程序设计模式报告

16281164 荆栋

导语

程序设计模式的实验围绕程序语言的多态特性展开。其中开闭原则是指导思想：程序对修改是封闭的，对拓展是开放的。相比于面向过程语言，面向对象语言解决的是需求改动时的变动，而不是算法问题。

本次所有课程实例均采用 C++编程实现。其实现多态的方式有继承、虚函数、函数重载和运算符重载等。结合成员变量和函数的开放范围，各种实用的程序设计模式被设计出来。

本文档先介绍学习设计模式的图示工具 UML，然后再总结不同类型的设计模式。最后写一些学习感受。

最后是针对提交的代码的说明。提交的.CPP 文件默认是将 main 函数注释掉的，若要运行出正确结果，请把 main 相关的注释取消。这样做是因为我是在一个项目下开发的，各个设计模式实例相互独立，一个项目只能有一个入口。

目录

程序设计模式报告 1
导语 1
一、 UML 3
二、 创建型模式 3
2.1 工厂方法模式 3
2.2 抽象工厂模式 4
2.3 建造者模式 5
2.4 单例模式 6
三、 结构体模式 7
3.1 配适器模式 7
3.2 桥接模式 8
3.3 装饰模式 9
3.4 外观模式 11
3.5 享元模式 12
3.6 代理模式 13
四、 行为型模式 14
4.1 职责链模式 14
4.2 命令模式 15
4.3 中介者模式 16
4.4 观察者模式 17
4.5 策略模式 18
五、 课程学习感受 19

一、 UML

类图（UML）描述类与类之间的静态关系。

类的图形符号为长方形，其中上中下三个区域分别存放类的名字、属性和服务。属性的可见性有下述三种：public、private 和 protected，分别用加号、减号和井号来表示。如果未声明可见性，则表示没有默认的可见性。

类与类之间通常有关联、泛化（继承）、依赖和细化 4 种关系。

普通关联是最常见的关联关系，只要类与类之间存在连接关系就可以用普通关联表示。**普通关联的图示符号是连接两个类之间的直线。**

聚集也成为聚合，是关联的特例。聚集表示类与类之间的关系是整体与部分的关系。聚集有两种聚集：共享聚集和组合聚集。如果在聚集关系中处于部分方的对象可同时参与多个处于整体方对象的构成，则该聚集称为共享聚集。**共享聚集用空心菱形线表示。**如果部分类完全属于整体类，部分与整体共存，整体不存在了部分也会随之消失，则该聚集为组合聚集。**组合聚集用实心菱形来表示。**

泛化关系就是通常所说的继承关系。**泛化关系用空心三角形的连线来表示。**

依赖关系描述两个模型元素之间的语义连接关系：其中一个模型元素是独立的，另一个模型元素不是独立的，它依赖于独立的模型元素，如果独立的模型元素改变，将影响依赖于它的模型元素。**有依赖关系的两个类用带有箭头的虚线连接，箭头指向独立的类。**

当对同一事物在不同抽象层次上描述时，这些描述之间有细化关系。例如实现 JAVA 的接口的类与该接口之间有细化关系。**细化关系用一端为空心三角形的虚线连接，被指向的是抽象层次高的。**

二、 创建型模式

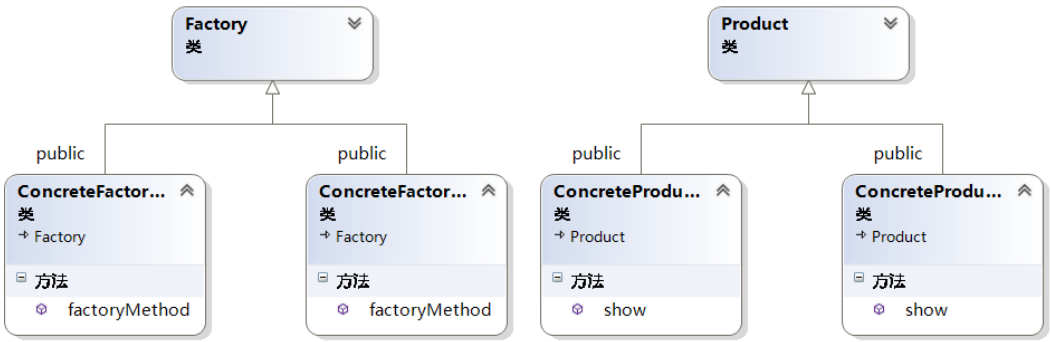
创建型模式(Creational Pattern)对类的实例化过程进行了抽象，能够将软件模块中对象的创建和对象的使用分离。包含如下模式：简单工厂模式、工厂方法模式、抽象工厂模式、建造者模式、原型模式和单例模式。

2.1 工厂方法模式

工厂方法模式(Factory Method Pattern)中，工厂父类负责定义创建产品对象的公共接口，而工厂子类则负责生成具体的产品对象，这样做的目的是将产品类的实例化操作延迟到工厂子类中完成，即通过工厂子类来确定究竟应该实例化哪一个具体产品类。

我实现的例子就是标准的不同的工厂生产对应的不同产品。

具有抽象工厂、抽象产品、具体工厂、具体产品四个角色。



（其中 factoryMethod 的返回值为 ConcreteProduct 类对象指针，存在依赖关系。但这里 vs studio 默认生成的类图里没有显示出来…）

工厂方法模式优点：

在工厂方法模式中，工厂方法用来创建客户所需要的产品，同时还向客户隐藏了哪种具体产品类将被实例化这一细节，用户只需要关心所需产品对应的工厂，无须关心创建细节，甚至无须知道具体产品类的类名。使用工厂方法模式的另一个优点是在系统中加入新产品时，无须修改抽象工厂和抽象产品提供的接口，无须修改客户端，也无须修改其他的具体工厂和具体产品，而只要添加一个具体工厂和具体产品就可以了。这样，系统的可扩展性也就变得非常好，完全符合“开闭原则”。

2.2 抽象工厂模式

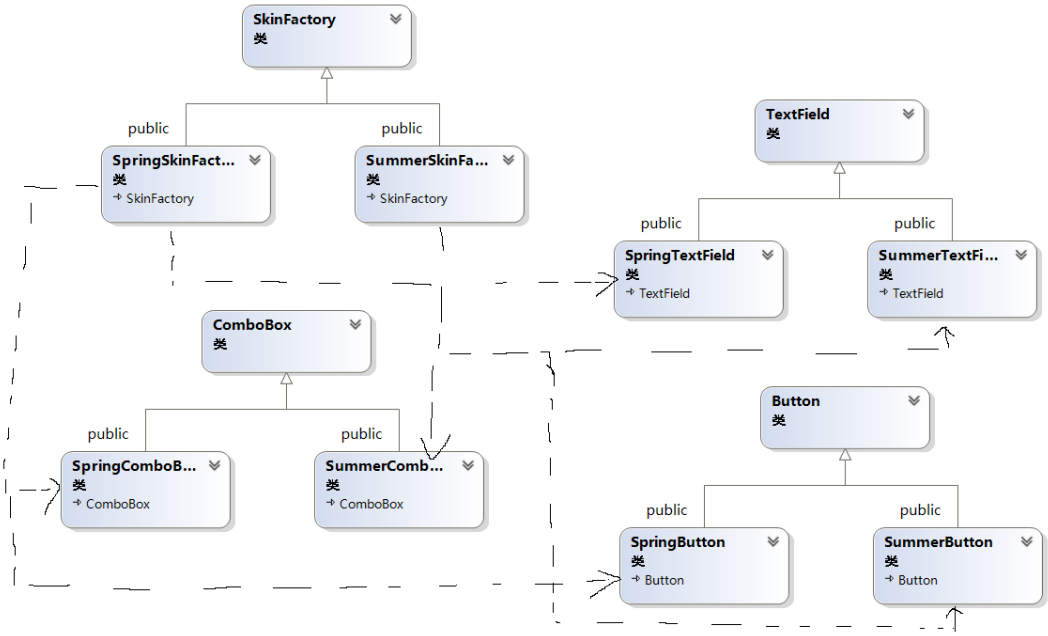
抽象工厂模式(Abstract Factory Pattern)：提供一个创建一系列相关或相互依赖对象的接口，而无须指定它们具体的类。

抽象工厂模式可以看作是工厂方法模式的一种改进。在工厂方法模式中具体工厂负责生产具体的产品，每一个具体工厂对应一种具体产品，工厂方法也具有唯一性，一般情况下，一个具体工厂中只有一个工厂方法或者一组重载的工厂方法。但是有时候我们需要一个工厂可以提供多个产品对象，而不是单一的产品对象。抽象工厂方法可以满足我们这种需求。

实例情景设计如下：

某软件公司要开发一套界面皮肤库。不同的皮肤将提供视觉效果不同的按钮、文本框、组合框等界面元素。例如春天(Spring)风格的皮肤将提供浅绿色的按钮、绿色边框的文本框和绿色边框的组合框。而夏天(Summer)风格的皮肤则提供浅蓝色的按钮、蓝色边框的文本框和蓝色边框的组合框。

UML：



抽象工厂模式优点：

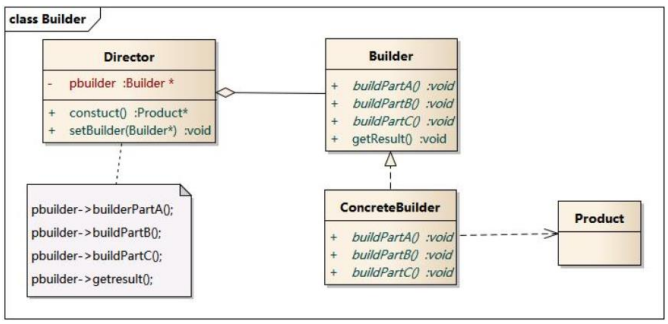
封装了产品的创建，用户不需要知道具体的产品信息，只需要知道工厂信息即可。可以支持工厂生成不同类型的产品，相比于工厂方法模式灵活性更强。增加新的具体工厂和产品族很方便，无须修改已有系统，符合“开闭原则”。此外，抽象工厂模式实现了高内聚低耦合的设计目的。

2.3 建造者模式

建造者模式(Builder Pattern)：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。

一般的建造者模式包括如下角色：

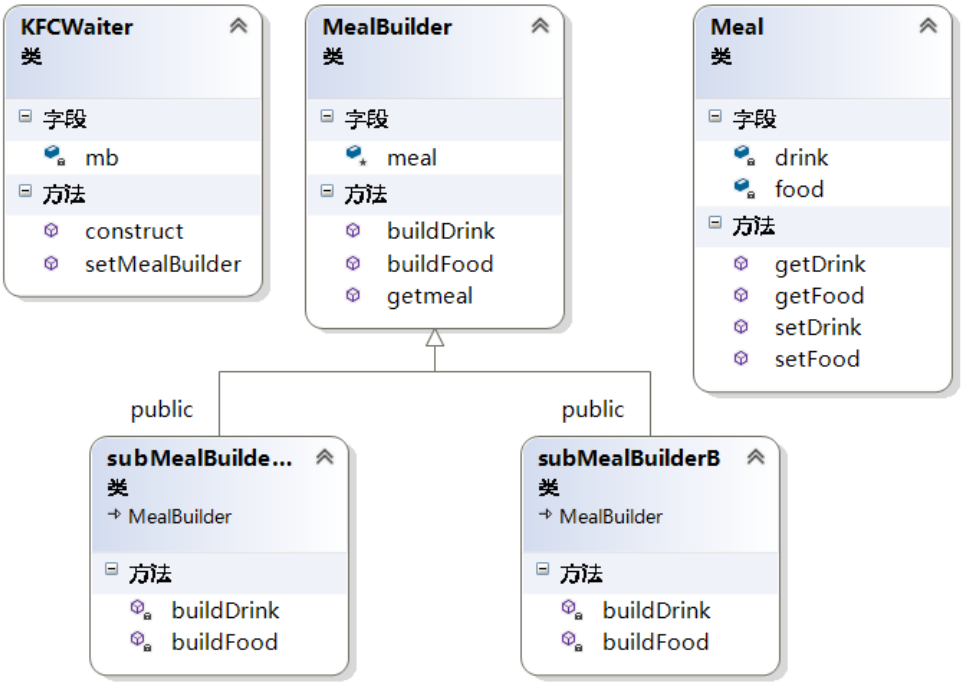
Builder：抽象建造者；ConcreteBuilder：具体建造者；Director：指挥



者；Produce：产品角色。

实例情景如下：KFC 卖套餐，一般包含主食和饮料等组成部分。KFC 的服务员可以根据顾客的需求生成套餐。

UML：



建造者模式的优点：

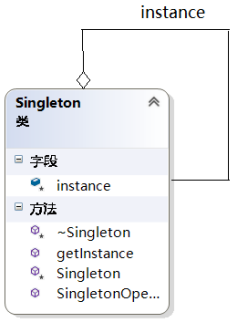
在建造者模式中， 客户端不必知道产品内部组成的细节，将产品本身与产品的创建过程解耦，使得相同的创建过程可以创建不同的产品对象。每一个具体建造者都相对独立，而与其他的具体建造者无关，因此可以很方便地替换具体建造者或增加新的具体建造者，用户使用不同的具体建造者即可得到不同的产品对象。符合开闭原则。

2.4 单例模式

单例模式(Singleton Pattern)：单例模式确保某一个类只有一个实例，而且自行实例化并向整个系统提供这个实例，这个类称为单例类，它提供全局访问的方法。

单例模式的要点有三个：一是某个类只能有一个实例；二是它必须自行创建这个实例；三是它必须自行向整个系统提供这个实例。

本次实例实现的是标准的单例模式：



单例模式在代码实现时有几个需要注意的地方：单例类的构造函数为私有；提供一个自身的静态私有成员变量；提供一个公有的静态工厂方法；**静态私有成员变量在类内声明，在类外定义。**

单例模式的优点：

提供了对唯一实例的受控访问。因为单例类封装了它的唯一实例，所以它可以严格控制客户怎样以及何时访问它。此外，单例模式只能生成一个对象，节约了系统资源，可以提高系统性能。

三、 结构体模式

结构型模式(Structural Pattern)描述如何将类或者对象结合在一起形成更大的结构。不同的结构型模式从不同的角度组合类或对象，它们在尽可能满足各种面向对象设计原则的同时解决了相关问题。

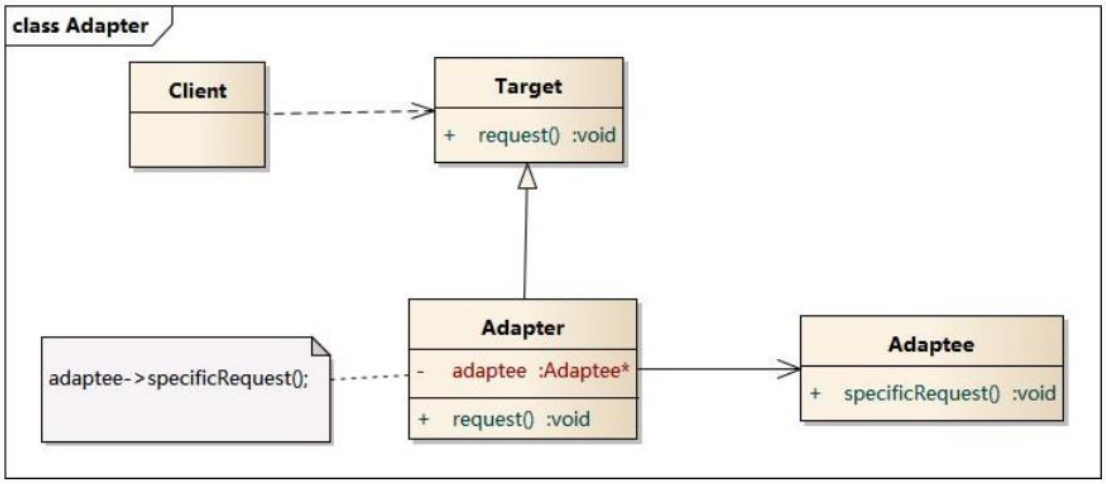
结构体模式包含如下模式：配适器模式、桥接模式、组合模式、装饰模式、外观模式、享元模式和代理模式。

3.1 配适器模式

适配器模式(Adapter Pattern) ：将一个接口转换成客户希望的另一个接口，适配器模式使接口不兼容的那些类可以一起工作，其别名为包装器(Wrapper)。

配适器模式包含如下角色：Target：目标抽象类；Adapter：配适器类；Adaptee：配适者类；Client：客户类。

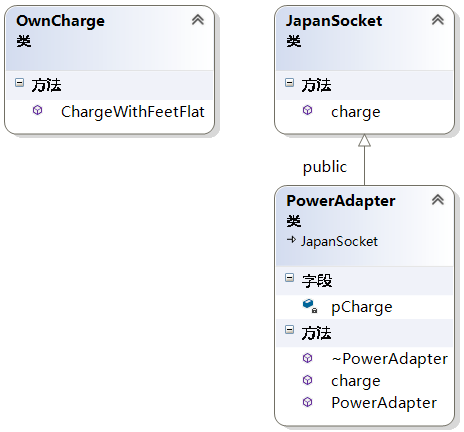
一般形式的 UML 如下：



客户端先初始化一个 Adaptee 对象。然后使用 Target 基类指针初始化一个 Adapter 类对象（该类对象使用之前初始化的 Adaptee 对象构造），然后就可以直接利用 Target 指针调用功能函数。

实例情景设计：去国外旅游时，不同国家可能提供功率不同的电源插口，因此需要用到电源配适器实现功率的统一。

UML 如下：



配适器模式的优点：

将目标类和适配者类解耦，通过引入一个适配器类来重用现有的适配者类，而无须修改原有代码。增加了类的透明性和复用性，将具体的实现封装在适配者类中，对于客户端类来说是透明的，而且提高了适配者的复用性。

3.2 桥接模式

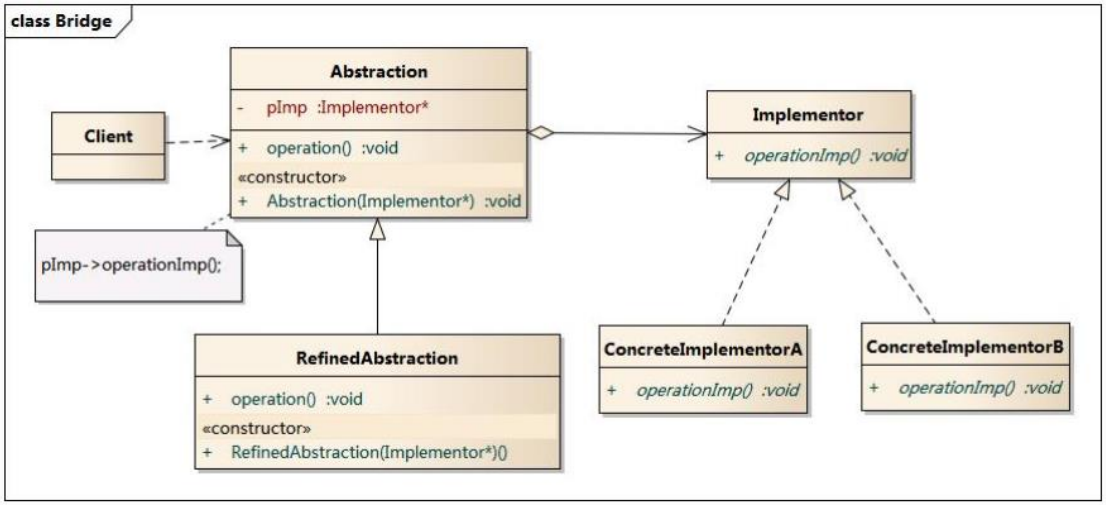
桥接模式(Bridge Pattern)：将抽象部分与它的实现部分分离，使它们都可以独立地变化。

其基本实现包括如下角色：

Abstraction：抽象类；RefinedAbstraction：扩充抽象类；Implementor：

实现类接口；ConcreteImplementor：具体实现类。

UML 如下：



理解桥接模式，重点需要理解如何将抽象化(Abstraction)与实现化(Implementation)脱耦，使得二者可以独立地变化。抽象化就是忽略一些信息，把不同的实体当作同样的实体对待。在面向对象中，将对象的共同性质抽取出来形成类的过程即为**抽象化**的过程。针对抽象化给出的具体实现，就是**实现化**。**脱耦**就是将抽象化和实现化之间的耦合解脱开，或者说是将它们之间的强关联改换成弱关联，将两个角色之间的继承关系改为关联关系。

这里我实现的实例就是上述的标准样例。

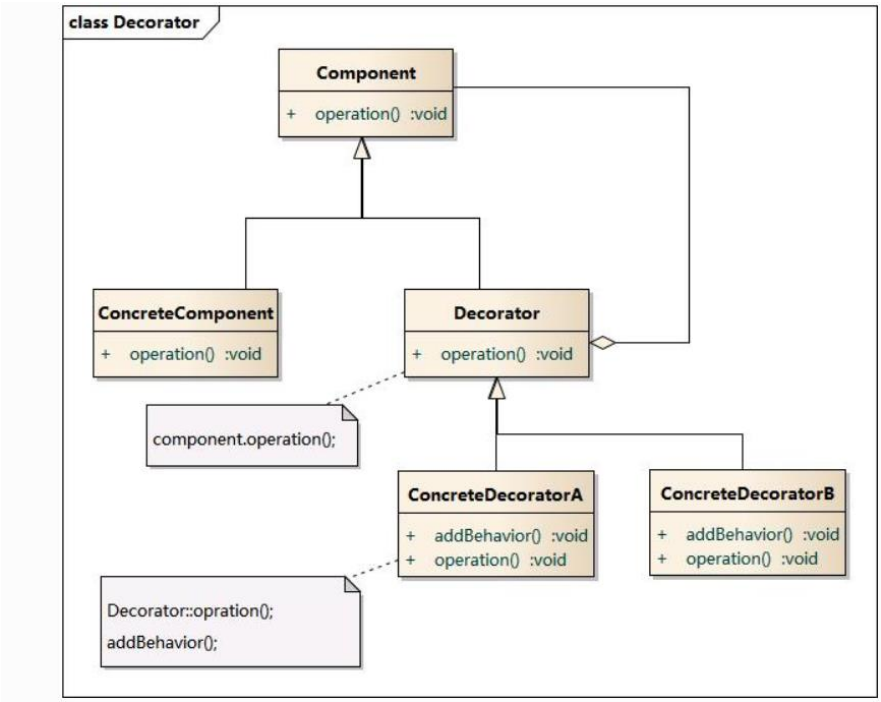
桥接模式的优点：

分离抽象接口及其实现部分。桥接模式有时类似于多继承方案，但是多继承方案违背了类的单一职责原则，复用性比较差，而且多继承结构中类的个数非常庞大，桥接模式是比多继承方案更好的解决方法。桥接模式提高了系统的可扩充性，在两个变化维度中任意扩展一个维度，都不需要修改原有系统。

3.3 装饰模式

装饰模式(Decorator Pattern) ：动态地给一个对象增加一些额外的职责。就增加对象功能来说，装饰模式比生成子类实现更为灵活。

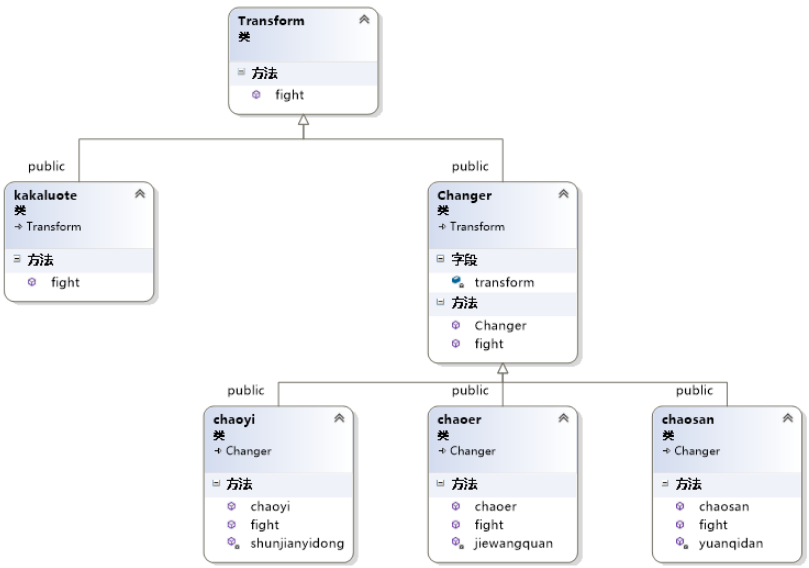
一般的装饰器模式结构如下：



可以看到，装饰模式在不改变一个对象本身功能的基础上给对象添加了额外的功能，是一种替代继承的技术。

这里我设想的场景是龙珠里孙悟空的变身。赛亚人具有战斗天赋，有可能超越极限觉醒成传说中的超级赛亚人。但是卡卡罗特更加变态，除了超一，他还有超二和超三的形态。假定卡卡罗特在超一时可以使用瞬间移动；在超二时可以使用界王拳；在超三时可以随意凝聚元气弹（当然这些设定和原著不符合，只用于模拟）。

UML：



作为一个龙珠粉，这里我写的很开心。输出如下：

```
C:\WINDOWS\system32\cmd.exe
kakaluote go!
super Saiyan 1 go!
shunjianyidong !
super Saiyan 2 go!
jiewangquan !
super Saiyan 3 go!
yuanqidan !
super Saiyan 3 go!
yuanqidan !
请按任意键继续. . .
```

装饰模式的优点：

装饰模式与继承关系的目的都是要扩展对象的功能，但是装饰模式可以提供比继承更多的灵活性。通过使用不同的具体装饰类以及这些装饰类的排列组合，可以创造出很多不同行为的组合。可以使用多个具体装饰类来装饰同一对象，得到功能更为强大的对象。具体构件类与具体装饰类可以独立变化，用户可以根据需要增加新的具体构件类和具体装饰类，在使用时再对其进行组合，原有代码无须改变，符合“开闭原则”

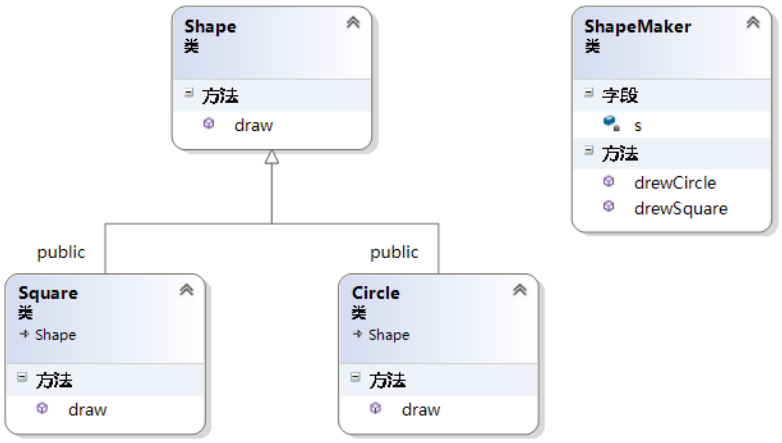
3.4 外观模式

外观模式(Facade Pattern)：外部与一个子系统的通信必须通过一个统一的外观对象进行，为子系统中的一组接口提供一个一致的界面，外观模式定义了一个高层接口，这个接口使得这一子系统更加容易使用。

外观模式是一个简单而且使用的模式。首先，根据“单一职责原则”，在软件中将一个系统划分为若干个子系统有利于降低整个系统的复杂性。其次，外观模式从很大程度上提高了客户端使用的便捷性，使得客户端无须关心子系统的工作细节，通过外观角色即可调用相关功能。

情景设计：画图工具里可以画图形。可选的形状 shape 里可以有不同的图形，比如说 square,circle 等。

UML：



外观模式的优点：

它对客户端屏蔽了子系统组件，减少了客户端所需处理的对象数目，并使得

子系统使用起来更加容易。实现了子系统与客户之间的松耦合关系，这使得子系统的组件变化不会影响到调用它的客户类，只需要调整外观类即可。只是提供了一个访问子系统的统一入口，并不影响用户直接使用子系统类。

3.5 享元模式

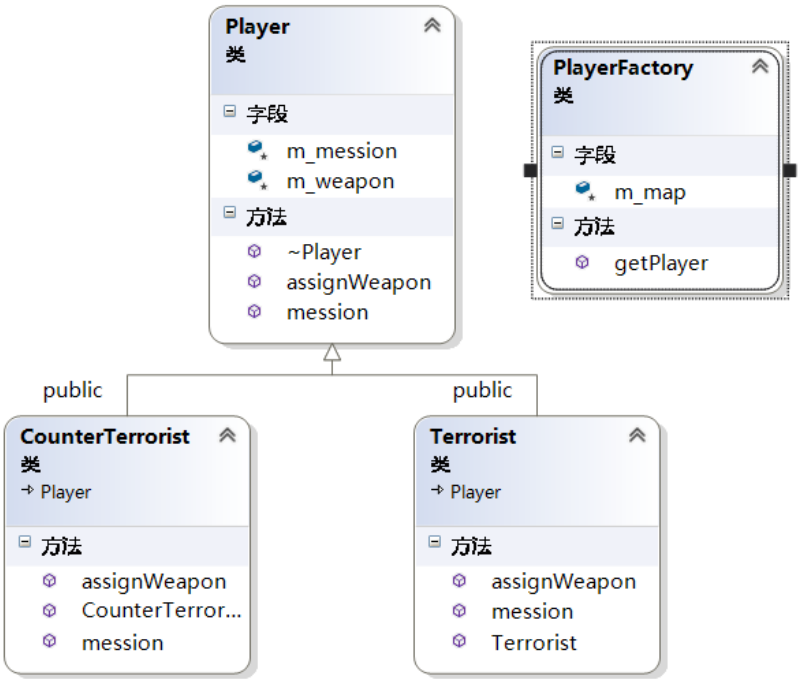
享元模式(Flyweight Pattern)：运用共享技术有效地支持大量细粒度对象的复用。系统只使用少量的对象，而这些对象都很相似，状态变化很小，可以实现对象的多次复用。由于享元模式要求能够共享的对象必须是细粒度对象，因此它又称为轻量级模式，它是一种对象结构型模式。

我认为享元模式的实用性非常高。面向对象技术可以很好地解决一些灵活性或可扩展性问题，但在很多情况下需要在系统中增加类和对象的个数。当对象数量太多时，将导致运行代价过高，带来性能下降等问题。享元模式就可以很好的解决这一问题。

享元模式的核心在于享元工厂类，享元工厂类的作用在于提供一个用于存储享元对象的享元池，用户需要对象时，首先从享元池中获取，如果享元池中不存在，则创建一个新的享元对象返回给用户，并在享元池中保存该新增对象。

实例情景设计：CS 游戏有警匪之分。但是在创建角色时可以使用享元模式来节省空间。

UML：



这个也是我觉得比较有意思的例子。输出如下：

```
D:\小栋的学习日历！\程序设计模式\code\PatternDesign\bin\Debug\PatternDesign.exe
create CT player
Counter Terrorist with weapon m4a1, Task is diffuse bomb!
create T player
Terrorist with weapon ak47, Task is plant a bomb!
Counter Terrorist with weapon m4a1, Task is diffuse bomb!
Terrorist with weapon ak47, Task is plant a bomb!
Terrorist with weapon ak47, Task is plant a bomb!
Counter Terrorist with weapon m4a1, Task is diffuse bomb!
Counter Terrorist with weapon m4a1, Task is diffuse bomb!
Counter Terrorist with weapon m4a1, Task is diffuse bomb!
Counter Terrorist with weapon m4a1, Task is diffuse bomb!
Counter Terrorist with weapon m4a1, Task is diffuse bomb!

Process returned 0 (0x0)   execution time : 0.082 s
Press any key to continue.
```

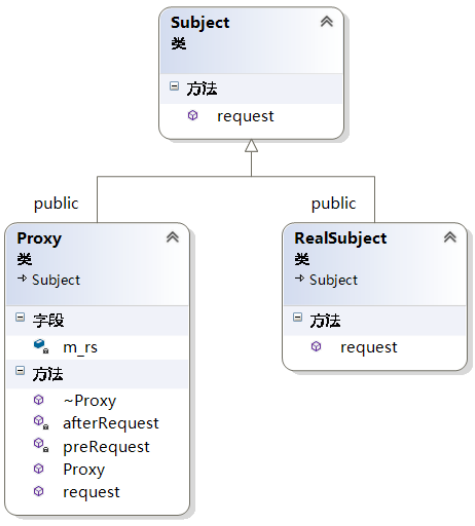
享元模式的优点：

享元模式的优点在于它可以极大减少内存中对象的数量，使得相同对象或相似对象在内存中只保存一份。享元模式的外部状态相对独立，而且不会影响其内部状态，从而使得享元对象可以在不同的环境中被共享。

3.6 代理模式

代理模式(ProxyPattern)：给某一个对象提供一个代理，并由代理对象控制对原对象的引用。

代理无论在日常生活中还是在通信服务、计算机架构等方面都有很广泛的应用。代理模式包含三个角色：抽象主题角色声明了真实主题和代理主题的共同接口；代理主题角色内部包含对真实主题的引用，从而可以在任何时候操作真实主题对象；真实主题角色定义了代理角色所代表的真实对象，在真实主题角色中实现了真实的业务操作，客户端可以通过代理主题角色间接调用真实主题角色中定义的方法。



情景设计：虚拟代理经常在系统中被使用。如果需要创建一个资源消耗较大的对象，先创建一个消耗较小的对象来表示，真正的对象只有在需要时才会被创建。

代理模式的优点：

代理模式能够协调调用者和被调用者，在一定程度上降低了系统的耦合度。远程代理使得客户端可以访问在远程机器上的对象，远程机器可能具有更好的计算性能与处理速度，可以快速响应并处理客户端请求。虚拟代理通过使用一个小对象来代表一个大对象，可以减少系统资源的消耗，对系统进行优化并提高运

行速度。保护代理可以控制对真实对象的使用权限。

四、 行为型模式

行为型模式(Behavioral Pattern)是对在不同的对象之间划分责任和算法的抽象化。

行为型模式分为类行为型模式和对象行为型模式两种：

类行为型模式：类的行为型模式使用继承关系在几个类之间分配行为，类行为型模式主要通过多态等方式来分配父类与子类的职责。

对象行为型模式：对象的行为型模式则使用对象的聚合关联关系来分配行为，对象行为型模式主要是通过对象关联等方式来分配两个或多个类的职责。根据“合成复用原则”，系统中要尽量使用关联关系来取代继承关系，因此大部分行为型设计模式都属于对象行为型设计模式。

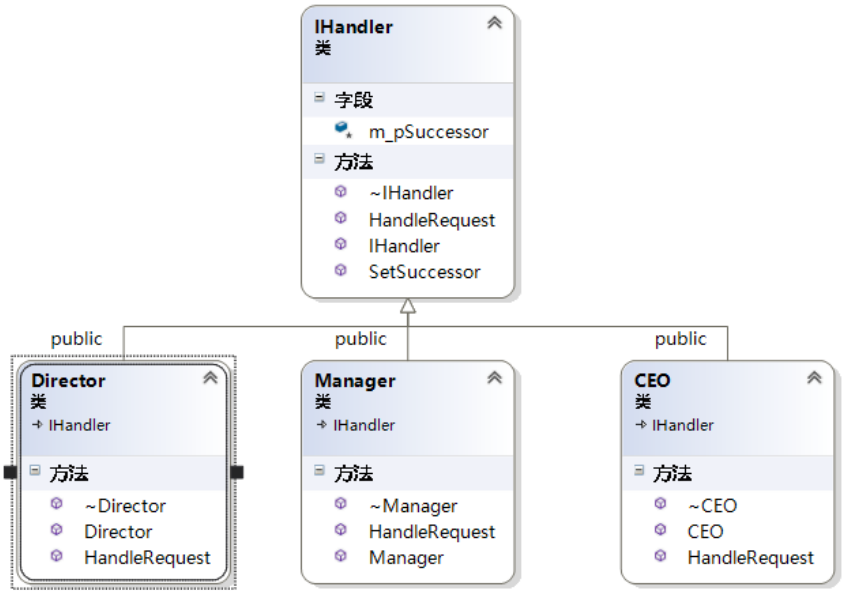
行为型模式包含如下模式：职责链模式、命令模式、解释器模式、迭代器模式、中介者模式、备忘录模式、观察者模式、状态模式、策略模式、模板方法模式和访问者模式。

4.1 职责链模式

职责链模式（Chain of Responsibility Pattern）：避免将一个请求的发送者与接收者耦合在一起，让多个对象都有机会处理请求。将接收请求的对象连接成一条链，并且沿着这条链传递请求，直到有一个对象能够处理它为止。

客户端无须关心请求的处理细节以及请求的传递，只需将请求发送到链上，将请求的发送者和请求的处理者解耦。这种模式很适合如下这些场景去使用：1、有多个对象可以处理同一个请求，具体哪个对象处理该请求由运行时刻自动确定。2、在不明确指定接收者的情况下，向多个对象中的一个提交一个请求。 3、可动态指定一组对象处理请求。

实例情景设计：员工发出请假请求。经理、总监、总裁有权批准或者传递给上级批准。但是不同等级的处理者批准天数的权限大小不同：经理(1 天及以下)，总监(3 天及以下)，总裁(7 天为界限)。



UML：

职责链模式的优点：

1、降低耦合度。它将请求的发送者和接收者解耦。 2、简化了对象。使得对象不需要知道链的结构。 3、增强给对象指派职责的灵活性。通过改变链内的成员或者调动它们的次序，允许动态地新增或者删除责任。 4、增加新的请求处理类很方便。

4.2 命令模式

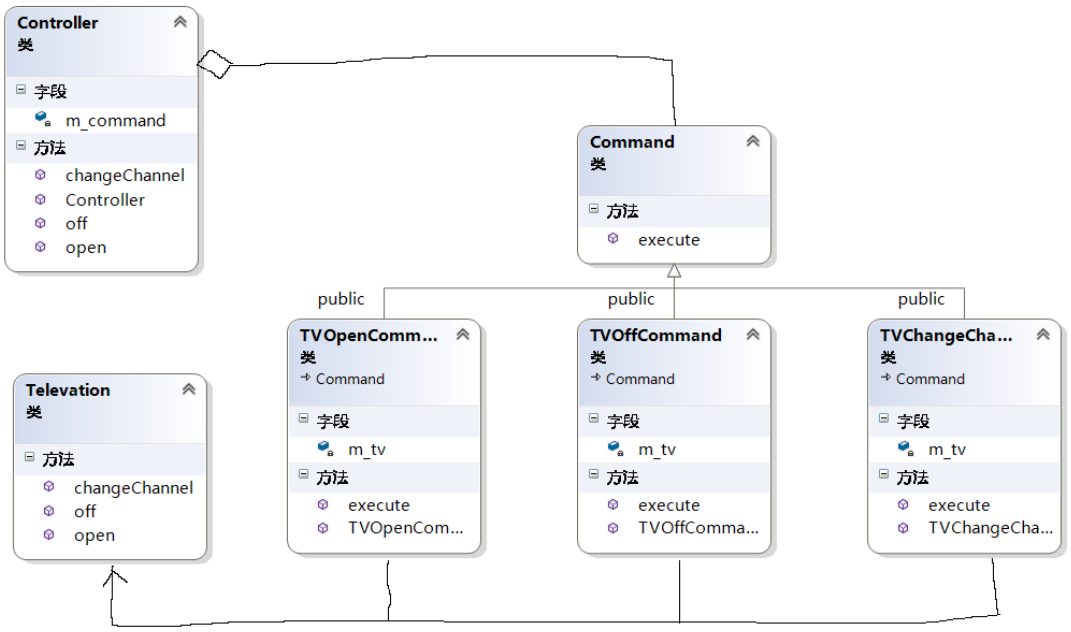
命令模式(Command Pattern)：将一个请求封装为一个对象，从而使我们可用不同的请求对客户进行参数化；对请求排队或者记录请求日志，以及支持可撤销的操作。

命令模式的本质是对命令进行封装，将发出命令的责任和执行命令的责任分割开。每一个命令都是一个操作：请求的一方发出请求，要求执行一个操作；接收的一方收到请求，并执行操作。命令模式允许请求的一方和接收的一方独立开来，使得请求的一方不必知道接收请求的一方的接口，更不必知道请求是怎么被接收，以及操作是否被执行、何时被执行，以及是怎么被执行的。命令模式使请求本身成为一个对象，这个对象和其他对象一样可以被存储和传递。

基本的命令模式包含四个角色：抽象命令类中声明了用于执行请求的execute()等方法，通过这些方法可以调用请求接收者的相关操作；具体命令类是抽象命令类的子类，实现了在抽象命令类中声明的方法，它对应具体的接收者对象，将接收者对象的动作绑定其中；调用者即请求的发送者，又称为请求者，它通过命令对象来执行请求；接收者执行与请求相关的操作，它具体实现对请求的业务处理。

情景设计：电视机作为请求的接收者。遥控器是请求的发送者。假设遥控器有仨按钮，分别对应三种操作：打开电视机、关闭电视机、切换频道。

UML：



命令模式的优点：

最大的优点是可以方便地实现对请求的 Undo 和 Redo。此外，可以比较容易地设计一个命令队列和宏命令（组合命令），新的命令可以很容易地加入到系统中。降低系统的耦合度。

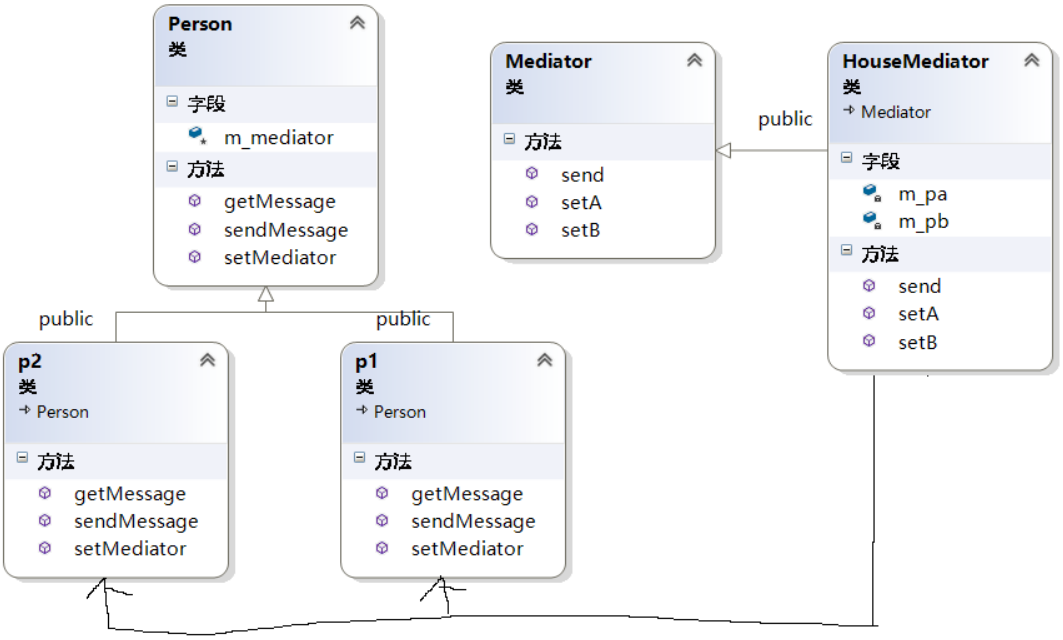
4.3 中介者模式

中介者模式(Mediator Pattern)定义：用一个中介对象来封装一系列的对象交互，中介者使各对象不需要显式地相互引用，从而使其耦合松散，而且可以独立地改变它们之间的交互。

一般的中介者模式包含四个角色：**抽象中介者**用于定义一个接口，该接口用于与各同事对象之间的通信；**具体中介者**是抽象中介者的子类，通过协调各个同事对象来实现协作行为，了解并维护它的各个同事对象的引用；**抽象同事类**定义各同事的公有方法；**具体同事类**是抽象同事类的子类，每一个同事对象都引用一个中介者对象；每一个同事对象在需要和其他同事对象通信时，先与中介者通信，通过中介者来间接完成与其他同事类的通信；在具体同事类中实现了在抽象同事类中定义的方法。

本次实例的情景：租房。房东可以把房子信息放到中介。房客可以在中介咨询房子信息。

UML：



中介者模式的优点：

简化了对象之间的交互，将各同事解耦，还可以减少子类生成，对于复杂的对象之间的交互，通过引入中介者，可以简化各同事类的设计和实现。

4.4 观察者模式

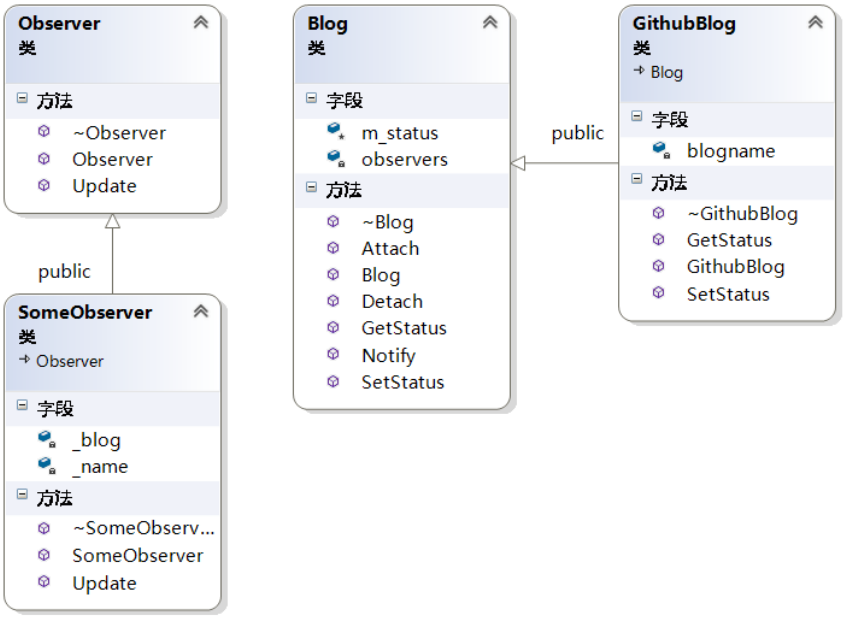
观察者模式(Observer Pattern)：定义对象间的一种一对多依赖关系，使得每当一个对象状态发生改变时，其相关依赖对象皆得到通知并被自动更新。

观察者模式又叫做发布-订阅系统，这有助于我们理解观察者模式发挥的主要功能。一般的观察者模式包含四个角色：目标又称为主题，它是指被观察的对象；具体目标是目标类的子类，通常它包含有经常发生改变的数据，当它的状态发生改变时，向它的各个观察者发出通知；观察者将对观察目标的改变做出反应；在具体观察者中维护一个指向具体目标对象的引用，它存储具体观察者的有关状态，这些状态需要和具体目标的状态保持一致。

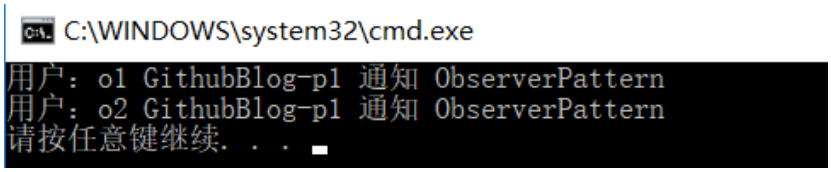
观察者模式描述了如何建立对象与对象之间的依赖关系，如何构造满足这种需求的系统。这一模式中的关键对象是观察目标和观察者，一个目标可以有任意数目的与之相依赖的观察者，一旦目标的状态发生改变，所有的观察者都将得到通知。

本次实例情景设计为一个订阅-发布系统：世界上最大的同性交友网站 github 已经进入了很多人的视野，很多人在上面发 blog。如果有用户关注了一个博主，该博主发 blog 后用户会收到提醒。

UML 如下：



执行结果：



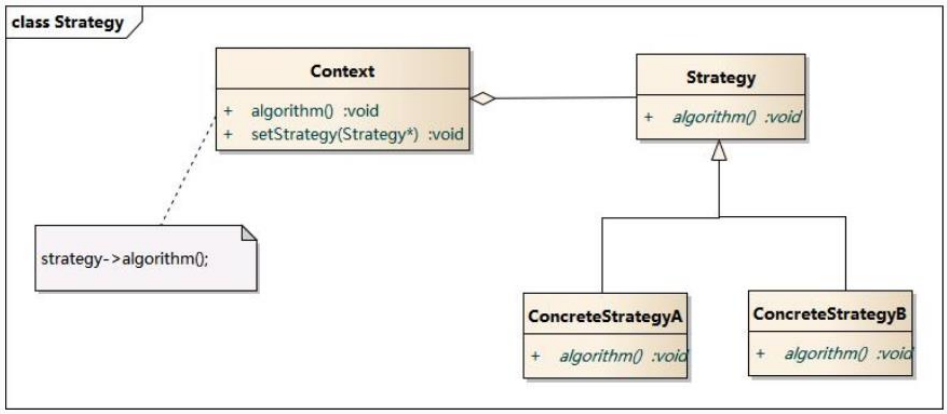
观察者模式的优点：

观察者模式可以实现表示层和数据逻辑层的分离，并定义了稳定的消息更新传递机制，抽象了更新接口，使得可以有各种各样不同的表示层作为具体观察者角色。观察者模式在观察目标和观察者之间建立一个抽象的耦合。观察者模式符合“开闭原则”的要求。

4.5 策略模式

策略模式(Strategy Pattern)：定义一系列算法，将每一个算法封装起来，并让它们可以相互替换。

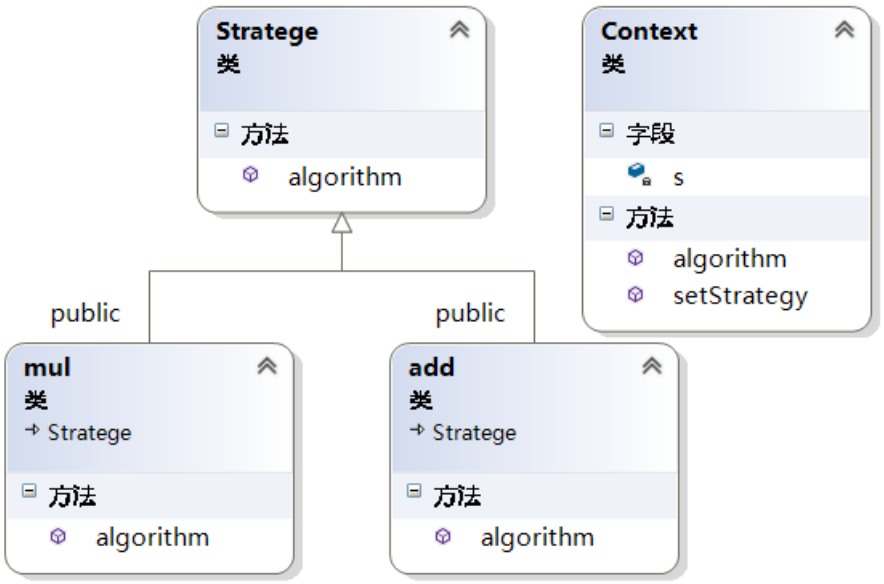
策略模式理解起来比较容易，我们可以参考标准实现的 UML：



标准的策略模式包含三个角色：环境类在解决某个问题时可以采用多种策略，在环境类中维护一个对抽象策略类的引用实例；抽象策略类为所支持的算法声明了抽象方法，是所有策略类的父类；具体策略类实现了在抽象策略类中定义的算法。

情景设计：这次设计的简单一些。我们可以控制对于输入的两个操作数是采取加操作还是乘操作。

UML：



策略模式的优点：

策略模式提供了对“开闭原则”的完美支持，用户可以在不修改原有系统的基础上选择算法或行为，也可以灵活地增加新的算法或行为。策略模式提供了管理相关的算法族的办法。策略模式提供了可以替换继承关系的办法。

五、 课程学习感受

首先最直观的感受就是，这是我接触面向对象语言许久以来第一次真正地发挥面向对象特性的优势。之前在学习 JAVA 和 C++时，只是理解了这些语言的语法规范和特性，并没有真正的发挥他们的功能。其次，程序设计地目的是为了减少工作量、提升正确率和提高软件效率等。设计模式可以帮我我们更好地实现这些目的。最后则是让我增进了对 C++使用的熟练度。

这里再来回顾一些要点。

开闭原则：对修改关闭，对拓展开放。

C++接口实现：virtual function=0; C++静态私有成员变量需在内部声明，在外部进行初始化。

读写类图，学会看类图写函数和客户端代码。