目录

[设计模式 1](#_Toc11134)

[复习： 3](#_Toc8519)

[一、 观察者模式 4](#_Toc335)

[1.1 基础观察者： 4](#_Toc16833)

[1.2 DOM二级观察者模式： 4](#_Toc12474)

[1.3 移除事件 5](#_Toc7448)

[1.4 绑定只触发一次的事件方法 5](#_Toc3473)

[二、 策略模式 6](#_Toc14108)

[2.1 jquery中的策略模式 6](#_Toc13958)

[2.1 登录策略 6](#_Toc15699)

# 复习：

设计模式：

一套编目分明、广为人知、可复用的代码经验的总结

工具库：

一些常用方法的集合体，例如jquery、underscore，这些方法之间是没有联系的

框架：

一套半成品代码，它里面也支持一些方法，但是这些方法之间通常是有联系的

架构：

一套大型项目的设计思路

设计模式的分为与作用：

所有的设计模式分为三类：

1 创建型设计模式： 解决了创建对象时候的问题

2 结构型设计模式： 解决了对象和类组合在一起时候的问题

3 行为型设计模式： 解决了对象和类耦合、职责之间的关系

简单工厂：其实就是创建对象的函数

寄生增强工厂：定义一个新的工厂，在该工厂内部实例化另一个工厂，并对其改造（增强）

工厂方法：通过一个工厂可以管理多个工厂的实例化创建

原型：每一个函数天生拥有的属性，它的值是一个对象

特点：原型中的所有内容都可以被实例化对象所访问

继承：类与类之间的继承

继承的方式：

1 类式继承： 将子类的原型指向父类实例

2 构造函数式继承：在子类的构造函数中执行父类的构造函数，并使用apply改变父类的this指向，将arguments传递

3 组合式继承： 类式继承 + 构造函数式继承

4 寄生式继承：定义一个寄生函数，接收两个参数，分别是父类和子类，定义一个新的构造函数，将构造函数的原型指向父类的原型，将构造函数的实例交给子类，补回子类的constructor属性，返回子类

5 寄生组合式继承： 定义类的时候使用构造函数式继承

安全工厂： 其实就是安全类

闭包类： 简单来说就是将一个类放入闭包中

单例模式： 单个实例模式

普通单例： 在定义的时候就执行的单例

惰性单例： 在第一次调用的时候执行的单例

静态变量：一旦定义将无法改变的变量

# 观察者模式

观察者模式又叫做消息管道、消失机制、订阅-发布、自定义事件，主要用于跨模块传递信息

## 1.1 基础观察者：

|  |
| --- |
| 1. // 定义对象 2. var Observer = (function() { 3. // 观察者对象 4. var ob = { 5. // a: function() {}, 6. // b: function() {} 7. } 8. // 定义接口 9. return { 10. /\*\* 11. \* on方法用于向观察者对象中添加事件 12. \* @type 要添加的事件名称 13. \* @fn 要添加的事件函数 14. \*\*\*/ 15. on: function(type, fn) { 16. ob[type] = fn; 17. // ob.a = function() {} 18. }, 19. /\*\* 20. \* trigger方法用于触发观察者对象中的事件 21. \* @type 要触发的事件名称 22. \*\*/ 23. trigger: function(type, val) { 24. // console.log(val); 25. ob[type](val); 26. // ob.a(); 27. }, 28. check: function() { 29. console.log(ob); 30. } 31. } 32. })() |

## 1.2 DOM二级观察者模式：

|  |
| --- |
| 1. // 升级观察者 2. var Observer = (function() { 3. // 定义真正的观察者 4. var ob = { 5. // aaa: [function() {console.log("ccc")}]; 6. } 7. // 定义接口 8. return { 9. on: function(type, fn) { 10. // 判断当前事件属性是否存在，直接push即可 11. if (ob[type]) { 12. ob[type].push(fn); 13. } else { 14. ob[type] = [fn]; 15. // ob.type = [f] 16. } 17. }, 18. // trigger用于触发观察者对象中的事件 19. trigger: function(type, val) { 20. for (var i = 0; i < ob[type].length; i++) { 21. ob[type][i](val); 22. } 23. }, 24. check: function() { 25. console.log(ob); 26. } 27. } 28. })() |

## 1.3 移除事件

|  |
| --- |
| 1. /\*\* 2. \* off方法用于取消事件 3. \* @type 要取消的事件名称 4. \* @fn 要取消的事件函数 5. \*/ 6. off: function(type, fn) { 7. // 判断是要删除一组， 还是要删除某个事件函数 8. if (fn) { 9. // 说明是要删除某个函数 10. for (var i = 0; i < ob[type].length; i++) { 11. if (ob[type][i] === fn) { 12. // 移除当前项 13. ob[type].splice(i, 1); 14. } 15. } 16. } else if (type) { 17. // 说明要删除一组事件函数 18. ob[type] = []; 19. } else { 20. // 清空所有 21. ob = {}; 22. } 23. } |

## 1.4 绑定只触发一次的事件方法

|  |
| --- |
| 1. once: function(type, fn) { 2. // 绑定的事件， 只会触发一次， 第二次将不会触发 3. // 大家还记得bind方法吗？ 4. // bind方法， 看上去好像在定义的时候就改变了this指向， 但是返回的函数和绑定的函数跟本不是同一个 5. // 这里也是同样的道理 6. // 定义一个新的函数， 绑定该函数， 在该函数中执行传递进来的fn函数， 当函数执行完毕之后再移除 7. function aaa() { 8. // 执行fn 9. fn(); 10. // 移除 11. Observer.off(type, aaa); 12. } 13. Observer.on(type, aaa); 14. } 15. } |

# 策略模式

策略模式定义： 将一组一组的算法封装起来，使之可以相互替换

## 2.1 jquery中的策略模式

|  |
| --- |
| 1. $("#box").animate({"left": 1000}, 1000, "easeInOutBounce", function() {}) 2. $("#box1").animate({"left": 1000}, 1000, "easeInOutElastic", function() {}) 3. $("#box2").animate({"left": 1000}, 1000, "easeInOutBack", function() {}) |

## 2.1 登录策略

|  |
| --- |
| 1. // 获取元素 2. var inp = document.getElementById("inp"); 3. // 失去焦点的时候要验证内容 4. inp.onblur = function() { 5. var result = Strategy.use("chunshuzi", this.value); 6. console.log(result); 7. } 8. // 定义对象 9. var Strategy = (function() { 10. // 定义存储对象 11. var s = { 12. "chunyingwen": function(str) { 13. // 定义正则表达式 14. var reg = /^[a-zA-Z]+$/g; 15. // 验证内容 16. if (reg.test(str)) { 17. return "恭喜你，通过了"; 18. } else { 19. return "请输入纯英文"; 20. } 21. }, 22. "chunshuzi": function(str) { 23. // 定义正则表达式 24. var reg = /^\d+$/g; 25. // 验证内容 26. if (reg.test(str)) { 27. return "恭喜你，通过了"; 28. } else { 29. return "请输入纯数字"; 30. } 31. } 32. } 33. // 定义接口 34. return { 35. use: function(type, str) { 36. return s[type](str); 37. } 38. } 39. })() |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |