医院网上预约系统

设计模式报告

科室、名医和病情介绍模块

作者信息：

张静圳 3190101085
石培材 3190105556
俞郭遥 3190103239
刘亚川 3190105123
丁铖琰 3190103116
陈振宇 3170100137

指导老师：

刘玉生

目录

1 设计模式调查

1.1 概述

在软件设计中，合适的设计模式可以帮助软件设计者更好地进行软件设计，使得设计变得规范。

设计模式包含很多方面，比如在一类问题中的常见错误，针对这类问题的解决方案等，并且它可以使得设计者在设计时能够避免一些将会引起问题的紧耦合，增强软件设计的适应变化的能力。

设计模式通常分为创建型、结构型与行为型三种模式大类。下文也将详述我们小组成员针对其中几种设计模式进行调查后进行的分析与总结。

设计模式为了达到高内聚、低耦合的目的，一般会采取以下基本原则：

1. 开放封闭原则（OCP, Open For Extension, Closed For Modification Principle），类的改动是通过增加代码进行的，而不是修改源代码。

2. 单一职责原则（SRP, Single Responsibility Principle），类的职责要单一，对外只提供一种功能，而引起类变化的原因都应该只有一个。

3. 依赖倒置原则（DIP, Dependence Inversion Principle），依赖于抽象（接口），不要依赖具体的实现（类），也就是针对接口编程。

4. 接口隔离原则（ISP, Interface Segregation Principle），不应该强迫客户的程序依赖它们不需要的接口方法。一个接口应该只提供一种对外功能，不应该把所有操作都封装到一个接口中去。

5. 里氏替换原则（LSP, Liskov Substitution Principle），任何抽象类出现的地方都可以用它的实现类进行替换。实际就是虚拟机制，语言级别实现面向对象功能。

6. 优先使用组合而不是继承原则（CARP, Composite/Aggregate Reuse Principle），如果使用继承，会导致父类的任何变换都可能影响到子类的行为。 如果使用对象组合，就降低了这种依赖关系。

7. 迪米特法则（LOD, Law of Demeter），一个对象应当对其他对象尽可能少地了解，从而降低各个对象之间的耦合，提高系统的可维护性。例如在一个程序中，各个模块之间相互调用时，通常会提供一个统一的接口来实现。这样其他模块不需要了解另外一个模块的内部实现细节，这样当一个模块内部的实现发生改变时，不会影响其他模块的使用。（黑盒原理）

1.2 设计模式调查与分析

1.2.1 创建型设计模式

创建型设计模式中我们调查了简单工厂设计模式。

(1)简单工厂模式

简单工厂模式（Simple Factory Pattern）：定义一个工厂类，它可以根据参数的不同，返回不同类的实例，被创建的实例通常都具有共同的父类。

简单工厂模式中包含的角色如下：

1. Factory（工厂）：核心部分，负责实现创建所有产品的内部逻辑，工厂类可以被外界直接调用，创建所需对象；

2. Product（抽象类产品）：工厂类所创建的所有对象的父类，封装了产品对象的公共方法，所有的具体产品为其子类对象；

3. Concrete Product（具体产品）：简单工厂模式的创建目标，所有被创建的对象都是某个具体类的实例。它要实现抽象产品中声明的抽象方法（有关抽象类）。

分析：在简单工厂模式中用于被创建实例的方法通常为静态（Static）方法，因此简单工厂模式又被成为静态工厂方法（Static Factory Method）。需要什么，只需要传入一个正确的参数，就可以获取所需要的对象，而无需知道其实现过程。

1.2.2 行为型设计模式

(1)命令模式

命令模式（Command Pattern）：该模式将一个请求封装成一个对象，从而能够使用不同的请求把客户端参数化，同时对请求排队或者记录请求日志，并提供命令的撤销和恢复功能。

