爱创课堂前端培训

# 设计模式第二天

第2天课堂笔记（本课程共3天）

班级：北京前端训练营19期

讲师：彭帅伟

日期：2019年1月8日

彭帅伟老师

QQ: 284337908

爱创课堂官网 ：[www.icketang.com](http://www.icketang.com)

# 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， 比如jquery、underscore，它们里面都自带了一些方法，通常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，通常这些方法之间是有联系的

架构：一套大型项目的设计思路

简单工厂：就是创建对象的函数

寄生增强工厂：定义一个新的工厂， 在该工厂内实例化其它工厂， 并对该实例化对象进行增强

工厂方法：定义一个工厂在该工厂内可以返回多个实例化对象

原型：是每一个函数都拥有的一个属性，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

作用：实例共享方法和属性

继承：

1 类式继承也叫作原型式继承： 将子类的原型指向父类的实例

2 构造函数式继承： 在子类构造函数中执行父类的构造函数并使用apply改变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， 在函数中定义一个新的构造函数， 将该构造函数的原型指向父类，将该构造函数的实例化对象交给子类， 补回子类的constructor属性，最终返回子类

5 寄生组合式继承：在定义类的时候，使用构造函数式继承

安全工厂：无论外部如何调用， 都会返回该类的实例化对象

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：在第一次执行的时候会执行的单例

# 观察者模式

观察者模式又称消息管道、 订阅-发布、消息机制、 自定义事件，主要用于跨模块传递信息

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

## 1.2 DOM2级观察者模式

|  |
| --- |
| 1. var Observer = (function() { 2. // 定义真正的观察者对象 3. var ob = { 4. // a: [function() {}, function() {}] 5. }; 6. // 返回接口 7. return { 8. /\*\* 9. \* on方法，用于向观察者对象添加方法 10. \* @type 添加的事件名称 11. \* @fn 添加的事件函数 12. \*\*/ 13. on: function(type, fn) { 14. // 判断当前事件属性是否是数组， 如果是数组，直接push即可 15. if (ob[type]) { 16. ob[type].push(fn); 17. } else { 18. // 说明不是数组， 变为数组 19. ob[type] = [fn]; 20. } 21. }, 22. /\*\* 23. \* triggle方法, 用于触发观察者对象中的方法 24. \* @type 要触发的事件名称 25. \*\*/ 26. trigger: function(type, val) { 27. for (var i = 0; i < ob[type].length; i++) { 28. ob[type][i](val); 29. } 31. } 32. } 33. })(); |

## 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

策略模式的定义： 将一组一组的算法封装起来， 使之可以相互替换

如果if else语句过多， 每一个分支中还有if else, 那么程序将会显得很臃肿

## 2.1 jquery中的策略模式

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

## 2.2 登录策略

|  |
| --- |
| 1. // 定义对象 2. var Strategy = (function() { 3. var s = { 4. "chunyingwen": function(str) { 5. // 定义正则表达式 6. var reg = /^[a-zA-Z]+$/g; 7. // 验证 8. if (reg.test(str)) { 9. return "通过"; 10. } else { 11. return "请输入纯英文" 12. } 13. }, 14. "chunshuzi": function(num) { 15. // 定义正则匹配 16. var reg = /^\d+$/g; 17. // 验证 18. if (reg.test(num)) { 19. return "通过"; 20. } else { 21. return "请输入纯数字" 22. } 23. } 24. } 25. // 定义接口 26. return { 27. use: function(type, str) { 28. return s[type](str); 29. } 30. } 31. })(); |

|  |
| --- |
|  |

# 三、委托模式

定义： 将原本子元素执行的事情，交给父元素去执行。从而能够减少事件数量、预言未来元素、防止内存泄露。

## 3.1 减少事件数量

原本，应该是每一个元素拥有一个事件。使用了委托模式之后，子元素没有事件了，只有父元素有该类型事件。

## 3.2 预言未来元素

后来增加的元素没有事件，因为在新增的时候没有给该元素添加事件。 使用委托模式之后，不考虑这个事情了。

## 3.3 防止内存泄露

如果在删除元素的时候，该元素也拥有事件，那么在某些浏览器中，可能会造成内存泄露的情况。 我们现在将事件委托给父元素了，子元素本身没有事件了，所以根本泄露不了。

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |