javascript设计模式

依赖倒置

条件，多个return出口?

传入对象参数或者数组，代替大量参数

多态来代替if else switch 实际上就是分成不同对象，执行不同对象的同一方法

传入对象参数，执行同名方法

闭包是为了封装和保存局部变量

函数作为参数 回调 sort

OBject.prototype.tostring.call()

j颗粒化

状态类 状态方法

拥有状态的对象 属性 状态 当前状态 被状态调用的属性

方法 ：设置当前状态

生成状态对象

状态对象 传入当前对象 先执行语句 再修改当前状态

点击委托给状态对象来做

委托的思想 使用者对象 ：当前对象.通用方法()

该方法里面修改使用者属性，最后修改当前状态

开关电灯 三级电灯 文件上传的各种状态

人物状态 动画状态

if state== 与并非

当前状态=FSM.xxx,调用状态方法，在方法中改变属性和当前状态

也可以改变其他人的当前状态

fsm{ 状态：{ 方法1 方法2}

倒地{攻击(){无效} 防御(){无效} 跳跃(){跳起，当前状态变为浮空}}

浮空{攻击(){坠地踢()，当前状态变为站立} }

}

js是无类语言

表驱动的有限状态机

适配器模式 实际上是不改变双方 增加中间层

渲染地图(地图名){地图.show}

百度.display() 谷歌.show()

封装百度 { show() {baidu.display} }

比如不同格式数据合并，中间加个转换层，不必修改两个数据

装饰者 给对象添加功能(比如方法)，不改变自身 属性

传统实现 一个对象引用原对象方法，然后加上自己的方法，组合起来 实际上是包装

然后我们直接使用包装对象就行了，可以层层包装

包装后可以再修改原来方法的引用，使用包装方法，记得每次都要保留原方法引用

p.fire=(){ olffire()

newmethod()

}

装饰函数，不应该修改语句，可以每次保留引用。。。。或者aop

function.before(fn)

a=a.after(b) 随后两个函数会顺序执行

代理 小明.送花(某人){ 对方.收到花()}

代理 小明.送花(小李)

小李.收到花(){小芳.收到花()}

小李是小芳的代理，在中间进行一些过滤之类的操作，然后才转交给小芳处理

保护代理 权限控制 拦截 再做转发

虚拟代理 条件满足才创建对象

预加载 合并http请求，多次请求惰性加载

缓存代理 如果有就放出缓存否则执行被代理函数 ajax异步，已经有的数据就不刷新，否则执行刷新

fn实现函数变量 动态执行

单例模式 如果有 则返回

单例的目的是唯一对象 全局访问，js中直接写个全局对象就行了吧

使用命名空间(其实就是对象包装)避免全局属性

单例是一种思想，点击某些方法创建对象，但我们只需要创建一次，于是给方法执行赋予条件，如果存在就不创建。。。参考对象池

封装为获取单例对象 ()

策略模式和状态模式很像啊，但是策略之间没有联系，用户随意切换

状态机是互相关联的用户不用知道内部发生了什么

策略类 策略对象 同样的方法

设置策略 策略对象.方法

除去if else

js直接用方法名当成策略名，无需策略对象，

a={方法}

对象[方法名]();

策略方法

享元模式 创建大量相似对象 比如类似的球，或者游戏中的粒子 大量图片

2000个文件对象

减少对象数量 内部状态和外部状态

通过修改对象的属性，代替生成多个不同对象 对象池

重复的属性给剥离出来，对象本身属性多，层层继承会占用大量空间

职责链 不写成if else if 处理不了就往后传递

每个人只处理自己能处理的

这种比ifelse麻烦，但是可以灵活组合，每个函数有自己的职责，不会互相耦合

要求灵活可拆分

生成节点 节点添加到链条

传递请求，返回值是成功，或者继续发送到下一节点

用aop实现职责链.after (节点函数2).after(节点3)

中介者模式，关于多对多关系，互相之间传递信息，执行方法通知

中介接收 对象事件法 或者对象使用中介接口，传入自己的参数(如自身对象)，告诉中介自己的改变

命令模式 发送请求

不知道接收请求的对象，也不知道有什么属性方法， 无法实现直接调用

解除行为者 和接收者的耦合

比如按钮 设置命令(发送者，命令)

命令对象(接收者) 接收者.接收者行为()

命令对象 执行 撤销

撤销需要记录原状态，条件是未改变之前

重做

命令队列

宏命令 子命令，一次执行一批 添加命令，同时执行

其实直接用回调实现不就好了?直接 执行receiver.方法

组合模式 整体和部分 组成树

使用一个请求，逐层向子对象提出请求，一次执行所有方法，或者部分方法

add remove 子对象引用

树遍历 扫描文件夹

减少对象之间的耦合

对象访问，推送

第5章 大话设计模式

类图

类的关系 实际上是对象之间的关系 has a

对象指向

对象内部的对象

依赖

组合

聚合

继承 三角

接口实现 虚线三角

把对象和main分离 界面和逻辑分离

单一职责 开放封闭 依赖倒转 里氏代换

迪米特法则

反射技术

设计模式实现不修改其他的，我觉得最好的办法就是传参

不要为了模式而模式，js有很多好的实现

策略模式 策略对象选择 执行策略方法

策略对象工厂，传入字符串生成对象

简单工厂，用字符串参数自动生成对象，违反开放封闭原则，需要修改分支

工厂方法，把判断移到客户端 new不同的工厂

原型模式 对象拷贝，，深复制浅复制

模板方法

模板方法是虚方法，在父类中直接调用在关键地方，就是让子类重写。。。最好方法直接return 答案

就相当于模板字符串一样，只是个占位符

不变的在父类，变动的在子类

建造模式，生成复杂对象的(多对象复合)

观察者模式

发消息对象 发消息方法 接收对象数组

添加接收对象，移除接收对象 add remove

消息方法(接收对象)｛接收对象.反应()｝

里面有个遍历接收数组的过程

事件委托

避免switch代码改动

依赖注入

反射 由类名生成对象

这里类名是字符串，可以写成变量，所以可以灵活转换

createInstance(name) 工厂方法也是这样

还可以用配置文件代替写字符串

方法太长，状态转换的条件表达式太多，

状态模式 自动调用状态对象通用方法

自动切换状态对象

备忘录模式 外部保存 对象内部状态以后恢复对象到原来的状态

需要保存状态的对象，生成备忘录对象，

组合模式 叶节点(组合的对象) 分支节点(用来控制叶) 增加 移除 执行

迭代器模式 顺序访问对象的所有元素 但是保持封装 如for in等 都是语言自带的

线程锁 避免单例被同时访问

双重