命令模式中包含的角色及其相应的职责如下：

1. Command（命令）：声明执行操作的接口，由接口或者抽象类来实现；

2. Concrete Command（具体命令）：将一个接收者对象绑定于一个动作；调用接收者相应的操作，以实现命令角色声明的执行操作的接口；

3. Client（客户）：创建一个具体命令对象（并可以设定它的接收者）；

4. Invoker（请求者）：调用命令对象执行这个请求；

5. Receiver（接收者）：知道如何实施与执行一个请求相关的操作，任何类都可能作为一个接收者。

分析：在命令模式中，我们不仅仅将命令直接封装起来提供调用，而是加入了调用者和接受者两个角色，使得一条命令将分步完成，降低耦合度，提高灵活性。并且将命令进行多层逻辑封装，可以重用底层封装，且确保了一定的扩展性，对于客户端来说不需要知道复杂的逻辑，也很便捷。

(2)责任链模式

责任链模式（Chain of Responsibility Pattern）：为了避免请求发送者与多个请求处理者耦合在一起，将所有请求的处理者通过前一对象记住其下一个对象的引用而连成一条链；当有请求发生时，可将请求沿着这条链传递，直到有对象处理它为止。

责任链模式中包含的角色及其相应的职责如下：

1. Handler（抽象处理者）：定义出一个处理请求的接口。如果需要，接口可以定义出一个方法以设定和返回对下家的引用。这个角色通常由一个抽象类或者接口实现。

2. Concrete Handler（具体处理者）：具体处理者接到请求后，可以选择将请求处理掉，或者将请求传给下家。由于具体处理者持有对下家的引用，因此，如果需要，具体处理者可以访问下家。

分析：很多对象由每一个对象对其下家的引用而连接起来形成一条链。请求在这个链上传递，直到链上的某一个对象决定处理此请求。发出这个请求的客户端并不知道链上的哪一个对象最终处理这个请求，这使得系统可以在不影响客户端的情况下动态地重新组织和分配责任。

(3)模板方法模式

模板方法模式（Template Method Pattern）：定义一个操作中的算法骨架，而将算法的一些步骤延迟到子类中，使得子类可以不改变该算法结构的情况下重定义该算法的某些特定步骤。

模板方法模式中包含的角色及其相应的职责如下：

1. Abstract Class（抽象类）：用来定义算法骨架和原语操作，在这个类里面，还可以提供算法中通用的实现。

2. Concrete Class（具体实现类）：用来实现算法骨架中的某些步骤，完成跟特定子类相关的功能。

分析：通过把不变行为搬移到超类，去除子类中的反复代码来体现它的优势。模板方法模式就是提供了一个非常好的代码复用平台，当不变的和可变的行为在方法的子类实现中混

合在一起的时候，不变的行为就会在子类中反复出现。我们通过模板方法模式把这些行为搬移到单一的地方。这样就帮助子类摆脱反复的不变行为的纠缠。

(4)观察者模式

观察者模式（Observer Pattern）：该模式定义了一种一对多的依赖关系，让多个观察者对象同时监听某一个主题对象。这个主题对象在状态上发生变化时，会通知所有观察者对象，使它们能够自动更新自己。

观察者模式中包含的角色及其相应的职责如下：

1. Subject（抽象主题）：抽象主题角色把所有对观察者对象的引用保存在一个集合（比如 Vector）里，然后每个主题都可以有任何数量的观察者。抽象主题提供的接口有：增加观察者对象、删除观察者对象、通知所有的观察者对象、获取观察者数量、设置改变。

2. Concrete Subject（具体主题）：将有关状态存入具体主题对象，在具体主题的内部状态改变时，给所有注册过的观察者发送通知。

3. Observer（抽象观察者）：为所有的具体观察者定义一个接口，订阅的主题改变时更新自己，这个接口叫做更新接口。

4. Concrete Observer（具体观察者）：存储与主题自恰的状态。具体观察者角色实现抽象观察者角色所要求的更新接口，以便使本身的状态与主题的状态相协调。如有需要，具体观察者角色可以保持一个指向具体主题对象的引用。

分析：一个软件系统中常常要求在某一个对象的状态发生变化的时候，某些和它关联的其他的对象做出相应的改变。做到这一点的设计方案有很多，但是为了使系统能够易于复用，应该选择低耦合的设计方案。减少对象之间的耦合有利于系统的复用，但是同时设计时需要使这些低耦合的对象之间能够维持行为的协调一致，保证高度的协作。观察者模式是满足这一要求的各种设计方案中最重要的一种。

1.3 总结

对所分析的 5 种设计方法的优缺点及应用的总结：

简单工厂模式

优点	缺点
1. 良好的封装性 2. 解耦框架 3. 扩展性	应用时调用者清楚应该使用那个具体工厂的服务

命令模式

优点	缺点
1. 比较容易设计一个命令队列 2. 允许接受请求的一方是否处理请求 3. 可以容易地实现对请求的添加和删除	1. 可能会导致系统有过多的具体命令类 2. 应用时需要支持命令的撤销和恢复操作

责任链模式

优点	缺点
1. 将请求的发送者和接收者解耦 2. 可以简化你的对象，不需要知道链的结构 3. 通过改变链内的成员或调动他们的次序，允许你动态地新增或删除责任	1. 不能保证请求一定会被执行 2. 可能不容观察运行时的特征

当算法牵涉到一种链型运算，而且不希望在过程中有过多的循环和条件选择语句，并且希望比较容易地扩充文法，可以选择责任链模式。

模板方法模式

把不变的行为搬移到超类，可以通过模板方法可以把不变和可变的混合行为搬到单一地方，摆脱纠缠。

观察者模式

优点	缺点
1. 观察者和被观察者可以独立改变 2. 松耦合导致代码关系不明显	1. 广播时会有效率问题

对一个对象状态的更新，需要对其他对象同步更新，而且其他对象的数量动态可变时可用观察者模式。

2 科室、名医和病情介绍模块体系结构分析

2.1 系统关键质量

依据本子系统的软件需求分析，科室、名医和病情介绍模块的关键质量属性为性能、安全性和服务获得性。其中在性能方面，操作响应时间有明确的下界需求：登录和栏目页面跳转时间、更新处理时间在 3s 以内，具体信息展示页面跳转时间和用户提交预约时间在 2s 内。

2.2 系统体系结构

客户端：浏览网页主要采用 IE7.0 以上浏览器或者 Chrome。使用 React 库中的视图组件，简化客户端的设计，增加客户端的美观性。

JavaScript 脚本：客户端静态页面中，各种文本框与按键的操作均能触发脚本函数，脚本通过 fetch（），与服务器进行异步交互。并且对象接受服务器反馈信息后，能通过脚本函数对客户端静态页面实现重新渲染。

JavaScript 访问/修改数据库：此模块通过 URL 参数方式从客户端得到数据与命令，并且对其进行安全检测，而后按要求访问/修改数据库，并且返回操作结果或是查询结果。

2.3 在关键质量属性中选择的策略

性能：

并发：毫无疑问，引入并发是提高性能必不可少的策略。

减少开销：减少开销包括减少计算开销和减少通信开销两方面。

应用：一些格式匹配的工作可以在浏览器端直接进行验证，而无须在服务器端进行，以减少网络通信所带来的开销。

使用性：

取消：取消策略几乎应用在了所有的体系结构设计中。

服务获得性：

概述：

衡量指标：a = MTBF /（MTBF + MTTR）

MTBF：Mean time between failure，即平均故障时间。

MTTR：Mean time to recover，即平均恢复时间。

策略：

异常检测：异常检测包括操作系统异常、常数异常等情况。

状态同步：在被动冗余的情况下，状态的同步操作一般由主服务器定期更新备份服务器的数据来完成。

3 科室、名医和病情介绍模块设计模式分析

本模块主要涉及科室、名医和病情信息的详情展示以供查询和参考。

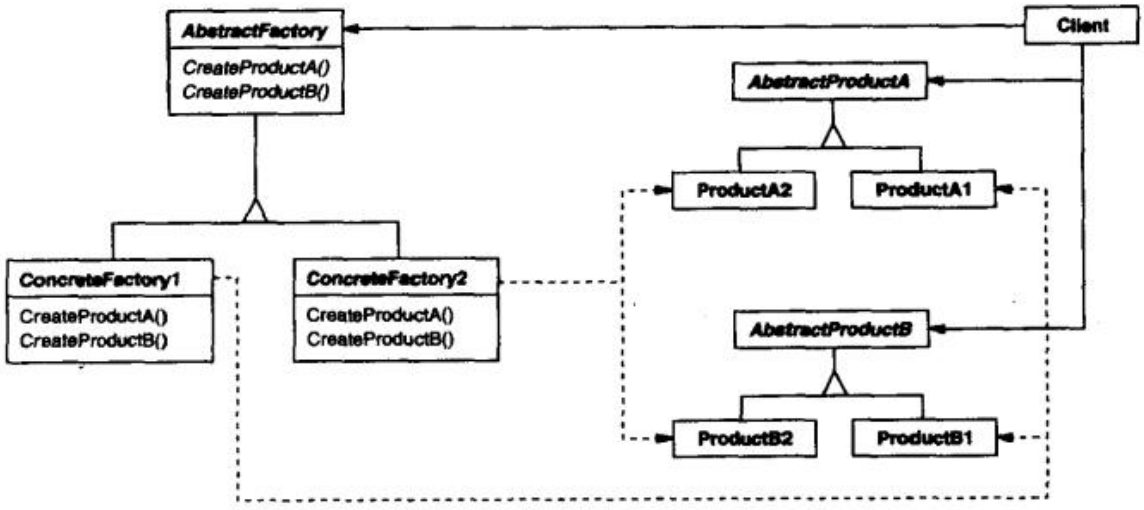
对于 Web 网页的布局，我们主要采用抽象工厂模式（Abstract Factory）；

对于和用户的交互，我们主要采用命令模式（Command）；

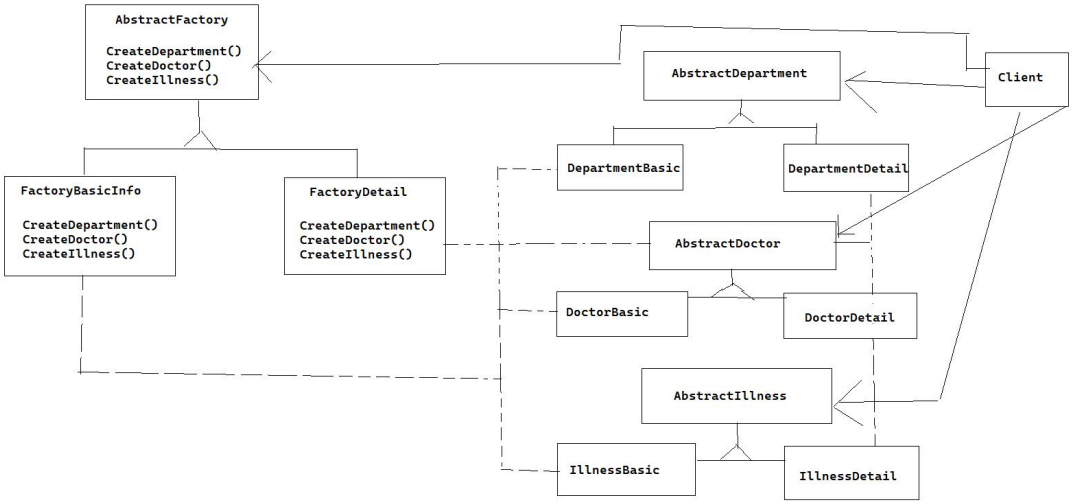
对于信息的解析与获取，我们主要采用职责链模式（Chain of Responsibility）。

3.1 抽象工厂模式

抽象工厂模式是一种试图提供一个创建一系列相关或相互依赖对象的接口，而无需指定它们具体的类的对象创建型设计模式。在我们的使用场景中，科室、名医和病情介绍子模块中，我们的系统由多个产品系列中的一个来配置，且提供一个产品类库时，只想显示它们的接口而不是实现，出于以上情况的原因，我们采用抽象工厂（Abstract Factory）模式，该模式的一般结构图如下：



具体到我们的系统场景中，设计如下：



抽象工厂设计模式通常在运行时刻创建一个 ConcreteFactory 类的实例，这一具体的工厂创建具有特定实现的产品对象，为创建不同的产品对象，客户应使用不同的具体工厂。AbstractFactory 将产品对象的创建延迟到其子类 ConcreteFactory 类中。

