**《 JavaScript设计模式 》**

技术部 架构组 陈庆奇

2016.12.15

**闭包**

**----------------------------------------------------------------------------------**

闭包是封装对象的一种方式

|  |
| --- |
| /\*\*  \* 封装  \*/  function Page()  {  //计数  var \_count = 0;  /\*\*  \* 设置计数  \* @n:int  \*/  function count(n)  {  \_count = \_count+n;  return \_count;  }  /\*\*  \* 公开接口  \*/  return{  count:count  }  }  /\*\*  \* 闭包方式  \*/  var pageA = Page()  console.log(pageA.count(1)) //输出：1  console.log(pageA.count(1)) //输出：2  /\*\*  \* new方式  \*/  var pageB = new Page()  console.log(pageB.count(1)) //输出：1  console.log(pageB.count(1)) //输出：2 |

**递归**

**----------------------------------------------------------------------------------**

函数调用自身的一种编程方法

|  |
| --- |
| /\*\*  \* 递归  \*/  function print(x)  {  if(x>0)  {  document.write(x+'<hr/>');  print(x-1);  }  }  print(10); |

**反射**

**----------------------------------------------------------------------------------**

增加程序的灵活性,避免将程序写死到代码里

|  |
| --- |
| /\*\*  \* 获取window的属性和方法  \*/  for(var i in window)  {  console.log(i+':'+window[i])  }  /////////////////////////////////////////////////////////////////////  /\*\*  \* 根据键盘值，执行相应方法  \*/  window.addEventListener('keydown',function(e)  {  keyEvent[e.keyCode]()  })  /\*\*  \* 键盘事件  \*/  var keyEvent = {  65:function()  {  alert('A')  },  83:function()  {  alert('S')  },  68:function()  {  alert('D')  }  }  /\*\*  \* 根据数据，更新元素样式  \*/  function style(data)  {  for(var i in data)  {  document.body.style[i] = data[i];  }  }  style({'background-color':'red'});  /////////////////////////////////////////////////////////////////////  /\*\*  \* switch耦合  \*/  switch(num)  {  case 0 :  console.log('跑步')  break;  case 1 :  console.log('跳跃')  break;  case 2 :  console.log('睡觉')  break;  }  /\*\*  \* 反射解耦  \*/  function people()  {  function run()  {  console.log('跑步')  }  function jump()  {  console.log('跳跃')  }  function sleep()  {  console.log('睡觉')  }  return{  0:run,  1:jump,  2:sleep  }  }  new people()[value] |

**单例**

**----------------------------------------------------------------------------------**

只能在全局实例化一次的对象

|  |
| --- |
| /\*\*  \* 全局变量，其实就是单例  \*/  window.count = 0;  function index()  {  count +=1  function content()  {  count +=1  }  }  /\*\*  \* 标准实现单例  \*/  function Singleton()  {  this.count = 0;  if(Singleton.interface !== undefined)  {  return Singleton.interface;  }  //静态属性  Singleton.interface = this;  }  var s1 = new Singleton();  var s2 = new Singleton();  console.log(s1 == s2) //输出：true |

**原型**

**----------------------------------------------------------------------------------**

实例可以被克隆（Javascript的继承就是基于原型的）

|  |
| --- |
| function People()  {  /\*\*  \* 钱  \*/  var money = 0;  /\*\*  \* 克隆  \*/  function clone()  {  function Copy(){}  Copy.prototype = this;  return new Copy();  }  return{  money:money,  clone:clone  }  }  var xiaoming = new People();  xiaoming.money = 1000;  var zhansan = xiaoming.clone();  console.log(zhansan.money) //输出:'1000' |

**适配器**

**----------------------------------------------------------------------------------**

兼容的不同系统的接口

|  |
| --- |
| /\*\*  \* 老系统  \*/  var AppUser = {  name:'app'  }  /\*\*  \* 实现登录  \*/  function login(obj)  {  console.log(obj.name)  }  login(AppUser) //输出：app；  /\*\*  \* 需要兼容的新增系统  \*/  var WebUser = {  username:'网站'  }  var WeixinUser = {  account :'微信'  }  /\*\*  \* 适配器  \*/  function Adapter(obj)  {  var name;  if(obj.name !== "undefined")  {  name = obj.name;  }  if(obj.username !== "undefined")  {  name = obj.username;  }  if(obj.account !== "undefined")  {  name = obj.account;  }  return{  name:name  }  }  /\*\*  \* 应用  \*/  login(new Adapter(WeixinUser)); |

**桥接**

**----------------------------------------------------------------------------------**

将对象的多个维度，连接合并在一起

|  |
| --- |
| /\*\*  \* 笔触  \*/  function Pen(color,shape)  {  /\*\*  \* 铅笔  \*/  this.pencil = function()  {  shape.draw(color.value);  console.log('应用铅笔滤镜');  }  /\*\*  \* 粉笔  \*/  this.chalk = function()  {  shape.draw(color.value);  console.log('应用粉笔滤镜');  }  }  /\*\*  \* 颜色  \*/  function Color()  {  this.value = '红色'  }  /\*\*  \* 图形  \*/  function Shape()  {  this.draw = function(color)  {  console.log('绘制'+color+'图形')  }  }  /\*\*  \* 应用  \*/  var pen = new Pen(new Color(),new Shape())  pen.pencil(); //输出：绘制红色图形，应用铅笔滤镜 |

**组合**

**----------------------------------------------------------------------------------**

构建一个树状结构

|  |
| --- |
| /\*\*  \* 树  \*/  function Tree()  {  this.children = [];  /\*\*  \* 添加节点  \*/  this.add = function(v)  {  this.children.push(v)  }  /\*\*  \* 删除节点  \*/  this.del = function(v)  {  this.children.splice(v,1)  }  /\*\*  \* 查找节点  \*/  this.has = function(v)  {  for(var i=0;i<this.children.length;i++)  {  if(v == this.children[i])  {  return true;  }  }  return false;  }  }  /\*\*  \* 人  \*/  function People(j)  {  this.job = j;  }  /\*\*  \* 应用  \*/  var director = new Tree(new People('董事长'));  director.add(new People('COO'));  director.add(new People('CTO'));  console.log(director.children[0].job) // 输出：COO |

**工厂**

**----------------------------------------------------------------------------------**

用一个类来创建其他类实例

|  |
| --- |
| /\*\*  \* 工厂模式  \*/  function ShapeFactory(name)  {  var shape;  switch(name)  {  case 'rect':  shape = new Shape()  break  case 'circle':  shape = new Circle()  break  case 'line':  shape = new Line()  break  case 'arrow':  shape = new Arrow()  break  }  return shape;  }  /\*\*  \* 应用  \*/  var rect = new ShapeFactory('rect'); |

**抽象工厂**

**----------------------------------------------------------------------------------**

用一个类来创建工厂或实例

|  |
| --- |
| /\*\*  \* 矢量图工厂  \*/  function ShapeFactory(name)  {  var shape;  switch(name)  {  case 'rect':  shape = new Rect()  break  case 'circle':  shape = new Circle()  break  case 'line':  shape = new Line()  break  case 'arrow':  shape = new Arrow()  break  }  return shape;  }  /\*\*  \* 位图工厂  \*/  function BitmapFactory(name)  {  var bitmap;  switch(name)  {  case 'rect':  bitmap = new Rect()  break  case 'circle':  bitmap = new Circle()  break  case 'line':  bitmap = new Line()  break  case 'arrow':  bitmap = new Arrow()  break  }  return shape;  }  /\*\*  \* 抽象工厂  \*/  function DrawFactory(name,type)  {  var factory;  switch(name)  {  case 'shape':  factory = new ShapeFactory(type)  break  case 'bitmap':  factory = new BitmapFactory(type)  break  }  return shape;  }  /\*\*  \* 应用  \*/  var rect = new DrawFactory('shape','rect') |

**生成器**

**----------------------------------------------------------------------------------**

实例化一组特定对象

|  |
| --- |
| /\*\*  \* 套餐一  \*/  function Builder()  {  /\*\*  \* 产品  \*/  var product = [];  /\*\*  \* 组装  \*/  function getProduct()  {  product.push('汉堡');  product.push('薯条');  product.push('可乐');  return product;  }  return{  getProduct:getProduct  }  }  var builder = new Builder();  console.log(builder.getProduct()) |

**状态机**

**----------------------------------------------------------------------------------**

不同状态下的相同操作

|  |
| --- |
| /\*\*  \* 灯  \*/  function Light()  {  var \_state = new StateOFF();  /\*\*  \* 更改状态  \*/  function state(v)  {  \_state = v;  }  /\*\*  \* 开  \*/  function open()  {  \_state.open(this)  }  /\*\*  \* 关  \*/  function close()  {  \_state.close(this)  }  return{  state:state,  open:open,  close:close  }  }  /\*\*  \* 灯开着的状态  \*/  function StateON()  {  function open(light)  {  console.log('本次操作被忽略，灯处于打开状态')  }  function close(light)  {  light.state(new StateOFF())  console.log('关灯成功')  }  return{  open:open,  close:close  }  }  /\*\*  \* 灯关着的状态  \*/  function StateOFF()  {  function open(light)  {  light.state(new StateON())  console.log('开灯成功')  }  function close(light)  {  console.log('本次操作被忽略，灯处于关闭状态')  }  return{  open:open,  close:close  }  }  /\*\*  \* 测试应用  \*/  var light = new Light();  /\*\*  \* 侦听用户交互  \*/  window.addEventListener('click',function(e)  {  if(e.target.id == 'on')  {  light.open()  }  else  {  light.close()  }  }) |

|  |
| --- |
| /\*\*  \* 播放器  \*/  function Player(video)  {  /\*\*  \* 默认状态  \*/  var \_state = new StateStop();  /\*\*  \* 切换状态  \*/  function state(v)  {  \_state = v;  }  /\*\*  \* 播放  \*/  function play()  {  \_state.play(this)  }  /\*\*  \* 暂停  \*/  function pause()  {  \_state.pause(this)  }  /\*\*  \* 停止  \*/  function stop()  {  \_state.stop(this)  }  return{  video:video,  state:state,  play:play,  pause:pause,  stop:stop  }  }  /\*\*  \* 停止状态  \*/  function StateStop()  {  /\*\*  \* 播放  \*/  function play(p)  {  console.log('切换到播放状态')  p.state(new StatePlay());  p.video.play();  }  /\*\*  \* 暂停  \*/  function pause(p)  {  console.log('此操作被忽略,停止中不可暂停')  }  /\*\*  \* 停止  \*/  function stop(p)  {  console.log('此操作被忽略,停止中不可再次停止')  }  return{  play:play,  pause:pause,  stop:stop  }  }  /\*\*  \* 播放状态  \*/  function StatePlay()  {  /\*\*  \* 播放  \*/  function play(p)  {  console.log('此操作被忽略,播放中不可再次播放')  }  /\*\*  \* 暂停  \*/  function pause(p)  {  console.log('切换到暂停状态')  p.state(new StatePause());  p.video.pause();  }  /\*\*  \* 停止  \*/  function stop(p)  {  console.log('切换到停止状态')  p.state(new StateStop());  //p.video.stop();  }  return{  play:play,  pause:pause,  stop:stop  }  }  /\*\*  \* 暂停状态  \*/  function StatePause()  {  /\*\*  \* 播放  \*/  function play(p)  {  console.log('切换到播放状态')  p.state(new StatePlay());  p.video.play();  }  /\*\*  \* 暂停  \*/  function pause(p)  {  console.log('此操作被忽略,暂停中不可再次暂停')  }  /\*\*  \* 停止  \*/  function stop(p)  {  console.log('切换到停止状态')  p.state(new StateStop());  //p.video.stop();  }  return{  play:play,  pause:pause,  stop:stop  }  }  /\*\*  \* 测试播放状态机  \*/  var player = new Player(document.getElementsByTagName('video')[0]);  window.addEventListener('click',function(e)  {  if(e.target.id == 'play')  {  player.play()  }  if(e.target.id == 'pause')  {  player.pause()  }  if(e.target.id == 'stop')  {  player.stop()  }  }) |

**模板**

**----------------------------------------------------------------------------------**

把做事情的步骤抽出来

|  |
| --- |
| /\*\*  \* 冲泡饮料的一般步骤  \*/  function Do()  {  /\*\*  \* 开始  \*/  this.start = function()  {  this.boilWater()  this.brew()  }  /\*\*  \* 烧水  \*/  this.boilWater = function()  {  console.log('烧水')  }  /\*\*  \* 冲泡  \*/  this.brew = function()  {  console.log('此方法须重写')  }  }  /\*\*  \* 泡茶  \*/  function BrewTea()  {  Do.call(this)  this.brew = function()  {  console.log('泡茶')  }  }  /\*\*  \* 泡咖啡  \*/  function BrewCoffee()  {  Do.call(this)  this.brew = function()  {  console.log('泡咖啡')  }  }  /\*\*  \* 运用  \*/  new BrewCoffee().start() |

**职责链**

**----------------------------------------------------------------------------------**

请求在链上传递，直到某个对象处理(申请费用流程)

|  |
| --- |
| function Handler()  {  /\*\*  \* 后续领导  \*/  var \_successor;  /\*\*  \* 处理请求  \*/  function handleRequest(name,fee)  {  if(fee<1000)  {  if(name == '张三')  {  console.log('可以批准')  }  else  {  console.log('一律拒绝')  }  }  else  {  if(\_successor)  {  console.log('交给领导处理')  \_successor.handleRequest()  }  else  {  console.log('一律拒绝')  }  }  }  /\*\*  \* 设置或获取后续对象  \*/  function successor(v)  {  if(!v)  {  return \_successor;  }  \_successor = v;  }  return{  handleRequest:handleRequest,  successor:successor  }  }  var handlerA = new Handler();  var handlerB = new Handler();  handlerA.successor(handlerB)  handlerA.handleRequest('张三',800) |

**享元**

**----------------------------------------------------------------------------------**

重用已被创建的对象

|  |
| --- |
| /\*\*  \* 棋子  \*/  function Chess(color)  {  //颜色  this.color = color;  /\*\*  \* 初始  \*/  function init()  {  console.log('创建'+color+'棋子')  }  //坐标  this.point = function(x,y)  {  console.log(x+':'+y)  }  }  /\*\*  \* 对象池  \*/  function Factory(){}  Factory.objects = {};  Factory.get = function(color)  {  var obj = Factory.objects[color];  if(obj == null)  {  obj = new Chess(color);  Factory.objects[color] = obj;  }  return obj;  }  /\*\*  \* 使用享元模式  \*/  var c1 = Factory.get('红色')  var c2 = Factory.get('红色')  var c3 = Factory.get('黑色')  console.log(c1 == c2) //输出：true  console.log(Factory.objects) //输出：Object {红色: Chess, 黑色: Chess} |

**中介者**

**----------------------------------------------------------------------------------**

处理多个对象之间的交互

|  |
| --- |
| /\*\*  \* 消息中介(聊天室)  \*/  function Mediator()  {  /\*\*  \* 存储  \*/  var storage = {};  /\*\*  \* 注册  \*/  this.register = function(object)  {  storage[object.name] = object;  object.mediator = this;  }  /\*\*  \* 发送  \*/  this.send = function(data,form,to)  {  if(to)  {  to.receive(data,form)  }  else  {  for(var i in storage)  {  if(storage[i] !== form)  {  storage[i].receive(data,form)  }  }  }  }  }  /\*\*  \* 人  \*/  function Person(name)  {  /\*\*  \* 名字  \*/  this.name = name;  /\*\*  \* 中介者  \*/  this.mediator;  /\*\*  \* 发送  \*/  this.send = function(data,to)  {  this.mediator.send(data,this,to)  }  /\*\*  \* 接收  \*/  this.receive = function(data,form)  {  console.log(this.name+'收到消息-----------'+form.name+'说：'+data)  }  }  var xiaoming = new Person('小明');  var zhangsan = new Person('张三');  var lisi = new Person('李四');  var mediator = new Mediator();  mediator.register(xiaoming);  mediator.register(zhangsan);  mediator.register(lisi);  xiaoming.send('大家好') //广播消息  xiaoming.send('咱俩私聊',lisi) //私聊消息 |

|  |
| --- |
| /\*\*  \* 方法中介(MVC)  \*/  function Mediator(view,model)  {  /\*\*  \* 初始化  \*/  function init()  {  view.setTitle(model.title);  }  init();  }  /\*\*  \* 视图  \*/  function View()  {  /\*\*  \* 修改标题  \*/  this.setTitle = function(data)  {  console.log(data)  }  }  /\*\*  \* 数据  \*/  function Model()  {  /\*\*  \* 标题数据  \*/  this.title = '暂无标题';  }  var mediator = new Mediator(new View(),new Model()); |

**观察者**

**----------------------------------------------------------------------------------**

订阅消息并接受推送

|  |
| --- |
| /\*\*  \* 用户  \*/  function User(name)  {  /\*\*  \* 名称  \*/  this.name = name;  /\*\*  \* 更新  \*/  this.updata = function(v)  {  console.log(this.name+':'+ v);  }  }  /\*\*  \* 微信  \*/  function Weixin()  {  var users = {};  /\*\*  \* 增加订阅者  \*/  this.attach = function(obj)  {  users[obj.name] = obj;  }  /\*\*  \* 删除订阅者  \*/  this.detach = function(obj)  {  users[obj.name] = null;  }  /\*\*  \* 更新消息  \*/  this.notify = function(obj)  {  for(var i in users)  {  users[i].updata(obj);  }  }  }  /\*\*  \* 测试  \*/  var weixin = new Weixin();  var user1 = new User('小明');  var user2 = new User('张三');  var user3 = new User('丽丽');  weixin.attach(user1);  weixin.attach(user2);  weixin.attach(user3);  weixin.notify('你有新消息') |

**迭代器**

**----------------------------------------------------------------------------------**

遍历聚合对象

|  |
| --- |
| /\*\*  \* 迭代器  \*/  function Iterator()  {  /\*\*  \* 聚合数据  \*/  var names = ['张三','李四','王二']  /\*\*  \* 指针  \*/  var index = 0;  /\*\*  \* 向前  \*/  this.next = function()  {  return names[index++];  }  /\*\*  \* 向后  \*/  this.prev = function()  {  index++;  return names[names.length-index];  }  }  /\*\*  \* 测试  \*/  var iterator = new Iterator();  window.addEventListener('click',function()  {  console.log(iterator.next())  console.log(iterator.prev())  }) |

**命令**

**----------------------------------------------------------------------------------**

将请求封装成对象

|  |
| --- |
| /\*\*  \* 命令  \*/  function Command()  {  /\*\*  \* 宏命令集合  \*/  var objects = [];  /\*\*  \* 添加  \*/  this.add = function(obj)  {  objects.push(obj)  }  /\*\*  \* 执行  \*/  this.execute = function()  {  for(var i=0;i<objects.length;i++)  {  objects[i].execute()  }  }  }  /\*\*  \* 立正  \*/  function Lizheng()  {  this.execute = function()  {  console.log('立正')  }  }  /\*\*  \* 稍息  \*/  function Shaoxi()  {  this.execute = function()  {  console.log('稍息')  }  }  /\*\*  \* 起步走  \*/  function Zou()  {  this.execute = function()  {  console.log('起步走')  }  }  /\*\*  \* 测试  \*/  var command = new Command();  command.add(new Lizheng())  command.add(new Shaoxi())  command.add(new Zou())  command.execute() |

**策略**

**----------------------------------------------------------------------------------**

封装算法

|  |
| --- |
| /\*\*  \* 环境类  \*/  function Strategy()  {  var object;  /\*\*  \* 设置策略  \*/  this.setType = function(\_object)  {  object = \_object;  }  /\*\*  \* 行动  \*/  this.action = function()  {  object.action()  }  }  /\*\*  \* 步行  \*/  function Walk()  {  this.action = function()  {  console.log('步行');  }  }  /\*\*  \* 开汽车  \*/  function Car()  {  this.action = function()  {  console.log('开汽车');  }  }  /\*\*  \* 做火车  \*/  function Train()  {  this.action = function()  {  console.log('做火车');  }  }  /\*\*  \* 测试  \*/  var strategy = new Strategy();  //路太远，应该做火车  strategy.setType(new Train());  strategy.action()  //路太近，应该步行  strategy.setType(new Walk());  strategy.action() |

**代理**

**----------------------------------------------------------------------------------**

一个类代表另一个类的功能

|  |
| --- |
| /\*\*  \* 4S店  \*/  function Shop(factory)  {  /\*\*  \* 售卖  \*/  this.sell = function()  {  factory.sell()  }  }  /\*\*  \* 工厂  \*/  function Factory()  {  /\*\*  \* 售卖  \*/  this.sell = function()  {  console.log('宝马轿车')  }  }  /\*\*  \* 应用  \*/  var shop = new Shop(new Factory());  shop.sell() //输出：宝马轿车 |

