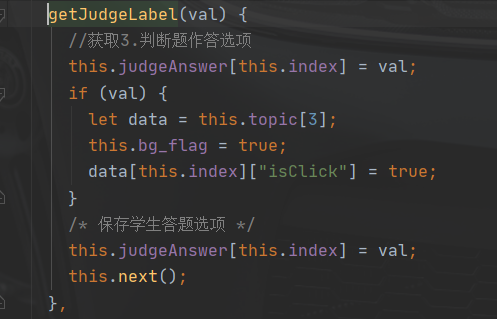
1. 左侧题目编号区。以判断题部分为例，获取试卷信息中选择题数组topic[3]后，遍历并显示图标，定位当前做题并判断是否已做，显示标记。





1. 右侧答题区。以判断题为例：





1. 各类型题目记分。以判断题为例：



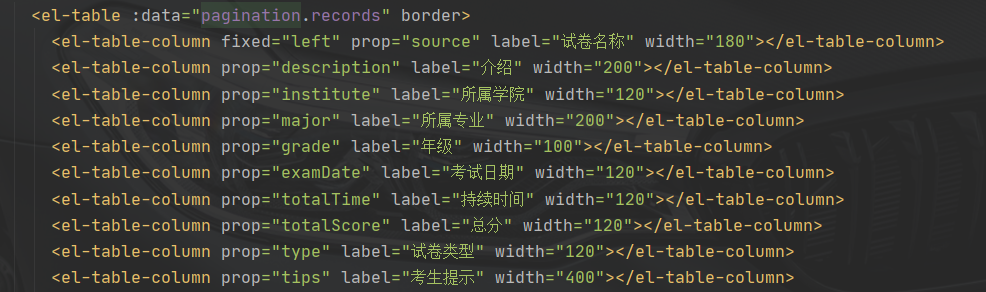
1. 计算总分并提交后端：

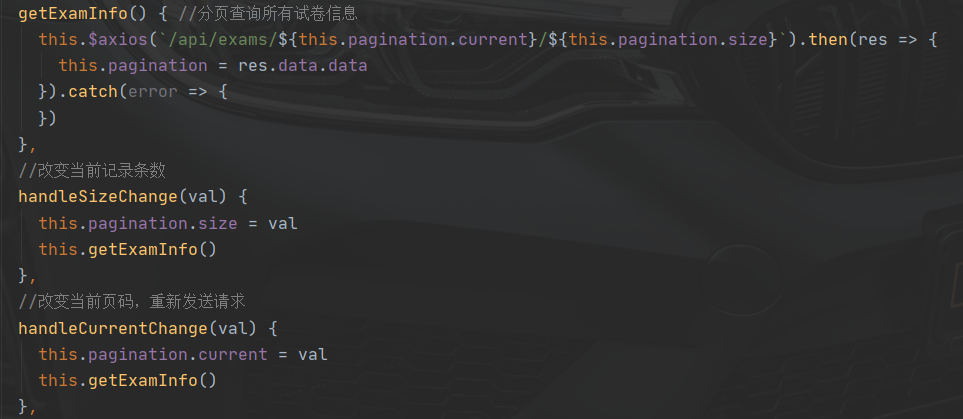


2、教师考试管理页面

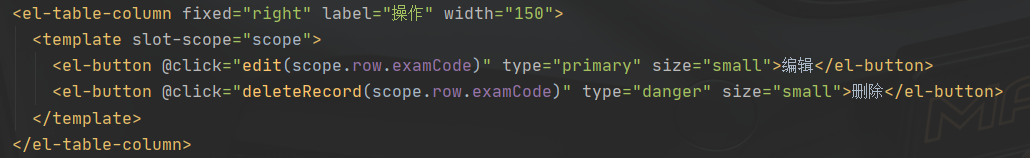


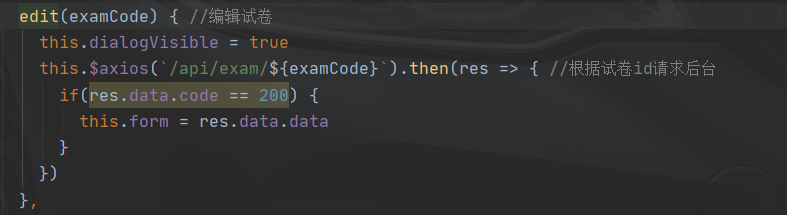
1. 考试信息分页展示，通过调取后端“查询所有考试信息”接口，将返回数据封装进pagination对象。

