小程序: WEB问题速查

1. 问题清单: 历届师兄师姐在今天课程中集中问过的高频的重点的问题！

那些自己发现不了问题，又迫切想知道其它同学都问了什么问题，的同学

小程序->首页->JSCORE->day03、day04

当老师在回答其它同学的问题时，你不要闲着！你可以打开问题清单浏览别人曾经问过的经典问题！

2. 高清无码小视频: 小程序->在线->JSCORE->day03、day04

2部分: 1. 课上重点知识和案例的回放！2. 重点高频笔试面试题讲解扩展视频

万一家里或中心网络断了！暂时看不了直播了！

因为某些事儿请假了，错过了当天的直播！

课下复习时，觉得TMOOC视频太大，浏览不方便或TMOOC不能看了！

——打开小程序中"在线"，就可下载观看当前所讲内容和当天所讲全部内容的回放。

基础比较好的，确实学有余力的同学，想拿高薪的同学，在我答疑时，不要闲着:

要看今天扩展视频:

小程序->在线->JSCORE->day03

作业: 复习 高频笔试题 按值传递

作业: 闭包 高频笔试题 funs

小程序->在线->JSCORE->day04

扩展笔试题2道 函数柯里化

扩展笔试题 反复调用多次外层函数 ...

高频笔试题 9道

——通用解题方法：画图！

3. 单词列表: 小程序->单词->JSCORE->day03、day04

第三阶段上课约定:

1. 上课千万不要跟着记笔记和写代码！

2. 凡是拼写错误，自己懒得找！我可以帮你找，但是找到后，错误的单词你要手抄20遍，并拍照！

3. 每个带注释的案例至少写三遍:

(1). 注释+代码抄一遍

(2). 保留注释，删除代码，自己试着把代码翻译回来！

(3). 删除注释和代码，自己试着用自己的话把注释写回来！

一个例子只有自己可以把人话的注释写回来！才算自己会！

问题: 一听就会一做就废！原因: 你自己无法用人话精确描述自己想干什么！

如何提问:

1. 上课时，案例中，笔记中那句话哪行代码看不懂，立刻问！

2. 小程序中问题清单里哪个问题和回答看不懂！立刻问！

3. 个人项目中哪个功能不会做！立刻问！

4. 刷笔试题时，遇到不会的笔试题！立刻问！

谁问的问题越多！谁的培训费交的越值！

正课:

\*\*\*\*Function\*\*\*\*

1. 作用域和作用域链

2. \*\*\*\*\*闭包

\*\*\*\*面向对象\*\*\*\*

1. 什么是面向对象

2. 封装

3. 继承

\*\*\*\*Function\*\*\*\*

一. 作用域和作用域链:

对应小程序视频中: 小程序->在线->JSCORE->day03 7. 原理 作用域和作用域链

1. 作用域(scope):

(1). 什么是作用域:

a. 作用: 一个变量的可用范围

b. 本质: 其实作用域是内存中保存变量的**对象**

(2). 为什么: 防止不同范围之间的变量互相污染

(3). 在js中共包括2级作用域:

a. 全局作用域:

1). 什么是: 专门保存全局变量的区域/存储空间

2). 什么是全局变量: 在函数function**之外**用**var**创建的变量

3). 特点: 随处可用，可反复使用！——也是全局变量的特点！

4). 全局作用域的本质: window对象

b. 函数作用域:

1). 什么是: 专门保存**仅**函数内可用的局部变量的区域/存储空间

2). 什么是局部变量: 2种:

i. 在函数**内**用var创建的变量

ii. **形参变量**虽然没有var，但是也是局部变量

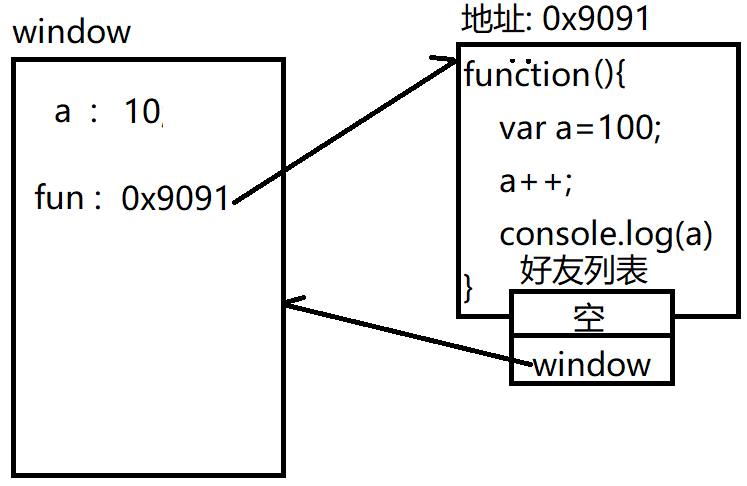
3). 特点: 仅函数内可用，不可重用！——也是局部变量的特点！

4). 函数作用域的本质: 也是一种专门保存局部变量的对象

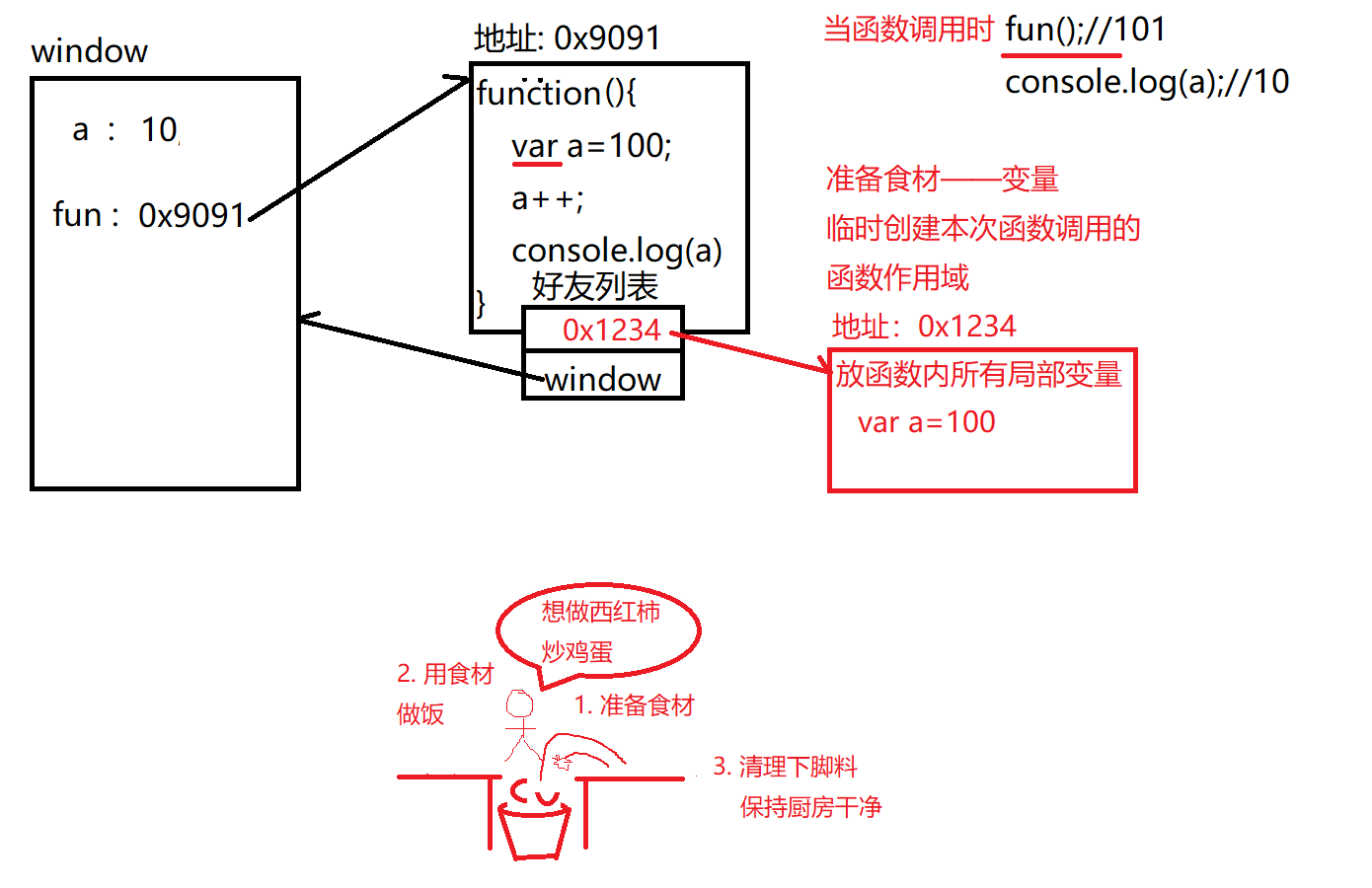
i. 每个函数在定义时，函数对象上都包含一个"好友列表"

好友列表中第一个格子暂时空

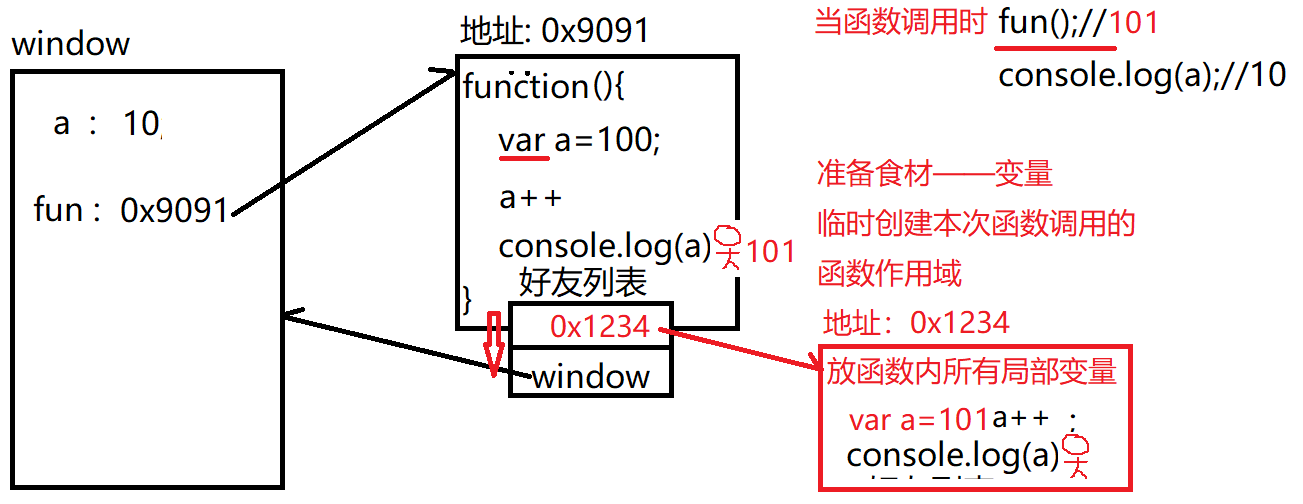
好友列表中第二个格子指向window



ii. 当**调用**函数时，会临时创建本次函数调用的函数作用域对象

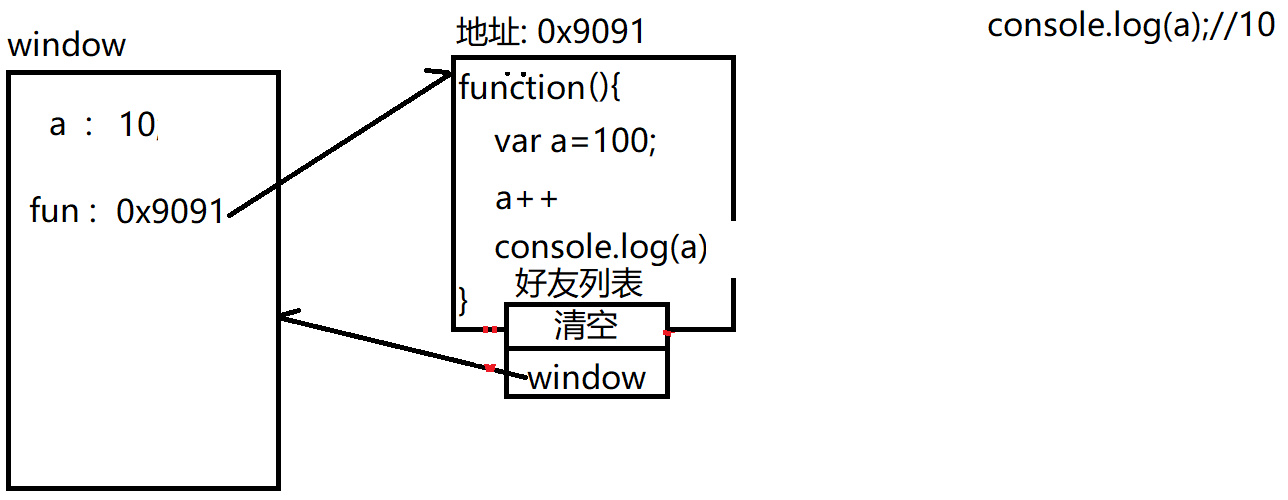


iii. 函数调用过程中: 函数总是优先使用好友列表中离自己近的函数作用域对象中的局部变量。除非离自己近的函数作用域对象中，没有需要的局部变量，才被迫去全局window中查找变量使用！——就近原则！



iv. 函数调用后，自动释放好友列表中离自己近格子中的地址，等效于释放了本次函数调用所使用的函数作用域对象。等效于将本次使用的所有局部变量一起释放了！

——所以，所有局部变量都不可重用！



c. 总结: 如果函数内自己创建了所需的局部变量，则优先使用局部变量

如果函数内自己没创建所需的局部变量，则去全局找需要的变量使用！

——先局部，后全局！

d. 示例: 判断以下三个程序的输出结果:

1\_scope.html

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <title>Document</title>  </head>  <body>    <script>      // var a=10;      // function fun(){      //   var a=100;      //   a++;      //   console.log(a)      // }      // fun();//101      // console.log(a);//10  C:\Users\web\Desktop\1.png      // var a=10;      // function fun(){      //   a=100;      //   a++;      //   console.log(a)      // }      // fun();//101      // console.log(a);//101  C:\Users\web\Desktop\2.png      var a=10;      function fun(a){        a=100;        a++;        console.log(a)      }      fun();//101      console.log(a);//10  C:\Users\web\Desktop\3.png    </script>  </body>  </html>  运行结果:  101  10 |

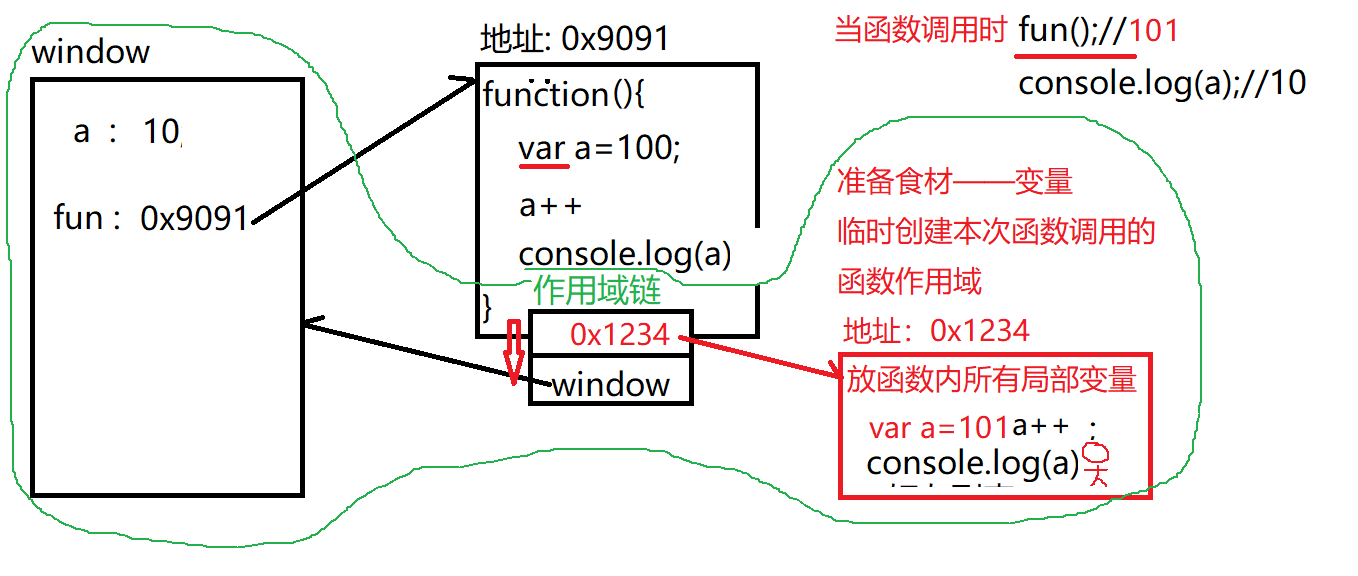
2. 作用域链（scope chain）：

(1). 什么是作用域链: 函数对象中保存函数调用时所有可用的作用域对象的链式结构

(2). 其实就是上图中的“好友列表”，学名就叫作用域链！

(3). 作用域链保存着: 一个函数可用的所有变量（局部变量和全局变量）

(4). 作用域链控制着: 变量的使用顺序（先局部后全局！）



二. \*\*\*\*\*闭包(closure):

对应小程序视频: 小程序->在线->JSCORE->day03 8. 闭包closure 压岁钱

1. 什么是闭包:

(1). 用法: 既重用一个变量，又保护变量不被篡改的一种编程方法

(2). 本质: ?

2. 为什么使用闭包: 全局变量和局部变量都有不可兼得的优缺点

(1). 全局变量: 优: 可重用，缺: 随处可以极易被污染

(2). 局部变量: 优: 仅函数内可用，不会被污染，缺: 不可重用！

3. 何时: 如果希望让一个函数即可重用一个变量，又保护这个变量不会被篡改，都用闭包！

4. 如何使用: 3步: (不要问什么)