**外观**

**----------------------------------------------------------------------------------**

为系统提供统一入口

|  |
| --- |
| /\*\*  \* 封装系统  \*/  function Facade()  {  this.register = function()  {  new Gongshang().jiancha();  new Yinghang().kaihu();  new Shuiwuju().dengji();  }  }  /\*\*  \* 工商局  \*/  function Gongshang()  {  /\*\*  \* 检查名字  \*/  this.jiancha = function()  {  console.log("检查名字是否有冲突")  }  }  /\*\*  \* 银行  \*/  function Yinghang()  {  /\*\*  \* 开户  \*/  this.kaihu = function()  {  console.log("在中国工商银行开户")  }  }  /\*\*  \* 税务局  \*/  function Shuiwuju()  {  /\*\*  \* 登记  \*/  this.dengji = function()  {  console.log("在海淀区税务局办理登记证")  }  }  /\*\*  \* 使用  \*/  new Facade().register(); |

**解析器**

**----------------------------------------------------------------------------------**

同一对象的不同解释

|  |
| --- |
| /\*\*  \* 中文解释  \*/  function Chinese()  {  this.say = function(value)  {  switch(value)  {  case 1:  return '一'  break  case 2:  return '二'  break  default:  return '暂不支持'  break  }  }  }  /\*\*  \* 英文解释  \*/  function English()  {  this.say = function(value)  {  switch(value)  {  case 1:  return 'one'  break  case 2:  return 'two'  break  default:  return 'can not explain'  break  }  }  }  var chinese = new Chinese();  var english = new English()  console.log(chinese.say(1)) //一  console.log(chinese.say(10)) //暂不支持  console.log(english.say(1)) //one |

**访问者**

**----------------------------------------------------------------------------------**

从某个角度遍历数据

|  |
| --- |
| /\*\*  \* 人员  \*/  function Person(\_name,\_work,\_money)  {  this.name = \_name;  this.work = \_work;  this.money = \_money;  }  /\*\*  \* 公司  \*/  function Company()  {  //数据  var persons = [];  /\*\*  \* 添加  \*/  this.add = function(\_person)  {  persons.push(\_person)  }  /\*\*  \* 获取  \*/  this.accept = function(visitor)  {  console.log(visitor.name+'获取数据：')  for(var i=0;i<persons.length;i++)  {  visitor.read(persons[i])  }  }  }  /\*\*  \* 财务  \*/  function Finance(\_name)  {  this.name = \_name;  this.read = function(\_person)  {  console.log(\_person.name+'-'+\_person.money);  }  }  /\*\*  \* 总监  \*/  function CTO(\_name)  {  this.name = \_name;  this.read = function(\_person)  {  console.log(this.name+': '+\_person.name+'-'+\_person.work);  }  }  /\*\*  \* 添加员工  \*/  var company = new Company();  company.add(new Person('小明','程序员','12k'));  company.add(new Person('小红','设计师','10k'));  company.add(new Person('成龙','业务员','20k'));  /\*\*  \* 财务获取员工数据  \*/  company.accept(new Finance('财务')) |

**备忘录**

**----------------------------------------------------------------------------------**

保存对象的某个状态

|  |
| --- |
| /\*\*  \* 备份模板  \*/  function Memento(p)  {  //名字  this.name = p.name;  //年龄  this.age = p.age;  //性别  this.sex = p.sex;  //财富  this.money = p.money  }  /\*\*  \* 人物  \*/  function Person()  {  //名字  this.name = '';  //年龄  this.age = 0;  //性别  this.sex = '男';  //财富  this.money = 0  /\*\*  \* 还原  \*/  this.recovery = function(p)  {  this.name = p.name;  this.age = p.age;  this.sex = p.sex;  this.money = p.money  }  /\*\*  \* 备份  \*/  this.memento = function()  {  return new Memento(this);  }  }  /\*\*  \* 历史备份  \*/  var data = [];  //定义人物  var person = new Person();  person.name = '张三';  person.money = 5000;  //备份  data.push(person.memento())  //修改人物状态  person.money = 100;  //还原财富初始值  person.recovery(data[0]);  //测试  console.log(person.money) //输出：5000 |