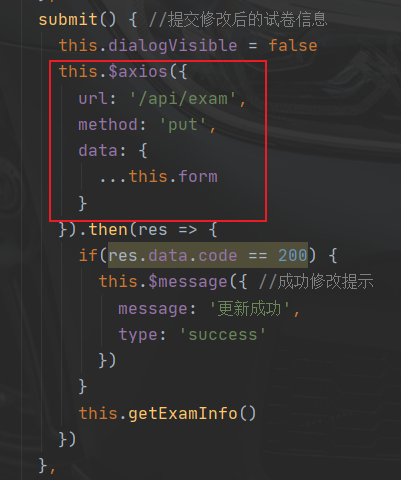


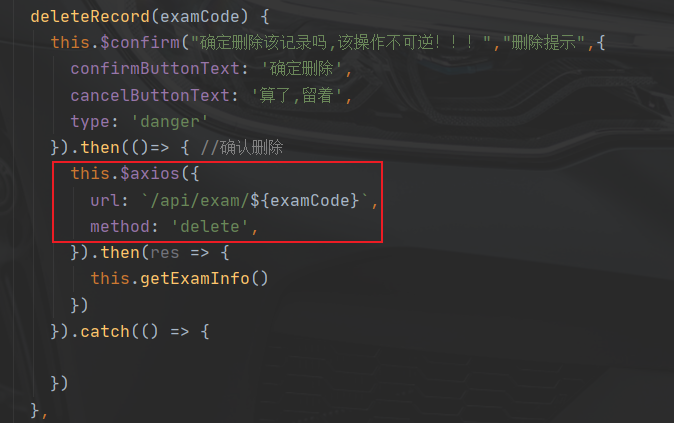


1. 编辑与删除操作。调用edit与deleteRecord方法时传入examCode









# 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。

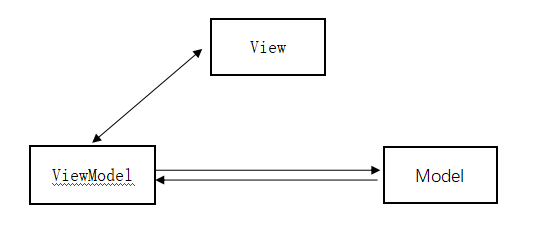
3. Mapper接口方法的输入参数类型和mapper.xml中定义的每个sql 的parameterType的类型相同。

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 。

# 设计中使用的设计模式

**MVVM设计模式**

1. 定义：MMVM模式是Model-View-ViewMode模式的简称。由视图(View)、视图模型(ViewModel)、模型(Model)三部分组成，结构如下图。通过这三部分实现UI逻辑、呈现逻辑和状态控制、数据与业务逻辑的分离。



使用MVVM模式有几大好处：

（1）低耦合。View可以独立于Model变化和修改，一个ViewModel可以绑定到不同的View上，当View变化的时候Model可以不变，当Model变化的时候View也可以不变。

（2）可重用性。可以把一些视图的逻辑放在ViewModel里面，让很多View重用这段视图逻辑。

（3）独立开发。开发人员可以专注与业务逻辑和数据的开发(ViewModel)。设计人员可以专注于界面(View)的设计。

（4）可测试性。可以针对ViewModel来对界面(View)进行测试

使用 MVVM 模式，程序的 UI 和其背后的展现与业务逻辑将被分离至三个类中：

（1）视图，封装 UI 与 UI 逻辑

（2）模型视图，封装展示逻辑与状态

（3）模型，封装程序的业务逻辑以及数据

在 MVVM 模式中，视图通过数据绑定以及命令行与视图模型交互，并改变事件通知。视图模型查询观察并协调模型更新，转换，校验以及聚合数据，从而在视图显示。

二、模型、视图、视图模型的分工

1.视图(View)

视图负责界面和显示。它通过DataContext(数据上下文)和ViewModel进行数据绑定，不直接与Model交互。 可以绑定Behavior/Comand来调用ViewModel的方法，Command是View到ViewModel的单向通行，通过实现Silverlight提供的IComand接口来实现绑定，让View触发事件，ViewModel来处理事件，以解决事件绑定功能。

2.视图模型(ViewModel)

视图模型主要包括界面逻辑和模型数据封装，Behavior/Command事件响应处理，绑定属性定义和集合等。它是View和Model的桥梁，是对Model的抽象，比如：Model中数据格式是“年月日”，可以在ViewModel中转换Model的数据为“日月年”供View显示。

实现视图模型需要实现Silverlight提供的接口INotifyPropertyChanged， INotifyPropertyChanged接口用于实现属性和集合的变更通知(Change Notifications)。使得在用户在视图上所做的操作都可以实时通知到视图模型，从而让视图模型对象有的模型进行正确的业务操作。

View的代码隐藏(Code-Behind)部分可能包含界面逻辑或者应用逻辑的代码，这些代码会很难进行单元测试，应根据具体情况尽量避免。

Mediator模式：

对不同View、ViewModel之间的参数传递，一般通过事件实现数据传递，也可以定义全局静态变量来进行数据共享，在MVVM里，我们可以使用Mediator模式(中介者模式)进行跟优雅的处理。可以简单得理解为不同的ViewModel或者View需要进行了操作之后，需要其他ViewModel或者View进行相应的其他操作或者数据更新，而中介类可以准确得将更新准确得通知到正确的ViewModel，同时中介者并不关心ViewModel或者View的任何操作或者数据更新。

3.模型(Model)

Model与MVC模式一样，Model用于封装与应用程序的业务逻辑相关的数据以及对数据的处理方法。它具有对数据直接访问的权利，例如对数据库的访问，Model不依赖于View和ViewModel，也就是说，模型不关心会被如何显示或是如何被操作，模型也不能包含任何用户使用的与界面相关的逻辑。Model在实际开发中根据实际情况可以进行细分。比如在广州市城乡规划资源平台就将Model将Service和Reposiroty结合为WCF服务由ViewModel进行调用。