设计模式

——第一步

Time：2024-07-13

目 录

[一、一点形而上学 1](#_Toc171847898)

# 一、一点形而上学

好的设计意味着当我作出改动，整个程序就好像正等着这种改动。 我可以仅调用几个函数就完成任务，而代码库本身无需改动。

最小化在编写代码前需要了解的信息；

简单：最简单、最直接的解决方案；不用为每种情况编写条件逻辑

抽象和解耦让扩展代码更快更容易，但除非确信需要灵活性，否则不要在这上面浪费时间；

在整个开发周期中为性能考虑并做好设计，但是尽可能推迟那些底层的，基于假设的优化，那会锁死代码；

快速地探索游戏的设计空间，但不要跑得太快，在身后留下烂摊子。毕竟你总得回来打扫；

如果打算抛弃这段代码，就不要尝试将其写完美。摇滚明星将旅店房间弄得一团糟，因为他们知道明天就走人了；

如果你想要做出让人享受的东西，那就享受做它的过程。

# 二、设计模式

## 1.命令模式

GoF：将一个请求封装为一个对象，从而使你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。

命令模式是一种回调的面向对象实现。

命令是具现化的方法调用。

将概念变成数据 ——一个对象——可以存储在变量中，传给函数。 所以称命令模式为“具现化方法调用”，意思是方法调用被存储在对象中。

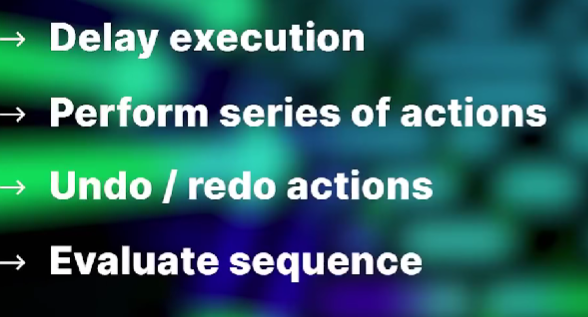
命令模式是一种行为设计模式，它将方法及其所需参数封装或打包到一个单独的命令对象中。

命令对象封装了单个操作并被设计为实现一个抽象的命令或命令接口；

命令调用程序负责执行和管理存储的命令；

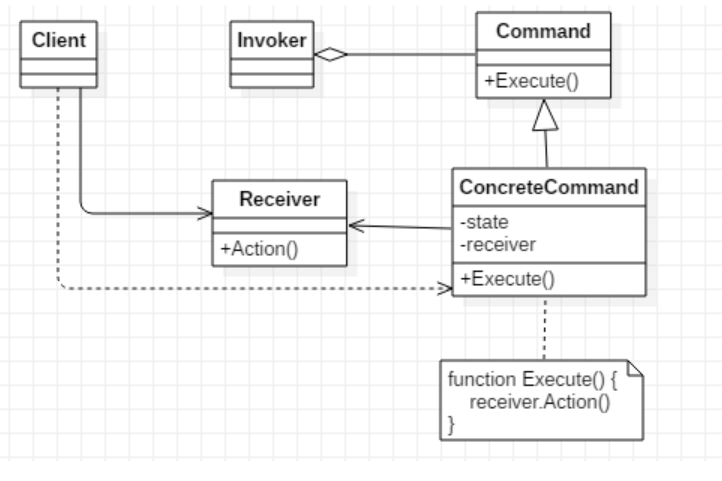
参考：[设计模式与游戏完美开发 (蔡升达 著) - void87 - 博客园 (cnblogs.com)](https://www.cnblogs.com/revoid/p/10647060.html#19)

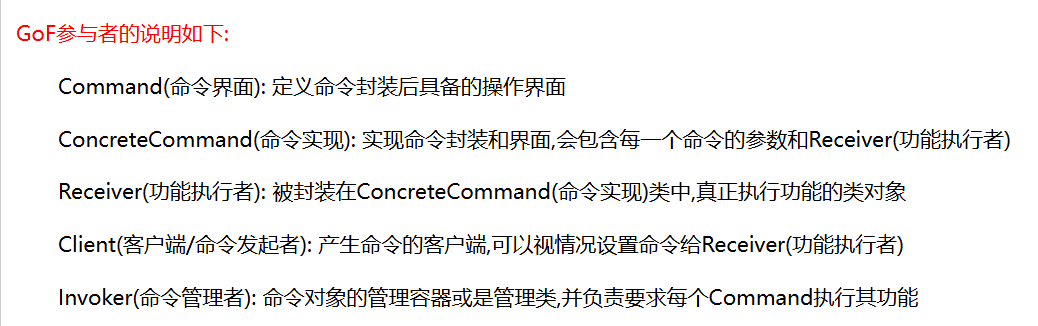
### 1.1 用途



参考：<https://www.bilibili.com/video/BV1ii4y1W7i6/?spm_id_from=333.337.search-card.all.click&vd_source=2b832b8a10300fd76ab04105bbd93c80>

### 1.2 实现





不使用模式：硬编码



这像是我会干的事

（函数式编程风格）

其实就干两件事：

请求的封装，将原来的硬编码部分（某种功能）封装在一个类中

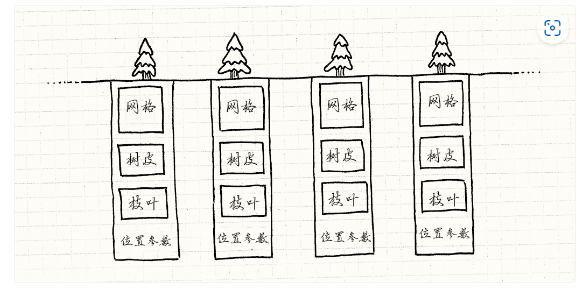
请求的操作：当请求可以被封装成一个对象时,那么这个请求对象就可以被操作（可以被实例化了）

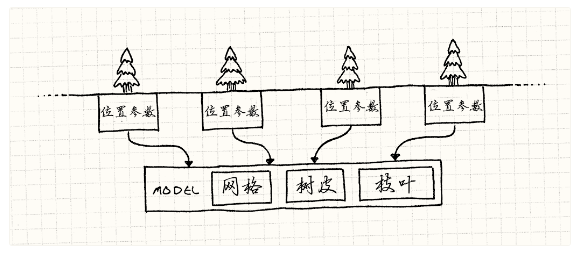
### 1.3 开发实例

Command类

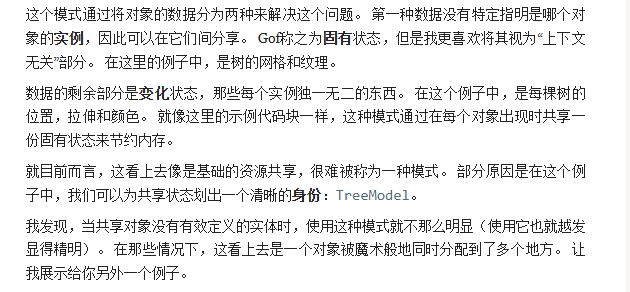
## 2.享元模式

### 1.1基本概念

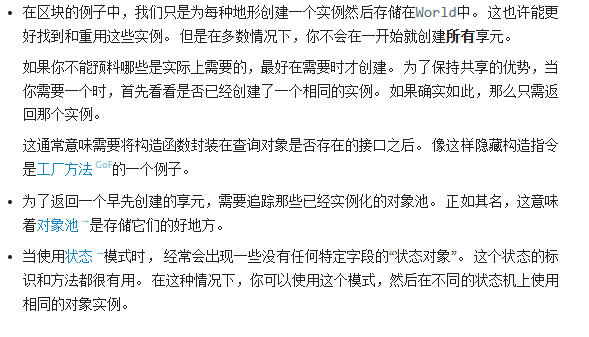




将原本一堆数据中共有的数据拿出来分离到另一个类中，游戏中每个对象得实例只需有一个对这个共享部分model的引用。



以上的共享部分比较清晰，能够很直接的划分出来。



### 1.2 举个例子

### 1.3 实现

## 3.观察者模式

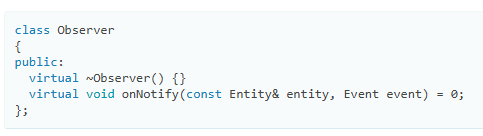
### 3.1 基本概念

这让代码宣称有趣的事情发生了，而不必关心到底是谁接受了通知。

即通知有事情发生了，至于谁处理我不管

#### （1）观察者

观察者接口



具体观察者：继承这个接口

#### （2）被观察者

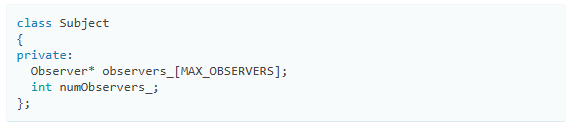
被观察者：拥有通知的方法函数。

两个任务：

①有一个列表，保存等它通知的观察者对象；

②暴露公开的API修改这个列表；

③发送通知；







 在多线程引擎中，你最好使用[事件队列](https://gpp.tkchu.me/event-queue.html)来做异步通信。

每当角色脸上（或者其他什么地方）挨了一拳，就发送一个通知。 UI观察到了，然后更新健康槽。

观察者模式是一个让这些不相关的代码块互相交流，而不必打包成更大的块的好方法。 这在专注于一个特性或层面的单一代码块内不会太有用。

### 3.2 举个例子

解耦成就系统和其它部分

### 3.3 实现

## 4.单例模式



### 4.1 基本概念

### 4.2 举个例子

### 4.3 实现

## 3.观察者模式

### 3.1 基本概念

### 3.2 举个例子

### 3.3 实现

# 三、心得

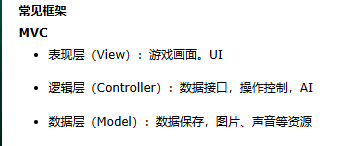
## 1.MVC

[【总结】游戏框架与架构设计（Unity为例） - sols - 博客园 (cnblogs.com)](https://www.cnblogs.com/sols/p/8455279.html)

入门简介：[MVC模式简介 - 底云飞 - 博客园 (cnblogs.com)](https://www.cnblogs.com/diyunfei/p/6752618.html)

推荐：[使用 MVC 和 MVP 编程模式构建模块化代码库 (unity.com)](https://unity.com/cn/how-to/build-modular-codebase-mvc-and-mvp-programming-patterns)

最基本的一个图例



但是在具体写的时候怎么体现：

第一点：业务逻辑不要写到View里面

（什么是业务逻辑）

（对于数据的处理）

M是指业务模型，V是指用户界面，C则是控制器

将M和V的实现代码分离，，使同一个程序可以使用不同的表现形式

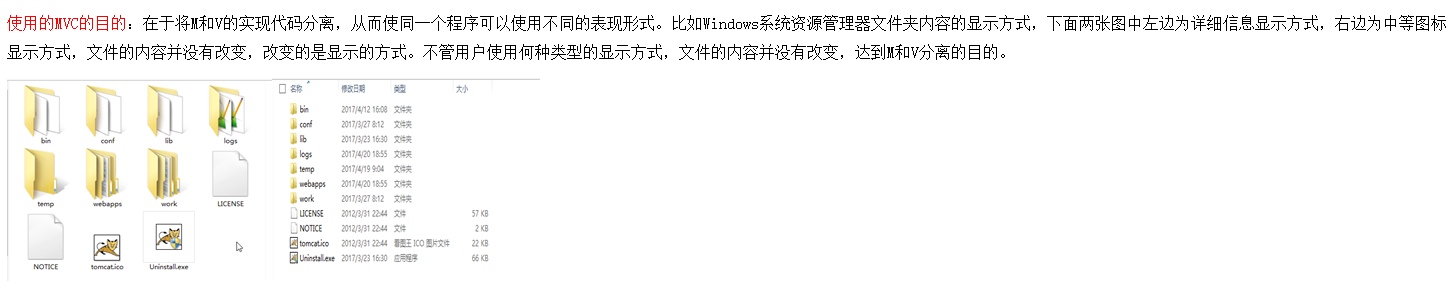
文件的内容并没有改变，改变的是显示的方式

那么我哪些是M，代码里面的耦合度，一个东西的变动我需要改几个地方

视图中其实没有真正的处理发生，它只是作为一种输出数据并允许用户操纵的方式；

模型返回的数据是中立的，模型与数据格式无关；

控制器接受用户的输入并调用模型和视图去完成用户的需求，控制器本身不输出任何东西和做任何处理。它只是接收请求并决定调用哪个模型构件去处理请求，然后再确定用哪个视图来显示返回的数据。



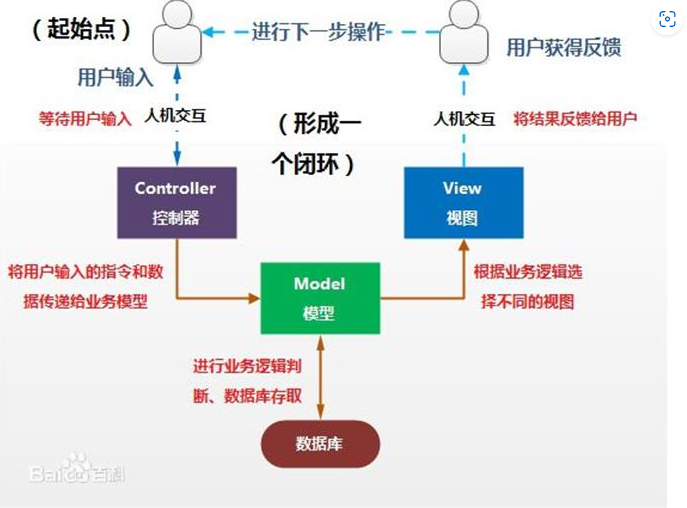
### 1.1 基本法

确定哪部分是M，哪部分是V，搞清楚业务逻辑

视图中其实没有真正的处理发生，它只是作为一种输出数据并允许用户操纵的方式

模型返回的数据是中立的，模型与数据格式无关

控制器接受用户的输入并调用模型和视图去完成用户的需求，控制器本身不输出任何东西和做任何处理。它只是接收请求并决定调用哪个模型构件去处理请求，然后再确定用哪个视图来显示返回的数据



Model：存放数据以及处理数据逻辑的地方，涉及对数据的操作或者算法放在这里完成，在一些MVC框架中，业务也可以在这里完成，就目前经验来看，Model是单例，是单个功能模块的数据管理中心；

Controller：一些指令操作、控制，不涉及业务部分仅仅涉及到如何分析用户指令，过滤处理指令，以及控制展示放在这里，将交互获取到的信息传给Model

View：就干一件事，展示界面。View的数据获取有两种方式，第一是通过单例Module进行填充；第二是接口定义需要接受的参数，使用controller控制，传参数进行调用。

一个问题是在逻辑概念上可以拆分出来，到具体编码的时候简单的可以拆分，逻辑复杂一点的可能就有混淆

解决逻辑是，使用一个View来启动MVC，剩下的逻辑交给MVC框架来处理

### 1.2 实现

MVC在具体实现上，就参考来看，在每个模块应该负责的功能上基本没有争议，但是在具体的实现上，每个