这种设计模式具有以下优点：

1.分离了具体的类。由于一个工厂封装创建产品对象的责任和过程，它将客户与类

的实现分离，客户可以通过它们的抽象接口操纵实例。产品的类名也在具体工厂的实现中被分离，不出现在客户代码中；

2.使得易于交换产品系列。一个具体工厂类在一个应用中仅出现一次，即在它初始化的时候，这使得改变一个应用的具体工厂变得很容易，只需要改变具体的工厂即可使用不同的产品配置，这是因为一个抽象工厂创建了一个完整的产品系列，所以整个产品系列会立刻改变；

3.有利于产品的一致性。当一个系列中的产品对象被设计成一起工作时，一个应用一次只能使用同一个系列中的对象，如统一调用 Basic 系列或 Detail 系列用于展示。

3.2 命令模式

命令模式是一种将一个请求封装为一个对象，从而使得可用不同的请求对客户进行参数化；对请求排队或记录请求日志，以及支持可撤销的操作的对象行为型设计模式。

命令模式中有五类参与者，分别为：

1.Command：声明执行操作的接口；

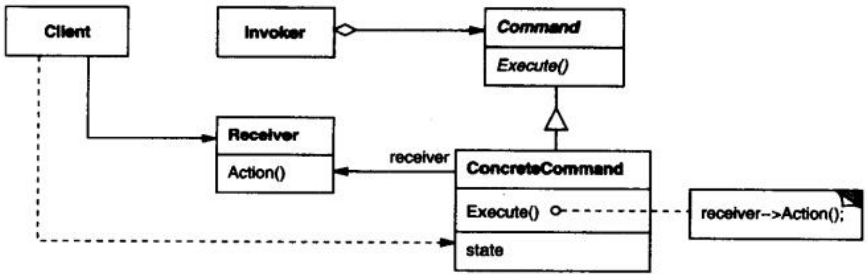
2.ConcreteCommand：将一个接收者对象绑定于一个动作，以当调用接收者相应的操作时，执行 Execute；

3.Client：创建一个具体命令对象并设定它的接收者；

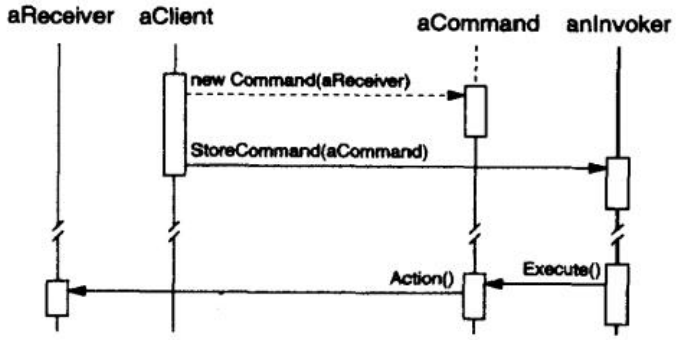
4.Invoker：要求该命令执行这个请求；

5.Receiver：知道如何实施与执行一个请求相关的操作。

其结构如下图所示：



在协作时，Client 创建一个 ConcreteCommand 对象并指定它的 Receiver 对象，某 Invoker 对象存储该 ConcreteCommand 对象，该 Invoker 通过调用 Command 对象的 Execute 操作来提交一个可撤销的请求，ConcreteCommand 执行 Execute 操作之前存储当前状态以用于撤销，ConcreteCommand 对象对调用它的 Receiver 进行一些操作以执行该请求。



在本系统中，用户的点击、查询或搜索事件将作为 a Invoker，通过调用预先设定好的交互逻辑中的 Command 对象的 Execute 操作来提交一个可撤销的请求，由 ConcreteCommand 和 Receiver 协作完成对于用户动作的处理，并返回数据。

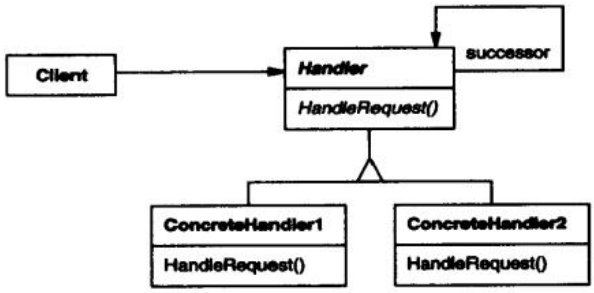
3.3 职责链模式

在对于信息的解析和获取情形下，我们采用职责链模式。

职责链模式，是使多个对象都有机会处理请求，从而避免请求的发送者和接收者之间的耦合关系，将这些对象连成一条链，并沿着这条链传递该请求，直到有一个对象处理它为止的一种对象行为型设计模式。

当我们给定一个 URL 时，可能会有多个不同的网页可以处理该请求，也可能需要返回错误信息，这种哪个对象处理该请求是在请求运行时刻自动确定的需求适用于职责链模式。

一个典型的结构如下图：



或：

这种设计模式降低耦合度，使得一个对象（本系统中是网页）无需知道是哪一个其他对象处理该请求；同时也增强了给对象指派职责的灵活性。

4 其他可用设计模式分析

4.1 创建型

简单工厂模式：

定义：定义了一个创建对象的类，由这个类来封装实例化对象的行为。

简单工厂存在的问题与解决方法：简单工厂模式有一个问题就是，类的创建依赖工厂类，也就是说，如果想要拓展程序，必须对工厂类进行修改，这违背了开闭原则，所以，从设计角度考虑，有一定的问题。对于如何解决该问题，我们可以定义一个创建对象的抽象方法并创建多个不同的工厂类实现该抽象方法，这样一旦需要增加新的功能，直接增加新的工厂类就可以了，不需要修改之前的代码。

4.2 结构型

组合模式：

定义：有时又叫作部分-整体模式，它是一种将对象组合成树状的层次结构的模式，用来表示“部分-整体”的关系，使用户对单个对象和组合对象具有一致的访问性。

意图：将对象组合成树形结构以表示“部分-整体”的层次结构。组合模式使得用户对单个对象和组合对象的使用具有一致性。

主要解决：它在我们树型结构的问题中，模糊了简单元素和复杂元素的概念，客户程序可以向处理简单元素一样来处理复杂元素，从而使得客户程序与复杂元素的内部结构解耦。

关键代码：树枝内部组合该接口，并且含有内部属性 List，里面放 Component。

组合模式的主要优点有：

1.组合模式使得客户端代码可以一致地处理单个对象和组合对象，无须关心自己处理的是单个对象，还是组合对象，这简化了客户端代码；

2.更容易在组合体内加入新的对象，客户端不会因为加入了新的对象而更改源代码，满足“开闭原则”。

其主要缺点是：

1.设计较复杂，客户端需要花更多时间理清类之间的层次关系；

2.不容易限制容器中的构件；

3.不容易用继承的方法来增加构件的新功能；

抽象构件（Component）角色：它的主要作用是为树叶构件和树枝构件声明公共接口，并实现它们的默认行为。在透明式的组合模式中抽象构件还声明访问和管理子类的接口；在安全式的组合模式中不声明访问和管理子类的接口，管理工作由树枝构件完成。

树叶构件（Leaf）角色：是组合中的叶节点对象，它没有子节点，用于实现抽象构件角色中声明的公共接口。

树枝构件（Composite）角色：是组合中的分支节点对象，它有子节点。它实现了抽象构件角色中声明的接口，它的主要作用是存储和管理子部件，通常包含 Add()、Remove()、GetChild()等方法。

代理模式：

定义：代理模式给某一个对象提供一个代理对象，并由代理对象控制对原对象的引用。通俗地来讲代理模式就是我们生活中常见的中介。

开闭原则，增加功能：代理类除了是客户类和委托类的中介之外，我们还可以通过给代理类增加额外的功能来扩展委托类的功能，这样做我们只需要修改代理类而不需要再修改委托类，符合代码设计的开闭原则。代理类主要负责为委托类预处理消息、过滤消息、把消息转发给委托类，以及事后对返回结果的处理等。代理类本身并不真正实现服务，而是同过调用委托类的相关方法，来提供特定的服务。真正的业务功能还是由委托类来实现，但是可以

在业务功能执行的前后加入一些公共的服务。例如我们想给项目加入缓存、日志这些功能，我们就可以使用代理类来完成，而没必要打开已经封装好的委托类。

优点：可以做到在符合开闭原则的情况下对目标对象进行功能扩展。

缺点： 代理对象与目标对象要实现相同的接口，我们得为每一个服务都得创建代理类，工作量太大，不易管理。同时接口一旦发生改变，代理类也得相应修改。

4.3 行为型

模版方式模式：

定义：定义一个操作中算法的骨架，而将一些步骤延迟到子类中，模板方法使得子类可以不改变算法的结构即可重定义该算法的某些特定步骤。

通俗点的理解就是：完成一件事情，有固定的数个步骤，但是每个步骤根据对象的不同，而实现细节不同；就可以在父类中定义一个完成该事情的总方法，按照完成事件需要的步骤去调用其每个步骤的实现方法。每个步骤的具体实现，由子类完成。

抽象父类（AbstractClass）：实现了模板方法，定义了算法的骨架。

具体类（ConcreteClass）：实现抽象类中的抽象方法，即不同的对象的具体实现细节。

优点：

1.具体细节步骤实现定义在子类中，子类定义详细处理算法是不会改变算法整体结构。

2.代码复用的基本技术，在数据库设计中尤为重要。

3.存在一种反向的控制结构，通过一个父类调用其子类的操作，通过子类对父类进行扩展增加新的行为，符合“开闭原则”。

缺点：

每个不同的实现都需要定义一个子类，会导致类的个数增加，系统更加庞大。

观察者模式：

定义：定义对象间的一种一对多的依赖关系，当一个对象的状态发生改变时，所有依赖于它的对象都得到通知并被自动更新。

优点：

1.观察者和被观察者是抽象耦合的。

2.建立一套触发机制。

缺点：

1.如果一个被观察者对象有很多的直接和间接的观察者的话，将所有的观察者都通知到会花费很多时间。

2.如果在观察者和观察目标之间有循环依赖的话，观察目标会触发它们之间进行循环调用，可能导致系统崩溃。

3.观察者模式没有相应的机制让观察者知道所观察的目标对象是怎么发生变化的，而仅仅只是知道观察目标发生了变化。

抽象被观察者角色：也就是一个抽象主题，它把所有对观察者对象的引用保存在一个集合中，每个主题都可以有任意数量的观察者。抽象主题提供一个接口，可以增加和删除观察者角色。一般用一个抽象类和接口来实现。

抽象观察者角色：为所有的具体观察者定义一个接口，在得到主题通知时更新自己。

具体被观察者角色：也就是一个具体的主题，在集体主题的内部状态改变时，所有登记过的观察者发出通知。

具体观察者角色：实现抽象观察者角色所需要的更新接口，一边使本身的状态与制图的状态相协调。

5 总结与参考

5.1 总结

在本次设计模式报告中，我们首先调研了工厂模式中的简单工厂模式，工厂方法模式等，但是对于简单工厂模式来说每次新增一个功能类就要在工厂类中增加一个判断，对于工厂方法模式来说，新增一个功能类，就要创建对应的工厂类，相比简单工厂模式，免去了判断创建的具体实例，但是会创建很多的类，这点就很冗余，因此我们选择了抽象工厂模式，这样的好处在于避免创建更多的功能实例。

另外，我们选择职责链模式不选择状态模式的原因是职责链模式并不指定下一个处理的对象到底是谁，避免发送者和接受者之间的耦合关系。对于命令模式，其封装的是整个类，操作要相对容易一些！

5.2 参考

[1]. https://en.wikipedia.org/wiki/Abstract_factory_pattern

[2]. https://en.wikipedia.org/wiki/Command_pattern

[3]. https://en.wikipedia.org/wiki/Chain_of_responsibility

[4].刘伟，夏莉，于俊洋，黄辛迪. 设计模式[M]. 北京: 清华大学出版社, 2018

[5].杨俊峰. 软件设计模式的最佳实践探索[J]. 企业家, 2019, 000(027):P.201-201

[6].陈天超. 单例设计模式研究[J]. 福建电脑, 2016,(08):14-15+20.