(1). **定义一个外层函数包裹内层函数和要保护的变量**

问题: 内层函数成了外层函数内的局部函数，外部不可使用！

(2). **外层函数将内层函数返回到外部！**

说明: 内层函数不要起名！因为反正外部接住这个内层函数后，人家还会起新的变量名。

问题:**函数只有调用才执行，才能获得返回结果**

函数不调用，就不会执行，也不会获得返回结果！

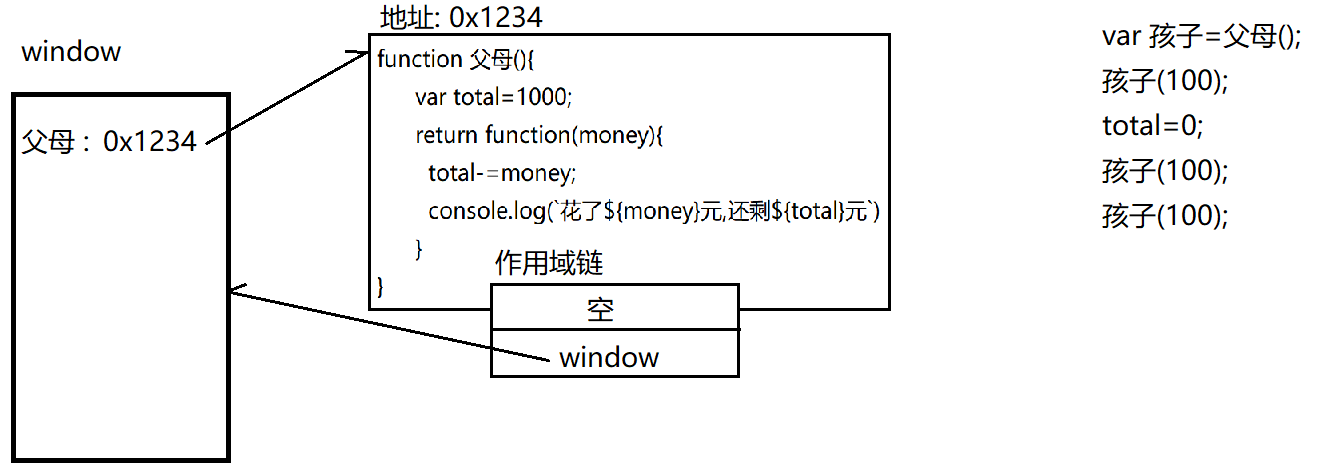
(3). **想使用内层函数的人，必须调用外层函数获得内层函数！才能使用！**

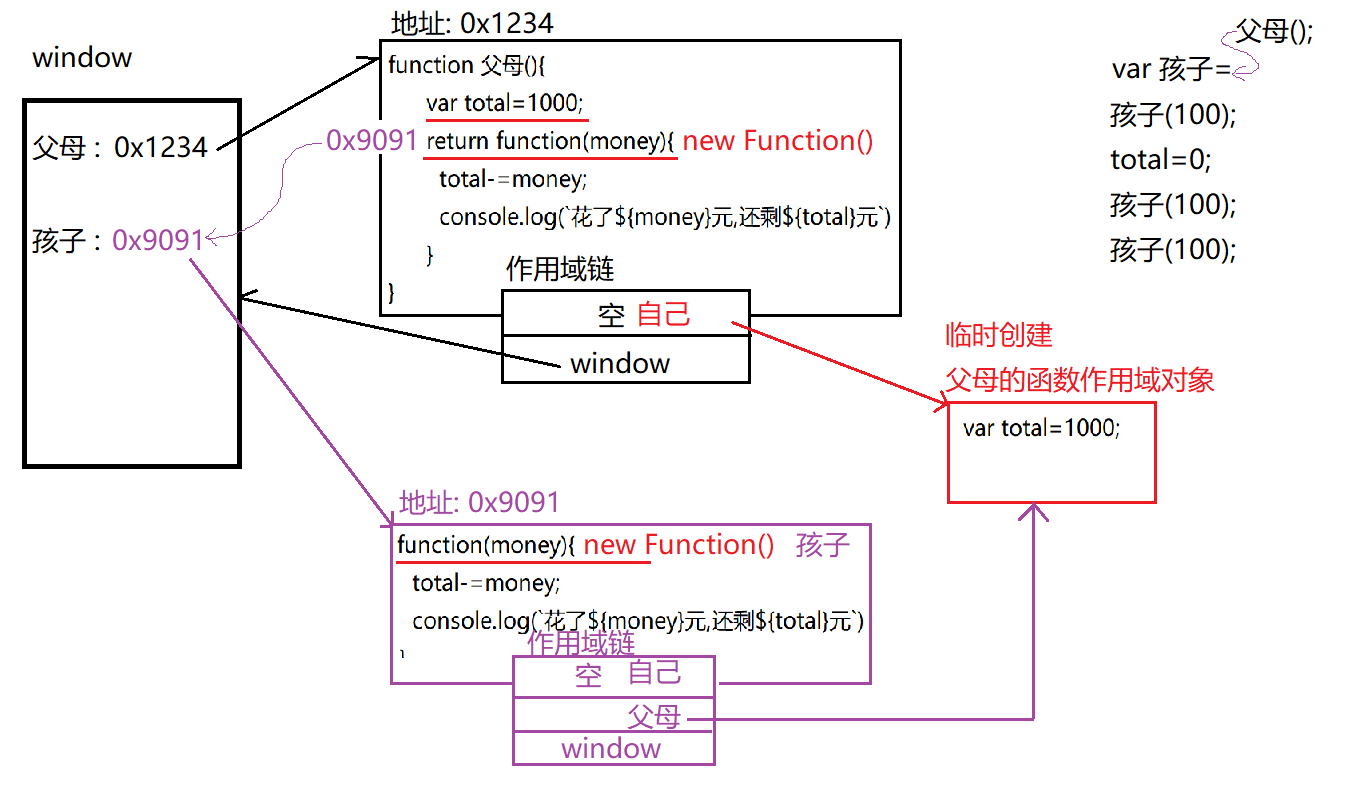
5. 示例: 使用闭包保护钱数，并重用钱数

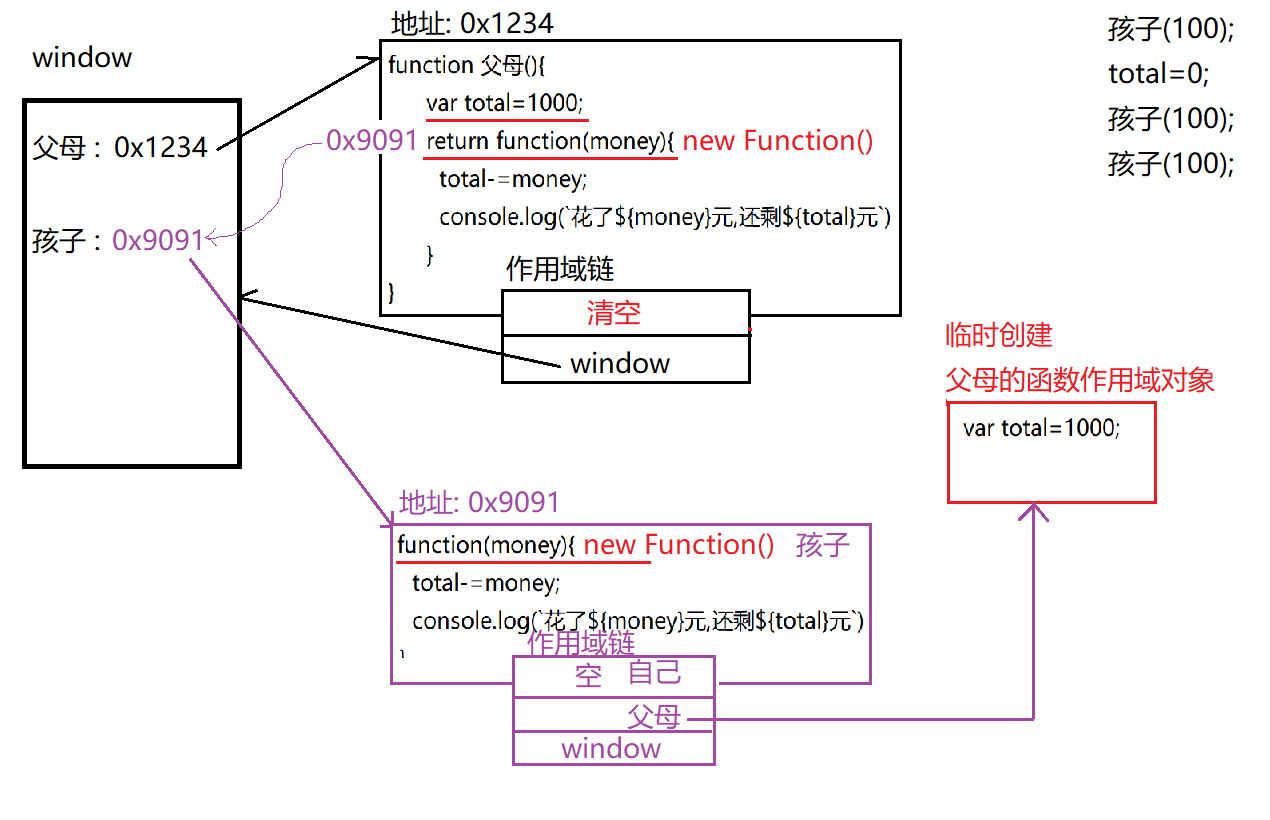
2\_closure.html

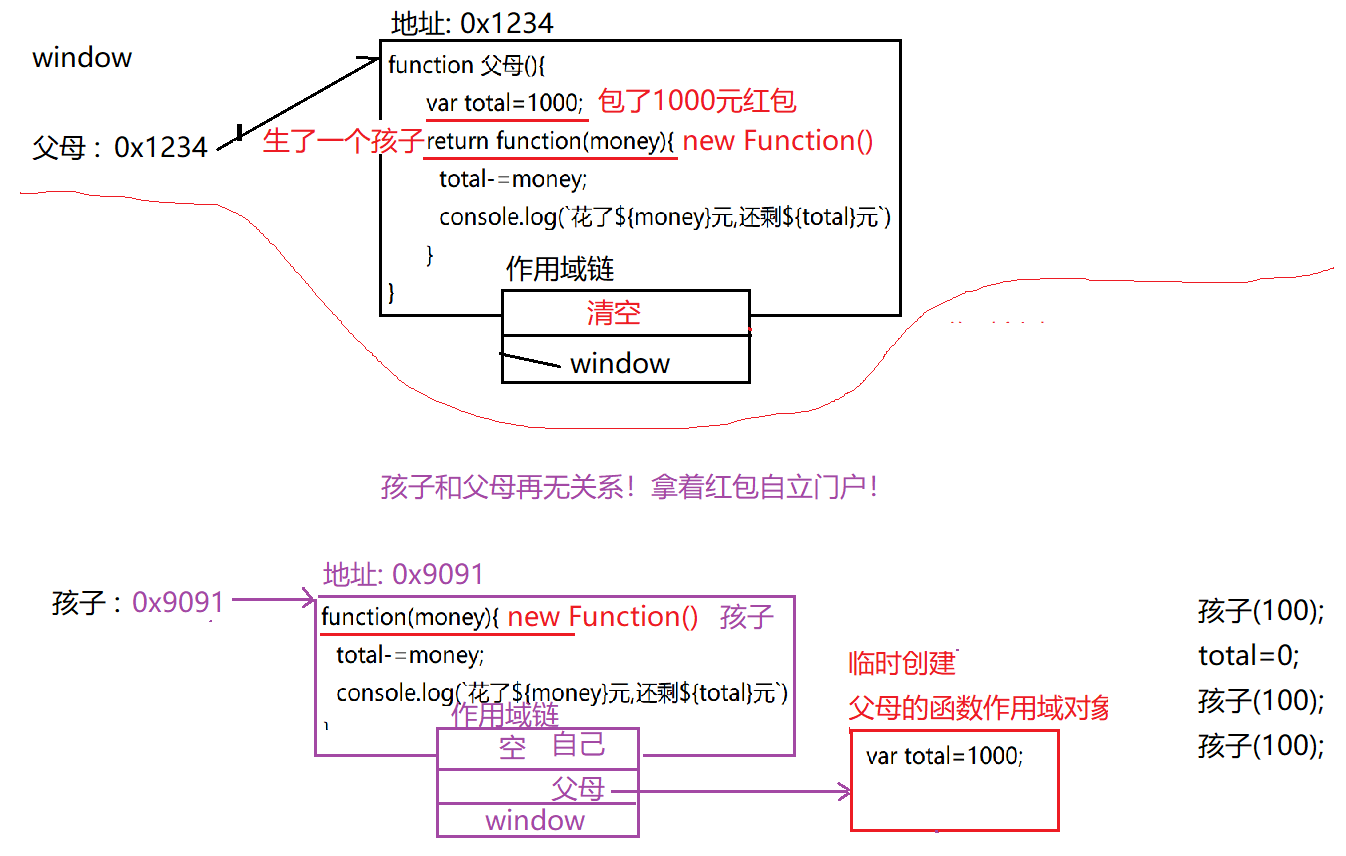
|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <title>Document</title>  </head>  <body>    <script>      //定义一个函数替小孩儿保管零花钱      //小孩儿每花一笔钱，就从总钱数中减去花的钱，提示余额      //问题: 比如有1000元总金额，保存在哪里？      //要么存在全局: 虽然可以实现功能，但是极易被篡改！      //闭包3步:      //1. 定义外层函数包裹要保护的变量和内层函数        function 父母(){        //在外层函数父母内定义要保护的变量        var total=1000;        //2. 外层函数将内层函数返回到外部        //生出  孩子函数  **return** function(money){          //要么存在局部: 虽然不会被篡改，但是不可重用          total-=money;          console.log(`花了${money}元,还剩${total}元`)        }      }      //强调: 函数只有调用才执行，才能获得返回结果      //      函数不调用，就不会执行，也不会获得返回结果！      //3. 调用者，必须调用外层函数，才能获得函数返回的内层函数。      var 孩子=父母();      //花了100      孩子(100);//希望剩900      //别人写的代码，可能有意或无意的篡改我的全局变量      total=0;      //又花了100      孩子(100);//希望剩800      //又花了100      孩子(100);//希望剩700    </script>  </body>  </html>  运行结果:  花了100元,还剩900元  花了100元,还剩800元  花了100元,还剩700元 |

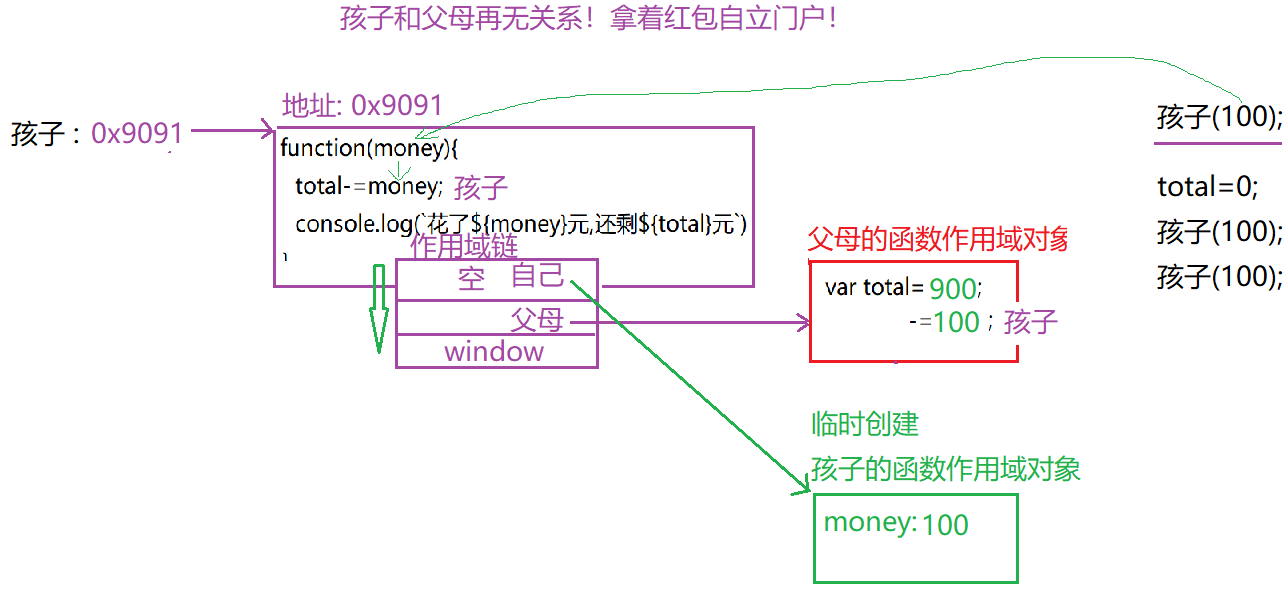
6. 原理:

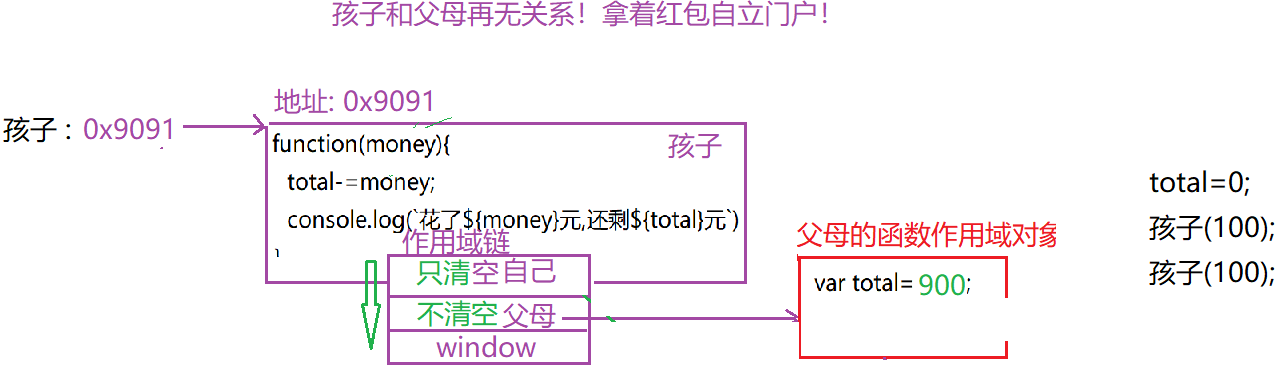


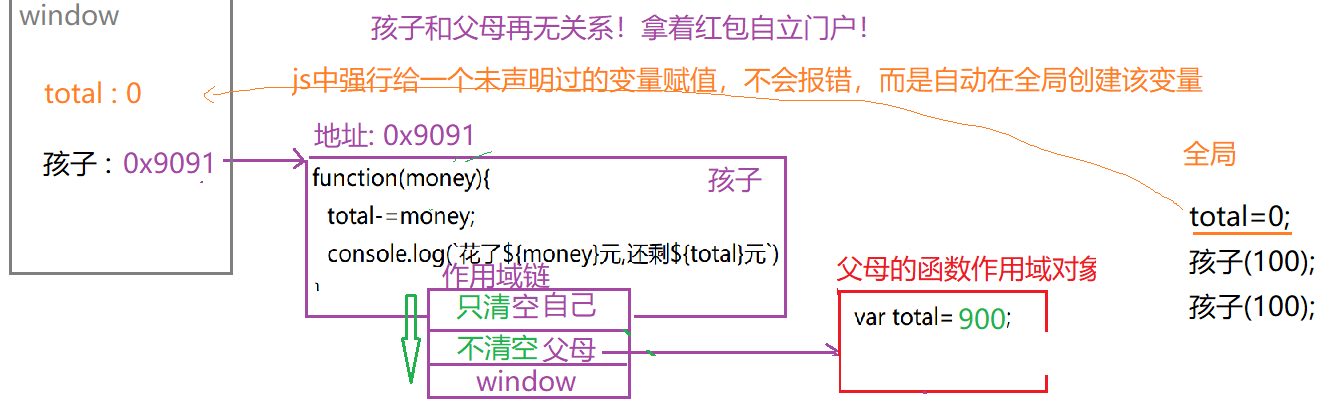


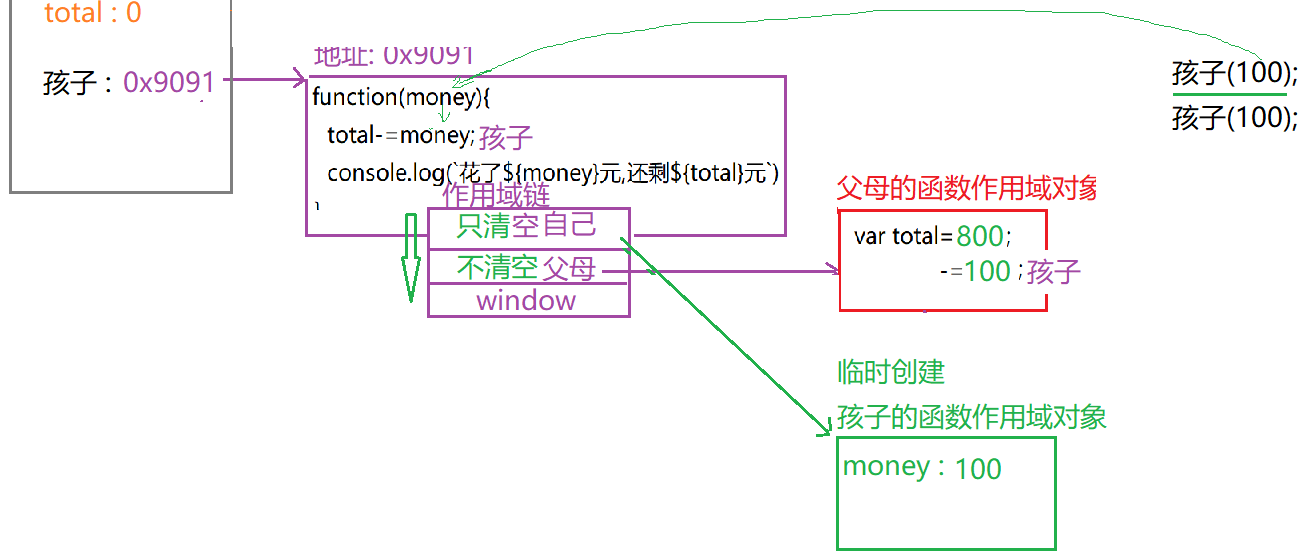


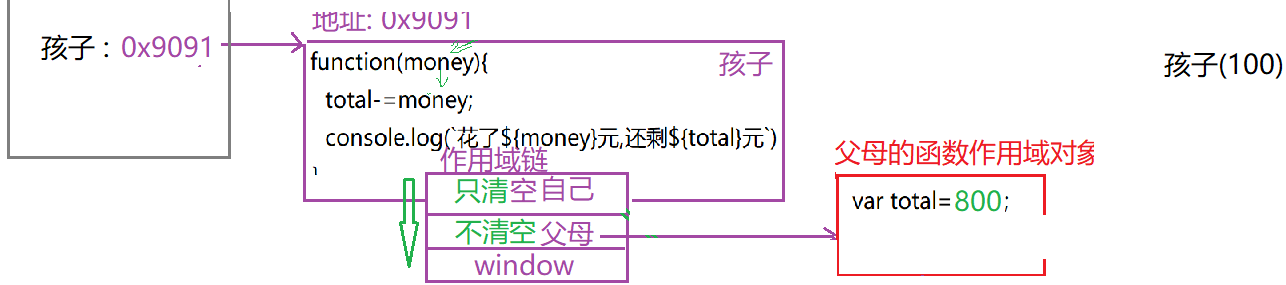


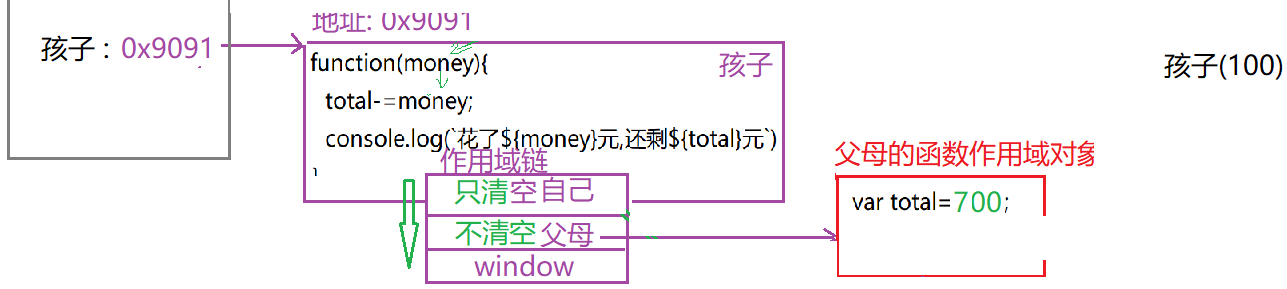






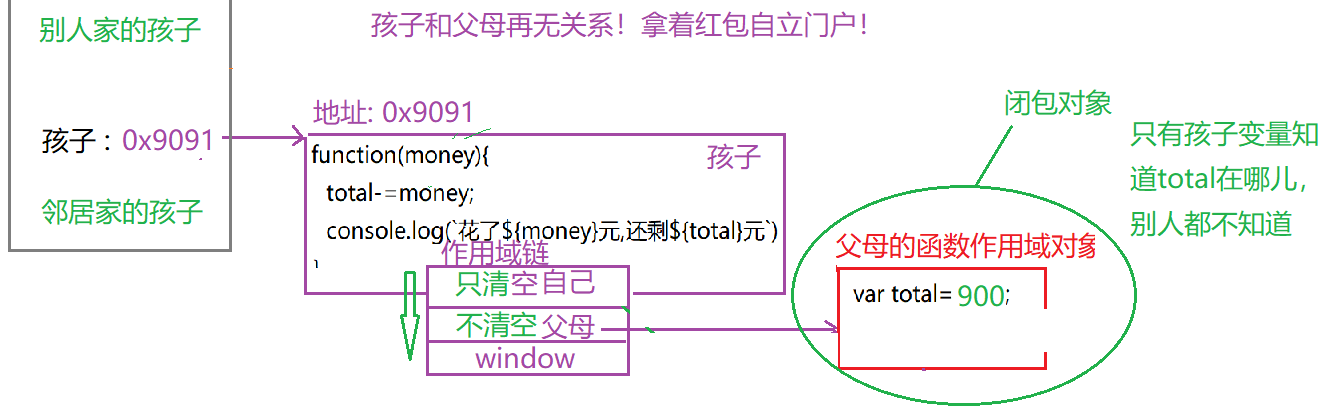






7. 总结:

(1). 到底什么是闭包: **外层函数的作用域对象**，因为受到内层函数的引用和保护，所以称为**闭包对象**。



(2). 鄙视题: 闭包形成的原因: **外层函数调用后，因为内层函数引用着外层函数的作用域对象，导致外层函数的作用域对象无法释放，形成了闭包！**

8. 简化: **理解闭包，只要找3样东西:**

(1). 妈妈函数是谁？function 父母(){}

(2). 妈妈函数要保护的局部变量是谁？var total=1000

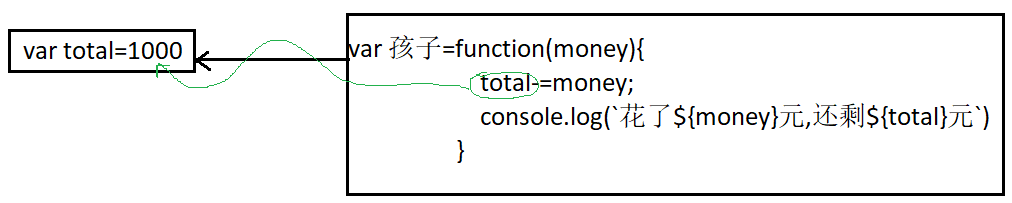
(3). 妈妈生出的内层函数孩子是谁，孩子最后叫什么名？

var 孩子=function(money){

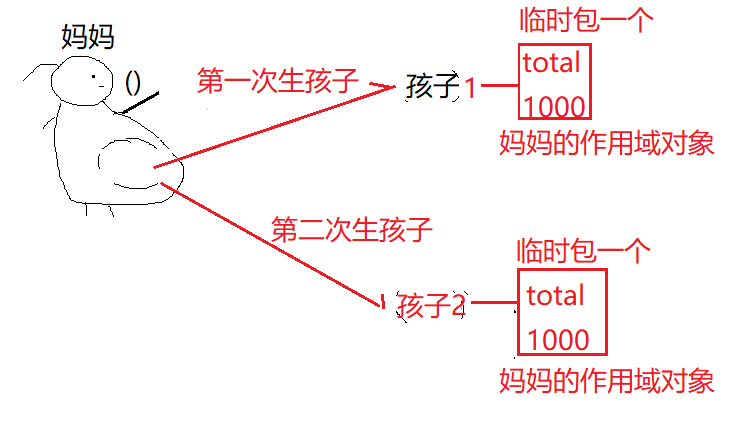
total-=money;

console.log(`花了${money}元,还剩${total}元`)

}



9. 问题: 多次调用外层函数妈妈，两次生出的孩子，是共用一个受保护的变量？还是各自有独立的受保护的变量？互补干扰？答: 多次调用妈妈，生出的多个内层函数，各自有独立的闭包变量，互不影响。



10. 总结:

(1). 妈妈**一次**调用生出的多个孩子，共用同一个红包

(2). 妈妈**多次**调用生出的多个孩子，各自有独立的红包，互不干扰！

11. 闭包的缺点: 比普通的函数多占用一块内存空间——外层函数的函数作用域对象

解决: 一旦一个闭包不再使用，应该尽快释放！

如何: **孩子=null**

\*\*\*\*面向对象\*\*\*\*

一. 什么是面向对象:

1. 什么是对象:

(1). 狭义的解释: 程序中专门保存现实中一个具体事物的属性和功能的程序结构

(2). 广义的解释: js中除五种原始类型(number, string, bool, null, undefined)之外，其余无法直接保存在变量中的数据类型都称为对象。

比如: var arr=[1,2,3]; arr是数组类型的对象，简称数组

var now=new Date() now是日期类型的对象，简称日期/时间对象

var fun=function(){ ... } fun是函数类型的对象，简称函数

... ...

在面向对象中只讨论狭义的自定义对象如何描述现实中的事物。

2. 什么是面向对象编程: 程序都是先用对象结构集中存储一个事物的属性和功能，然后再按需调用对象中的属性和功能

3. 为什么: 便于大量数据的管理和维护

4. 何时: 今后，几乎所有的项目，都用面向对象方式开发！

5. 如何: 面向对象三步/三大特点: **封装 继承 多态**

二. 封装:

对应小程序视频: 小程序->在线->JSCORE->day04 1. 用{}创建对象 this

1. 什么是: 创建一个对象集中保存一个事物的属性和功能

2. 为什么: 便于大量数据的管理和维护

3. 何时: 今后，只要使用面向对象方式编程，都要先创建对象。

4. 如何: 3种:

(1). 用{}创建一个对象: **{}**是new Object()的简写

var 对象名={

属性名 : 属性值,

... : ... ,

方法名 : function(){

... this.属性名 ...

}

}

(2). 用new Object()创建: ——很少用

对应小程序视频: 小程序->在线->JSCORE->day04 2. 3. 对象底层是关联数组 ...

a. 2步:

1). 创建一个空对象: **var 对象名=new Object();**

2). 强行向新对象中添加新属性和方法:

**对象名.属性名=属性值;**

**对象名.方法名=function(){ ... }**

b. 揭示: 对象底层其实也是**关联数组**

1). 都是名值对儿的集合

2). 都可用["属性名"]访问成员，都可简写为.属性名

**. 其实是 [""]**

强调: **如果将来属性名不是固定的，来自于其他变量，则~~既不能写["变量"]，又不能写.变量~~。只能写[变量]**

3). 尝试获取对象或关联数组中一个不存在的属性，都不会报错！而是返回**undefined**

固定套路: 判断一个对象中是否包含某个属性: 强行获取，看是不是undefined

**对象.属性名!==undefined**  说明包含！否则，说明不包含！

4). 尝试向对象或关联数组中不存在的属性强行赋值，都不会报错！而是自动添加该属性

固定套路: 如何给一个对象添加新属性或新方法: 唯一办法: 强行赋值！

对象.新属性名=新值;

对象.新方法名=function(){ ... }

5). 都可用for in循环遍历每个属性:

for(var 变量 in 对象或关联数组){

//in会依次取出对象或关联数组中每个属性名，自动保存在in前的变量中

}

5. 如何访问对象中的成员:

(1). 访问对象中的属性: 对象名.属性名

(2). 调用对象中的方法: 对象名.方法名()

6. this:

(1). 需求: 对象自己的方法中的内容，需要随对象自己的属性值动态变化

(2). 错误的解决: 直接在方法中写属性名

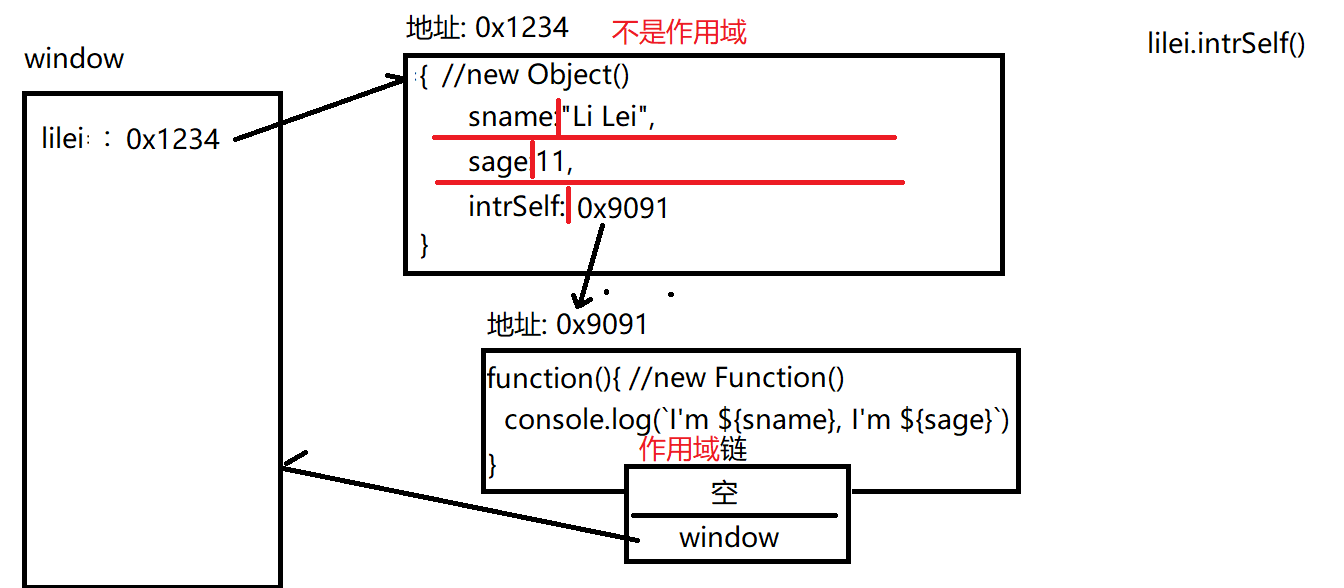
a. 结果: 报错: 属性名 is not defined

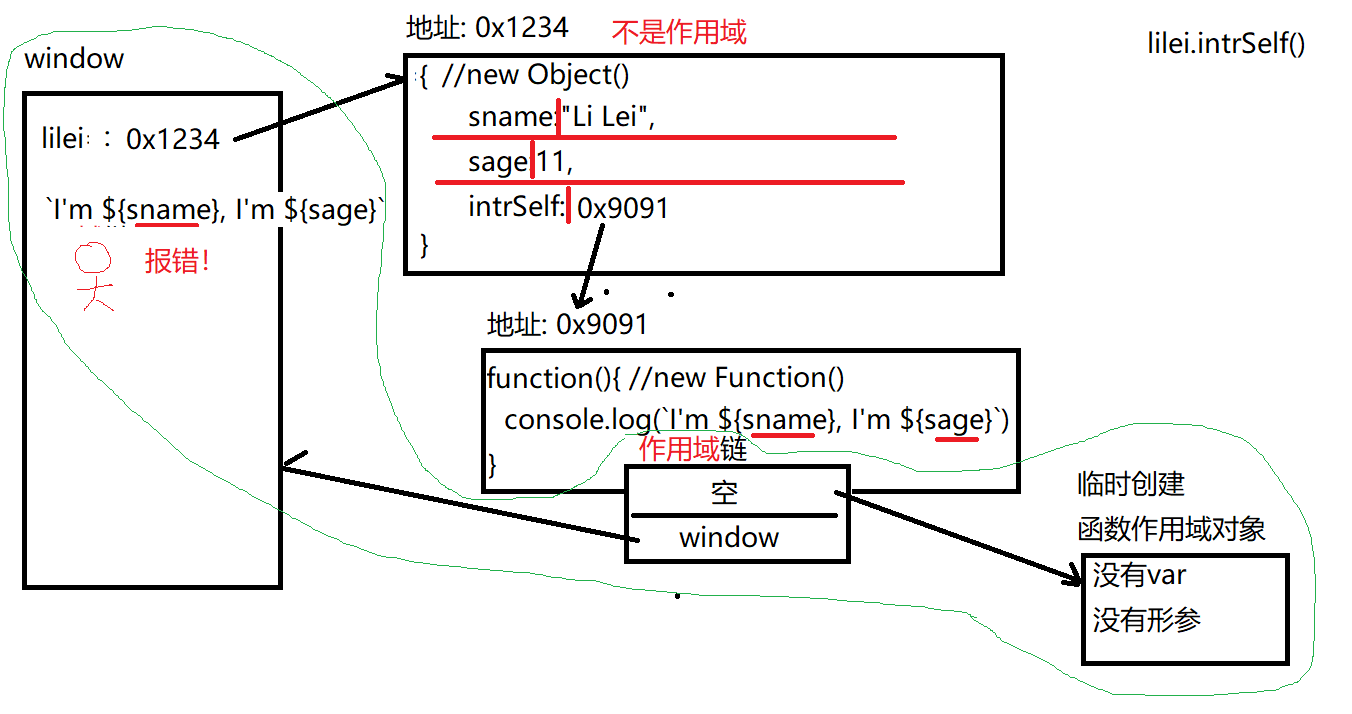
b. 原因:

1). 对象不在方法的作用域链中

2). 方法默认只能在自己内部或全局作用域查找变量，无权擅自进入某个对象中查找属性！

3). 对象的属性又不是全局变量

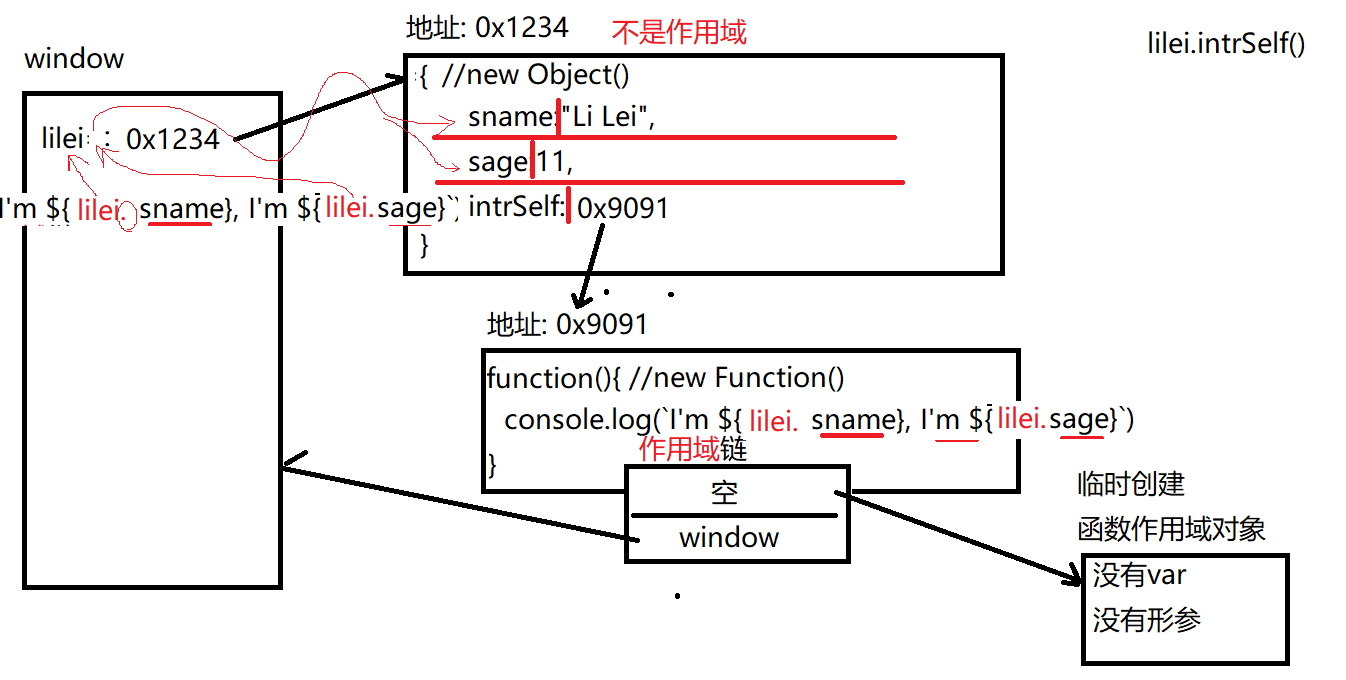




(3). 正确的但是不好的解决: 在对象的方法中写死"对象名.属性名"

原理: 让程序先找到指定名称的对象，.运算符可让程序进入对象中，读取对象的属性

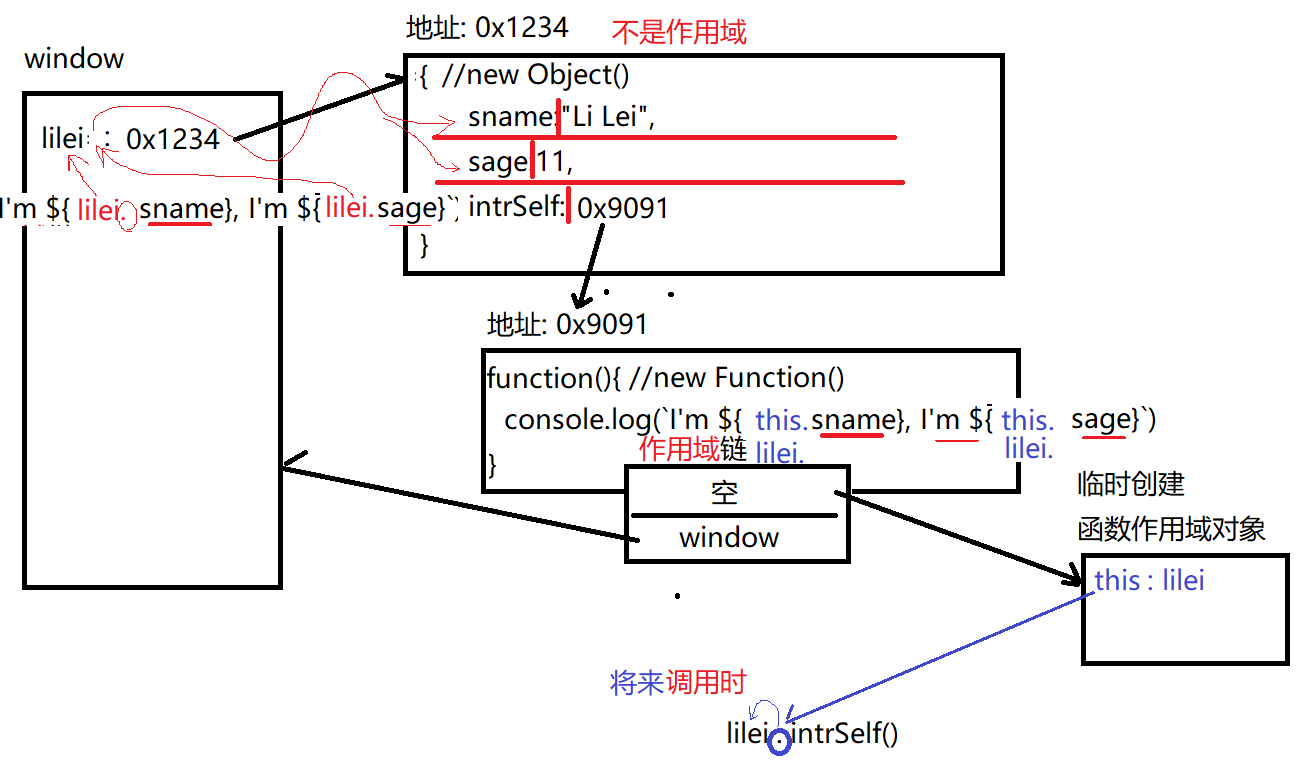
问题: 紧耦合，外部的对象名改变，也必须手动修改内部写死的对象名。一旦忘记修改内部的写死的对象名，程序立刻就出错！——不好！不便于维护！

 (4). 正确且好的解决方法: this

a. 什么是this: 自动获得正在调用当前函数的**.前的对象** 的js关键词

b. 何时: 今后只要对象的方法想使用对象自己的属性都必须加"this."

c. 强调: 今后判断this指向哪里，一定~~不要看它定义在哪儿~~！只看**调用时.前**是谁！



7. 示例: 使用{}创建一个对象，并在对象方法中使用this获得对象自己的属性

3\_{}.html

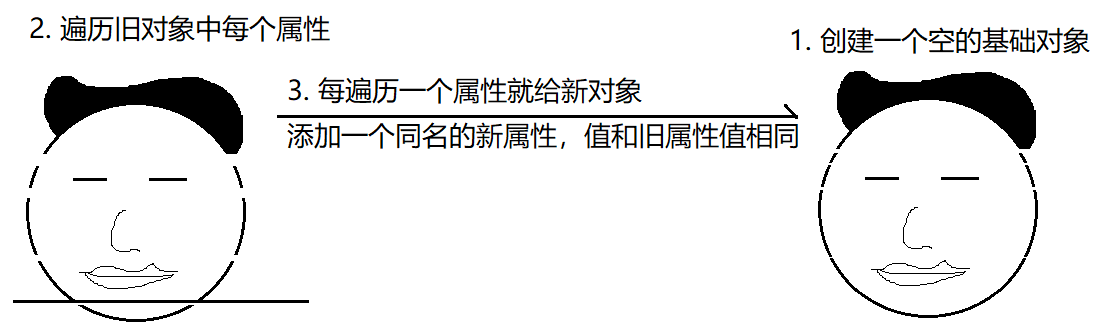
|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <title>Document</title>  </head>  <body>    <script>      //创建一个学生对象lilei，保存李磊的:      //姓名: Li Lei      //年龄: 11      //保存李磊会做的一件事:      //自我介绍:      var lilei={//不是作用域      //js中只有两种作用域: 全局作用域，函数作用域      //对象中的内容既不是全局，也不是函数作用域      //对象仅仅是一种结构复杂的特殊存储结构而已！        sname:"Li Lei",        sage:11,        intrSelf:function(){          console.log(`I'm ${this.sname}, I'm ${this.sage}`)        }//作用域链: ？个格子？      }        console.log(lilei);      //输出李磊的年龄      console.log(`李磊今年:${lilei.sage}`);      //请李磊做自我介绍:      lilei.intrSelf();//this.sname    this.sage      //                  ↓             ↓      //                lilei.sname    lilei.sage      //过了一年，李磊长了一岁      lilei.sage++;      //再输出李磊的年龄      console.log(`李磊今年:${lilei.sage}`);      //再请李磊做自我介绍:      lilei.intrSelf();//this.sname    this.sage      //                  ↓             ↓      //                lilei.sname    lilei.sage  **var intrSelf=lilei.intrSelf;**      intrSelf();// **I'm undefined  I'm undefined**  //其实执行的是**window.intrSelf()**，**intrSelf()**中的this指window，因为window中没有sname和sage，所以window.sname和window.sage返回undefined      </script>  </body>  </html>  运行结果:  {sname: "Li Lei", sage: 11, intrSelf: ƒ}  李磊今年:11  I'm Li Lei, I'm 11  李磊今年:12  I'm Li Lei, I'm 12  I'm undefined, I'm undefined |

8. 示例: 验证对象底层就是关联数组:

4\_newObject.html

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <title>Document</title>  </head>  <body>    <script>      //创建lilei      var lilei=new Object();      console.log(lilei);//{}      lilei.sname="Li Lei";      lilei["sage"]=11;      lilei.intr=function(){        console.log(`I'm ${this.sname}, I'm ${this.sage}`)      }      console.log(lilei);      lilei["intr"]();      //尝试获取lilei的班级      console.log(lilei["className"]);//undefined      //尝试为李磊添加班级属性      lilei.className="初一2班";      console.log(lilei);      //用关联数组      var lilei=[];      lilei.sname="Li Lei";      lilei["sage"]=11;      lilei.intr=function(){        console.log(`I'm ${this.sname}, I'm ${this.sage}`)      }      console.log(lilei);      lilei.intr();      console.log(lilei.className);//undefined      lilei["className"]="初一2班";      console.log(lilei);    </script>  </body>  </html>  运行结果:  {}  {sname: "Li Lei", sage: 11, intr: ƒ}  I'm Li Lei, I'm 11  undefined  {sname: "Li Lei", sage: 11, className: "初一2班", intr: ƒ}  [sname: "Li Lei", sage: 11, intr: ƒ]  I'm Li Lei, I'm 11  undefined  [sname: "Li Lei", sage: 11, className: "初一2班", intr: ƒ] |

9. 示例: 克隆一个对象:



5\_clone.html

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <title>Document</title>  </head>  <body>    <script>      //定义函数，可以克隆一个对象      function 克隆(旧对象){        //1. 先创建一个新的空对象        var 新对象={};        //2. 遍历旧对象中每个属性        for(var 旧属性名 in 旧对象){          //3. 每遍历一个属性，就为新对象添加同名的属性，值为旧对象中同名属性值                        //取出旧对象中旧属性值                      //放入          //新对象中强行添加的同名属性中  **新对象[旧属性名]=旧对象[旧属性名]**          //           ←搬家←          //千万不能加""，因为 旧属性名 是in前的一个变量，变量值每遍历到一个新属性时，获得的属性名都不一样！是变化的！所以不能加""          //也不能用.旧属性名。因为.等效于[""]        }        //返回创建好的新对象  **return 新对象;**      }      //测试:      var lilei={        sname:"Li Lei",        sage:11      }      //错误:      //var lilei2=lilei;//将lilei变量中保存的旧对象地址复制一份给lilei2变量。结果: lilei2和lilei使用相同的对象地址，引用同一个对象！——不叫克隆！      //正确:      var lilei2=克隆(lilei);      console.log(lilei);      console.log(lilei2);      console.log(lilei==lilei2);//false      //如果==左右都是对象，则==不再比较对象内容，而是比较两个对象的地址是否相同！      //如果返回true，说明克隆失败！因为地址是同一个对象，说明没有多出一个对象      //如果返回false，说明克隆成功！因为地址不同！说明是先后创建的两个对象！    </script>  </body>  </html>  运行结果:  {sname: "Li Lei", sage: 11}  {sname: "Li Lei", sage: 11}  false |

总结:

2. RegExp对象:

(1). 创建正则表达式对象:

a. 如果正则是固定的: var reg=/正则/ig

b. 如果正则需要动态生成: var reg=new RegExp("正则",ig)

(2). 验证字符串格式: var bool=reg.test(str)

(3). 既查找每个关键词的内容又查找每个关键词的位置:

do{

var arr=reg.exec(str);

if(arr!=null){

获得本次找到的敏感词的内容(arr[0])和位置(arr.index)

}

}while(arr!=null);

3. 函数:

(1). 创建函数三种方式:

a. function 函数名(形参列表){ 函数体; return 返回值 } //会被声明提前，不好

b. var 函数名=function(形参列表){ 函数体; return 返回值 }//不会被声明提前，首选

c. var 函数名=new Function("形参1", "形参2", ... , "函数体; return 返回值")

|  |
| --- |
| 函数本质:  1. 函数也是一个对象，对象中保存着函数的函数体代码  2. 函数名只是一个普通的变量，函数名通过函数对象地址，引用着函数对象  3. function在底层等效于new Function()  function 函数名(){ ... }，底层等效于 var 函数名=new Function(...) |

(2). 重载: 今后，只要一个函数将来调用时不确定传入实参值的个数，都要用重载

function 一个函数名(不写形参变量){

//arguments对象自动接住所有实参值

if(arguments.length==0){

执行一种逻辑

}else if(arguments.length==1){

执行另一种逻辑

}else{

执行其它逻辑

}

}

其中arguments是类数组对象: 和数组相比:

a. 相同点: 也有下标，length属性，也可for循环遍历

b. 不同点: 不是数组类型，无法使用数组家的函数

(3). 匿名函数:

a. 所有回调函数优先使用匿名函数——用完释放，节约内存

b. 所有js代码都应该保存在匿名函数自调中，禁止使用全局变量，避免全局污染！

(function(){

要执行的js代码

})()

结果: 匿名函数内的都是局部变量，不会产生全局变量。

局部变量随匿名函数一起释放。不会污染全局。

(4). 作用域和作用域链: (跟着视频亲自画图！！！)

a. 作用域:

1). 全局作用域:window，保存全局变量

优: 可重用，缺: 随处可用, 极易被污染

2). 函数作用域: 保存局部变量

优: 仅函数内可用，不会被污染，缺: 不可重用

函数作用域对象原理:

i. 每个函数定义时都自带好友列表，好友列表里2个格子，一个是空，一个引用window

ii. 调用函数时临时创建函数作用域对象保存函数局部变量。并将函数作用域对象的地址保存到函数好友列表中离自己近的格子里。

iii. 函数执行过程中按就近原则先在自己的函数作用域对象中找局部变量使用。如果找不到，才被迫去全局window中找变量使用.

iv. 函数调用后，好友列表中离自己近的格子清空，导致函数作用域对象以及内部的局部变量被释放！——所以局部变量不可重用！

b. 作用域链: 保存一个函数所有可用的作用域对象的链式结果(好友列表)学名就叫作用域链。作用域链保存着一个函数可用的所有变量，作用域链控制着变量的使用顺序。先局部后全局。

(5). 闭包: （跟着视频亲自画图!!!）只要希望给一个函数保护一个可反复使用的变量，又防止这个变量被外界篡改时，都用闭包。

a. 闭包三步:

1). 用外层函数妈妈包裹要保护的变量和内层函数

2). 外层函数妈妈用return把内层函数孩子返回到外部

3). 外部想使用内层函数的人，必须调用外层函数，才能获得return出来的内层函数对象。并将内层函数保存在一个变量中反复使用。

b. 闭包形成的原因: 外层函数调用后，外层函数的作用域对象被内层函数引用着无法释放，形成了闭包对象

c. 闭包的缺点: 闭包比一般的函数占用多一块内存——外层函数的函数作用域对象。所以，用完闭包后，应该尽快释放: 变量=null

d. 画简图:

1). 找三样东西:

i. 外层函数——妈妈

ii. 受保护的变量——红包

iii. 外层函数共返回几个内层函数对象——孩子(们)

2). 规律:

i. 调用一次妈妈，返回的多个孩子，共用一个红包

ii. 调用多次妈妈，返回的多个孩子，有各自的红包，互不干扰！

4. 面向对象: 封装， 继承， 多态

(1). 封装: 3种:

a. 用{}:

var 对象名={

属性名:属性值,

... : ... ,

方法名: function(){

... this.属性名 ...

}

}

b. 用new Object():

1). 2步:

i. var 对象名=new Object()

ii. 对象名.属性名=属性值;

对象名.方法名=function(){ ... }

2). 对象底层也是关联数组:

i. 都是名值对儿的集合

ii. 都可用[""]和.方式访问成员。但是如果属性名来自于变量就只能用[]

iii. 访问不存在的属性，不报错，返回undefined

判断是否包含某个属性: 对象.属性名!==undefined

iv. 强行给不存在的属性赋值，不报错，而是自动添加该属性

给对象添加新属性，唯一办法，强行赋值！

v. for in遍历

|  |
| --- |
| 总结: this 2种:  1. obj.fun() this->obj  2. fun() this->window |

今日对应小程序视频列表:

小程序->在线->JSCORE->day03 7. 原理 作用域和作用域链

8. 闭包 closure压岁钱

小程序->在线->JSCORE->day04 1. 用{}创建对象 this

2. 3. 对象底层是关联数组 ...

作业:

1. 复习今日问题清单(小程序->在线->JSCORE->day03、day04)

2. 使用闭包实现取号机功能:

//通过闭包三步实现取号机函数

//....

//测试代码:

var getNum=外层函数()

getNum();//1

getNum();//2

//如果用i记录当前号码，即使写i=0

i=0;

getNum();//3

getNum();//4

答案: 小程序->首页 搜取号

3. 看小程序视频学习闭包笔试题——自己试着找关键要素画简图

小程序->在线->JSCORE->day03 作业: 闭包 高频笔试题 nAdd getN

4. 跟着小程序视频或tmooc视频，亲自用画图工具（不要用纸笔！！！）画程序运行过程中的内存变化图，体会程序运行过程和内存变化。

5. 预习: 小程序->在线->JSCORE->day04 5,6,9