# 设计模式，MVC，MVVM

第2天课堂笔记（本课程共4天）

[设计模式，MVC，MVVM 1](#_Toc23813)

[复习： 3](#_Toc23223)

[一 适配器模式 4](#_Toc14083)

[二 组合模式 4](#_Toc22118)

[三 观察者模式 5](#_Toc32275)

[四 策略模式 6](#_Toc19836)

[作业： 7](#_Toc17442)

# 复习：

简单工厂模式：创建单一产品实例对象的模式

寄生增强型：在工厂内部对实例对象拓展，来增强实例对象方法或属性

安全型：防止直接调动该类的构造函数时，不能正确的创建对象。

工厂方法模式：创建多种产品实例对象的模式

补充继承：

构造函数式继承

原型式继承

组合式继承

原型式继承和构造函数式的继承都执行了父类构造函数，但是是有区别的。

1 构造函数式继承，子类实例化时，得到的属性或者方法，是这些子类实例化对象私有的

2 原型式继承，是对子类的原型添加属性或者方法，它添加到子类的圆形上，但在添加属性或者方法时候是有区别的，

1. 父类的构造函数中的属性或者方法添加子类的原型上
2. 父类原型上的属性或者方法，没有被添加到子类的原型上，而是被子类的原型引用

3 原型式继承，在子类的原型上实例化父类对象是，只实例化一次，所以子类实例化时候他的原型就不能被父类更改，但是构造函数式继承可以

单例模式：只能被实例化一次。

静态变量，通过闭包隔离作用域，只能添加之类型的数据。

惰性单例：在调用时候才被实例化，而且实例化一次

适配器：将一种对象或者的类的接口（属性或者方法）适配到另外一种对象或者类的接口，以满足需求。

原型模式：实现原理是javascript的原型链，实现是javascript中的继承

# 一 适配器模式

定义：将一个类（对象）的接口（属性或者方法）转化成另一个类（对象）的接口，以满足用户的需要，使类（对象）之间接口的不兼容问题得以解决。

1 请求数据适配

|  |
| --- |
| 1. function dataAdaptor (data) { 2. var arr = []; 3. // 适配映射表 4. /\*\* 5. \* key data数据属性 6. \* value arr数据索引 7. \*/ 8. var map = { 9. src: 1, 10. href: 0, 11. title: 2 12. } 13. for (var i in map) { 14. arr[map[i]] = data[i]; 15. } 16. // arr[0] = data.href; 17. // arr[1] = data.src; 18. // arr[2] = data.title 19. return arr; 20. } |

1，是结构型设计模式

2，是对被适配的数据的一个分解再封装的一个过程

3 这个过程中会造成一定的开销。但远比更改原有业务逻辑成本低。

2 参数适配

# 二 组合模式

定义：又叫部分整体模式，将对象表示成树形结构，表示部分整体关系。所以部分与整体的行为习惯达到一致型。

创建类的步骤

一 构造函数

1 构造函数继承

2 保留参数

3 初始化数据

二 原型式继承，继承基类方法

三 重写init方法

四 添加其他方法

组合模式也是一种结构型

它是将整体分解成为一个个部分，再有部分重新拼装成一个整体。部分分解的也多，整合结果也就越多。

它的部分与整体之间具有行为的一致性。

部分拼装成整体的过程具有多样性

# 三 观察者模式

定义，又叫消息系统，消息机制，或者发布订阅额模式。通过消息系统实现对象或者类之间的解耦。

|  |
| --- |
| 1. // 消息系统 2. var Observer = (function () { 3. // 消息系统内定一个消息管道，将用户订阅的消息保存起来 4. var \_\_message = {}; 5. return { 6. /\*\* 7. \* 订阅消息的方法 8. \* @type： 表消息名称 9. \* @fn： 消息的回调函数 10. \*/ 11. regist: function (type, fn) { 12. // 如果消息管道中已经订阅果这类消息 13. if (\_\_message[type]) { 14. // 向消息管道中推入消息 15. \_\_message[type].push(fn) 16. // 如果消息管道中没有订阅过这则消息 17. } else { 18. // 向消息管道中添加消息，放在数组中的原因是，方便后面继续添加消息 19. \_\_message[type] = [fn] 20. } 21. return this; 22. }, 23. /\*\* 24. \* 触发已经订阅过的消息 25. \* @type： 表示消息类型 26. \* @data: 表示消息传递的参数 27. \*/ 28. fire: function (type, data) { 29. // 首先判断消息是否存在 30. if (\_\_message[type]) { 31. // 重组传递的数据，将消息的类别传递进来 32. var e = { 33. type: type, 34. data: data 35. } 36. // 如果存在，就要遍历消息管道中每个回调函数并且执行 37. for (var i = 0; i < \_\_message[type].length; i++) { 38. // 执行每个回调函数 39. \_\_message[type][i](e) 40. } 41. } 42. return this; 43. }, 44. /\*\* 45. \* 取消消息订阅 46. \* @type: 消息名称 47. \* @fn: 消息的回调函数的名 48. \*/ 49. remove: function (type, fn) { 50. // 要想取消某个消息，必须判断该消息存在 51. if (\_\_message[type]) { 52. // 遍历我们消息消息系统。看有没有该回调函数 53. for (var i = \_\_message[type].length - 1; i >= 0; i--) { 54. // 如果有这个回调函数 55. if (\_\_message[type][i] === fn) { 56. // 删除这个回调函数 57. \_\_message[type].splice(i, 1); 58. } 59. } 60. } 61. return this; 62. } 63. } 64. })() |

是一种行为型模式

解决是一种依赖关系，

解决了，主体与观察者之间的一种依赖关系。

被观察者对象或者类也可以是观察者，观察者也可以是被观察者

观察者内部变化不会影响到被观察者，反过来一样

# 四 策略模式

定义：封装一组算法，使其可以互相替换，这组算法本身具有独立性，不受客户端影响。

总结

1 它是行为型模式

2 每种都是独立的，所他们之间可以相互替换

3 他解决是使用者与策略算法之间的耦合

4 算是是独立的方便我们进行单测

5 算法在使用时候的过程是不一样的，但结果是一样的

# 作业：

