一. 代码与设计模式

在此之前，我们先来理清楚这样一个问题，我们写代码到底是在写什么？我们可以理解

从上图我们可以大概总结到，我们的项目，主要是模块和沟通，设计模式也是让我们写出更好的设计模块，更好的连接他们之间的沟通。

二.设计模式扮演的角色

- 帮助我们组织模块，通过一些设计模式，组织模块间的组织模块
- 帮助我们更好的设计沟通，有的设计模块可以帮助我们设计模块间如何沟通
- 提高代码质量 通过设计模式让代码更优雅

二.设计原则

- 开闭原则　我们的程序应该对扩展开放，对修改关闭，我们的程序要给具体使用的时候扩展功能的接口，但是在具体使用的时候，我们不能让它修改我们的源码，也就是说，我们可以扩展功能，但是我们又不需要修改源码，像我们的jq，vue，react插件，都是开闭原则的最好体现。
- 单一职责原则 我们模块只做一件事情，我们的模块越单一越好
- 依赖倒置原则 我们的上层模块不要依赖具体的下层模块，而是依赖于抽象，我们举个例子：我们食物有很多种，炒饭，炒面 等等，我们会选择餐厅，厨房，来作为依赖，而不是以具体的事物作为依赖，我们可以看下具体的代码

```
function food1(){
}
function food2(){
}
function food3(){
}
function food4(){
}

function order(){

}
order.prototype.orderFood1=function(){

}
order.prototype.orderFood2=function(){

}
order.prototype.orderFood3=function(){

}
```

看上述代码，如果我们把每个食物，都写一个一个方法，我们点餐就得往 order原型方法上面添加，有多少就得添加多少，这样做会搞死人。再看下面代码，

```
function resturn(food){
  var list={
    food1:new food1(),
    food2:new food2(),
    food3:new food3()
  }
  return list[food];
}

function order(food){
  return resturn(food);
}
```

如果我们再点餐和食物中间加一个中间层，resturn函数，点餐只是依赖中间层resturn，而不具体的食物，不管食物怎么更改变更，我们都不理会，只管中间层resturn

- 接口隔离原则 接口应该细化，功能应该单一，这和上面的单一模块原则有点类似，但是关注点不同，模块单一原则，是关注模块，接口隔离原则是关注接口
- 迪米特法则 最少知识原则 我们让两个对象去沟通，我们尽量让两个对象双方知道的越少越好，中间者模式就是迪米特法则的思想
- 里氏替换原则 主要关注的是继承，任何父类使用地方，都可以用子类进行替换用直白的话说，我们用子类去继承父类，必须保证子类完全继承父类的方法和属性，这样的话父类使用的地方，子类可以进行替换

二.设计模式的分类

1. 创建型：设计模式大家可以看成就是帮助和指导我们怎么去创建对象、怎么去创建模块以及怎么去设计模块之间的沟通，创建型的目的就是帮助我们更优雅的创建对象
2. 结构型：帮助我们更优雅的设计整体以及局部的代码结构
3. 行为型：它是模块间的行为模式总结，目的是帮助我们组织模块行为，组织它们之间怎么互相沟通来达成功能目标
4. 技巧型：帮助我们优化代码

创建型中有四个设计模式：

- 工厂模式-大量创建对象：工厂模式相当于建造一个创建对象的工厂，然后告诉工厂我们需要什么对象，工厂就把对象返回给我们，工厂模式适用于大量创建对象的场景。

- 单例模式-全局只能有我一个：单例模式意思是我们需要如何设计代码，保证全局某一个特定的实例化对象只能有一个。

- 建造者模式-精细化组合对象：建造者模式适用于创建一个复杂的对象，其实就是精细化组合一个对象。

- 原型模式-JavaScript的灵魂：原型模式相当于原型链，它的意义是通过定义原型，后面创建的对象都依赖这个原型。

通过以上四个设计模式可以看出创建型设计模式都是指导我们如何去创建对象。

结构型中有五个设计模式：

- 外观模式-给你一个套餐：外观模式就像我们去餐厅点一个套餐，我们不用关心具体要求点什么菜，只需要点具体的套餐就行了，对于代码就是说我们把接口细化之后可以给调用者一个套餐，而不是让他关心具体应该使用哪一个接口。
- 享元模式-共享来减少数量：享元模式的目的主要是为了减少对象或者代码块的数量，当存在大量重复的对象或代码块时，可以观察一下这些对象和代码块之间有哪些不同的地方，然后把不同的地方提取为一个公共的享元，通过这个公共的享元来减少对象或代码块的数量。
- 适配器模式-用适配代替更改：当存在两个对象之间接口、数据不适配的时候，我们不用去更改这两个对象，而是通过写一段适配代码把接口或者数据进行适配。
- 桥接模式-独立出来，然后再对接过去：桥接模式关注于减少代码的耦合度，它的做法是将代码中的一些东西独立出来，然后再组合回去，来减少代码之间的耦合度。
- 装饰者模式-更优雅的扩展需求：装饰者模式主要应对于当我们的方法不满足现在的需求时如何更优雅的去扩展这个方法。

通过以上五个设计模式可以看出结构型设计模式是帮助我们更优雅的设计整体以及局部的代码结构。

行为型中有六个设计模式：

- 观察者模式-我作为第三方转发：观察者模式相当于定义一个第三方，模块间的沟通通过这个第三方来进行转发。
- 职责链模式-像生产线一样组织模块：职责链模式相当于把各个模块之间组织成一条生产线，这个模块完成了就把任务交到下一个模块继续包装，依次传递。
- 状态模式-用状态代替判断：状态模式主张用状态代替判断，可以非常有效的减少if else分支，能够让对象根据不同的状态来展现不同的行为。

- 命令模式-用命令去解耦：命令模式指导我们用命令去解除执行者与命令者之间的耦合，我们无需关心执行者具体是谁、他要做什么，我们只需要给执行者一道命令执行者就会去执行。
- 策略模式-算法工厂：策略模式和状态模式有点类似，但是策略模式可以理解为一个算法工厂，告诉它要什么算法，它就返回什么算法。
- 迭代器模式-告别for循环：迭代器模式就是我们在不去了解某一个对象内部的情况下能够有序的去遍历这个对象内部，例如ES6的forEach循环。

通过以上六个设计模式可以看出行为型设计模式目的是帮助我们组织模块的行为，组织它们之间怎么互相沟通来达成功能目标。

技巧型有五个设计模式：

- 链模式-链式调用：如jQuery的链式调用。
- 惰性模式-我要搞机器学习：惰性模式说高大上点是搞机器学习，说白点是在第一次执行之后把执行状态记录下来。
- 委托模式-让别人代替你收快递：委托模式相当于收快递，把消息委托别人代收。
- 等待者模式-等你们都回来再吃饭：等待者模式适用于多种异步操作情况下，当我们发出一堆的异步操作之后，需要等待者模式帮助我们让这些异步操作都返回了再进行下一步操作。
- 数据访问模式-一个方便的数据管理器：数据访问模式就是说我们去建立一个数据管理器。

通过以上五个设计模式可以看出技巧型设计模式目的是帮助我们更好的优化代码。