提前准备:

1. 今天的历届师兄师姐问过的经典的高频的问题都已经整理在小程序->首页->JSCORE->day03里

2. 今天的课程中所需的单词列表都已经整理在小程序->单词->JSCORE->day03里

3. 今天的重点商品、扩展高频鄙视题视频已经整理在小程序->在线->JSCORE->day03里

如果不知道问什么: 谁问的越多！谁培训费花的越值！

1. 笔记中或画图中那句话或者那个位置不明白，立刻问！

2. 案例中哪行代码或注释不明白，立刻问！

3. 课下自己做项目，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！——公众号: 前端大全 定期总结大厂笔试题

5. 看小程序问题清单时，遇到看不懂的问题，立刻问！

所有带注释的案例: 每个案例至少写三遍！

1. 注释+代码抄一遍！

2. 保留注释，删掉代码，自己试着把代码填回来！

3. 删除注释和代码，用自己的话把注释和代码一起写回来！

标准: 直到自己可以用人话把注释写出来！(代码记不住都没关系！可以翻笔记！)

/未来6天，将学习内功心法和基本功，不要考虑功能怎么做！/

正课:

1. 函数

2. 重载

3. 匿名函数

4. 作用域和作用域链

5. \*\*\*\*\*闭包

一. 函数:

1. 什么是函数: 程序中专门保存一段可反复使用的代码段的程序结构，再起一个名字

2. 为什么使用函数: 重用!

3. 何时: 只要一段代码片段，可能被反复使用时，都要保存在一个函数中，然后反复调用函数。

4. 如何定义函数: 3种:

(1). 用声明方式定义函数:

a. function 函数名(形参列表){

要反复使用的代码片段

return 返回值;

}

b. 形参:

1). 什么是: 专门接受函数执行时必须的参数值的变量

2). 为什么:

i. 因为有些函数必须提供某些数据才能正常执行！

ii. 为了让函数变的更灵活！可根据不同的实参值，执行不同的操作。

3). 何时:

i. 如果一个函数必须提供某些数据才能正常执行时，必须提供形参！必须几个值，就要定义几个形参变量

ii. 如果一个函数根据不同的实参值，执行不同的操作时，也可以定义形参。

4). 如何:

function 函数名(形参变量1, 形参变量2, ...){

... 使用形参变量 ...

}

强调: 形参变量即使没有var！也能自动创建该变量

5). 如果一个函数不需要外界传入值，就没必要定义形参

c. 返回值:

1). 什么是: 一个函数的执行结果！

2). 为什么: 因为函数内的执行结果，默认函数外不可使用！

3). 何时: 如果将来调用这个函数的人，希望获得函数的执行结果时，才用返回值!

4). 如果调用函数的人不需要知道函数的执行结果时，就不用返回值！

5). 如何:

function 函数名(){

...要重用的代码段...

return 返回值 //小提示: return和返回值之间一定不要加换行

}

6). 强调:

i. return和返回值之间一定不要换行，因为return后如果没东西，js会自动给return后加; 表示什么都不返回

ii. return不能用于三目内部:

~~条件?return 值1: return 值2~~

return 条件?值1:值2

d. 示例: 定义一个可以做饭的函数，接受不同的食材，做不同的饭

|  |
| --- |
| //想定义一个函数可以做饭  function cook(dan,cai,mian){    console.log(`炒${dan}`);    console.log(`炒${cai}`);    console.log(`${cai}和${dan}一起炒`);    console.log(`煮${mian}`);    console.log(`将炒好的${cai}${dan}和煮好的${mian}搅拌在一起`)    return `一碗热腾腾香喷喷的${cai}${dan}${mian}`  }  //星期一想吃西红柿鸡蛋面:  var wan=cook("鸡蛋","西红柿","面条");  console.log(`吃 ${wan}`);  //星期二没西红柿了！只有黄瓜！  var wan=cook("鸡蛋","黄瓜","面条");  console.log(`吃 ${wan}`);  //星期三没有鸡蛋了，只有辣条!  var wan=cook("辣条","黄瓜","面条");  console.log(`吃 ${wan}`); |

(2). 用赋值方式定义函数:

a. var函数名=function (形参列表){

要反复使用的代码片段

return 返回值;

}

b. vs 声明方式比较:

1). 将来调用时，用法完全一样！

2). 声明提前:

i. function 函数名(形参列表){ ... } 会整体被声明提前

ii. var 函数名=function(){ ... } 只有=前的函数名会被声明提前，=后的function(){ ... } 会留在原地.

c. 总结: 基本上都会用赋值方式，代替声明方式来定义函数，避免被声明提前，打乱顺序!

d. 揭示了: js中函数的本质:

1). 函数其实也是一个对象，只不过对象中保存的不是一个数字，不是一个字符串等值，而保存的是一段代码片段！

2). 函数名其实仅仅是一个普通的变量而已！

3). 函数名中通过保存函数对象的内存地址值，来引用着函数对象

4). 当调用函数时，js引擎会先找到函数名变量，再通过变量保存的函数地址找到函数对象，然后才执行函数对象内保存的代码片段！

更多声明提前的知识: 下课后做作业:

看小程序->在线->JSCORE->day03: 1. 声明提前和3. 声明提前...

e. 示例: 比较声明方式创建函数和赋值方式创建函数的差别

|  |
| --- |
| //var fun=function(){  function fun(){    console.log(1);  }  fun(); //2  //var fun=function(){  function fun(){    console.log(2);  }  fun(); //2 |

(3). 用new来创建函数:

a. var 函数名=new Function("形参变量1","形参变量",...,"函数体; return 返回值");

b. 很少使用！

c. 更揭示了: js函数的本质: 函数也是一个对象

其实function 只是 new Function()的简写而已！

function等效于new Function()

d. 示例: 两个函数做比较:

|  |
| --- |
| var a=function(){console.log(1)};      //new Function() 地址: 0x1234  var b=function(){console.log(1)};      //new Function() 地址: 0x9091  console.log(a==b); //false  //     0x1234==0x9091 |

5. 如何调用函数:

(1)var 变量=函数名(实参值列表);

(2)结果:

a. js引擎先找到指定名称的函数

b. 将调用函数时()中的实参值列表，逐一赋值给定义函数时()中的形参变量

c. 按函数定义中的代码片段，执行任务

d. 将函数返回的执行结果，保存到=左边的变量中

二. 重载(overload):

1. 什么是: 相同函数名，不同形参列表的多个函数，在调用时，可根据传入实参值的不同，自动匹配对应的函数调用执行.

2. 何时: 今后只要一件事，希望根据传入实参值的不同，动态执行不同的操作时

3. 为什么: 为了减少函数名的个数，减轻调用者的负担！

4. 如何:

(1). 问题: Js语法默认不支持重载效果！

因为js中不允许多个同名函数同时存在

如果同时存在多个同名函数，则后定义的同名函数会覆盖先定义的同名函数

(2). 解决: js中是通过一个特殊arguments对象来变通实现重载效果

a. 什么是arguments:

1). 每个函数中自带的，无需自己创建，就可直接使用

2). 自动接收所有传入函数内实参值，

js中即使没有定义形参变量，将来调用时，依然可以所以传入实参值！因为有arguments自动接着呢！

3). 类数组对象:

i. 什么是: 长得像数组的对象

ii. vs 数组:

相同: 1. 下标i, 2. length, 3. 可用for循环遍历

arguments: { 0: 值1, 1:值2,... , length:元素个数 }

不同: 类数组对象不是数组家孩子！无权使用数组家任何函数！

b. 如何arguments变通实现重载:

1). 只定义一个函数，且不要定义形参！

2). 在函数内部，判断arguments对象内容或元素个数不同，来选择执行不同的操作

5. 示例: 定义一个pay函数，支持三种付款方式:

|  |
| --- |
| //想定义一个函数可以付款:  //但是现实中有三种付款方式:  //1. 手机支付: 不需要提供任何东西，直接用手机扫描即可支付  //2. 现金支付: 需要将现金交给收款员手中  //3. 刷卡支付: 既需要刷卡识别卡号，又需要输入密码  function pay(          ){   //arguments{          }.length   //           0  1  2    console.log(arguments);    if(arguments.length==0){//说明没传入任何实参值      console.log("手机支付...");    }else if(arguments.length==1){//说明只传入一个实参值      console.log(`现金支付...,收您:${arguments[0]}元现金`)    }else{//说明传入了1个以上的实参值      console.log(`刷卡支付...从你卡号:${arguments[0]}中扣款成功！`)    }  }  //将来调用时:  pay();//手机支付  pay(100);//现金支付  pay("6553 1234","123456")//刷卡支付 |

三. 匿名函数:

1. 什么是: 定义函数时，不指定函数名

2. 为什么:

(1). 为了节约内存:

(2). 避免使用全局变量造成全局污染——将来程序中，禁止使用全局变量！

3. 何时:

(1). 只要一个函数只使用一次时，都要使用匿名函数！

(2). 今后所有js程序都要包裹在一个匿名函数中！避免使用全局变量

4. 如何:

(1). 几乎所有回调函数都用匿名函数:

a. 比如:

arr.sort(function(a,b){return a-b});

str=str.replace(/\b[a-z]/ig,function(kw){ return kw.toUpperCase() })

b. 目的都是: 节约内存，让回调函数在外层函数调用后，也跟着自动释放！

c. 示例: 在数组排序中使用匿名函数

|  |
| --- |
| var arr=[12,23,123,32,2,3,1];  //arr.sort();//默认将每个元素都转为字符串，按字符串中字符编码的大小来排序，按位比较每个字符的unicode号。  //不好的做法: 用有名称的函数，在arr.sort()调用后，依然占用内存，无法释放。所以，浪费内存  // var fun=function(a,b){return a-b}  // arr.sort(fun);  //好的做法: 没有名称的函数在arr.sort()调动后，就一起被垃圾回收了！节约内存!  arr.sort(function(a,b){return a-b});  //arr.sort(function(a,b){    //console.log(a,b,"比较")    //return a-b  //});  console.log(arr); |

(2). 匿名函数自调:

a. 什么是: 定义一个匿名函数后，立刻调用该函数

b. 何时: 今后所有js程序都要包裹在一个匿名函数自调中！避免使用全局变量

c. 如何:

1). 标准: (function(){

... ...

})()

强调: 匿名函数自调前后的语句必须都加";"，不能偷懒!

报错: Uncaught TypeError: (intermediate value)(intermediate value)(...) is not a function

2). 非主流写法:

i. !function(){ ... }()

ii. +function(){ ... }()

d. 示例: 使用匿名函数自调避免使用全局变量

|  |
| --- |
| <body>  <script>  !function(){    var start=new Date();    alert(`开始加载页面内容, at: ${start.toLocaleString()}`);  }();//分号不能省略！  (function(){    alert("正在努力加载中...")  })();  </script>  <h1>Welcome</h1>  <p>welcome to my web site</p>  <script>  (function(){    var end=new Date();    alert(`页面内容加载完成, at: ${end.toLocaleString()}`);  })()  </script>  </body> |

四. 作用域(scope)和作用域链(scope chain)：

1. 什么是作用域:

(1).一个变量的可用范围

(2). 保存变量的对象

2. 为什么: 为了避免不同范围的变量间互相干扰！

3. js中作用域包括: 2种:

(1). 全局作用域对象: window

a. 保存着: 所有全局变量和全局函数

b. 特点: 随处可用，可反复使用！

(2). 函数作用域: ？

a. 保存着: 仅函数内可用的所有局部变量

b. 特点: 仅函数内可用，且不可重用！

4. 示例: 比较全局变量和局部变量

|  |
| --- |
| //楼道:  var 饮水机=10;  function 教室(){    /\*var\*/ 饮水机=100;    饮水机++;    console.log(饮水机)  }  教室();//101  console.log(饮水机);//101 |

5. 作用域对象的原理:

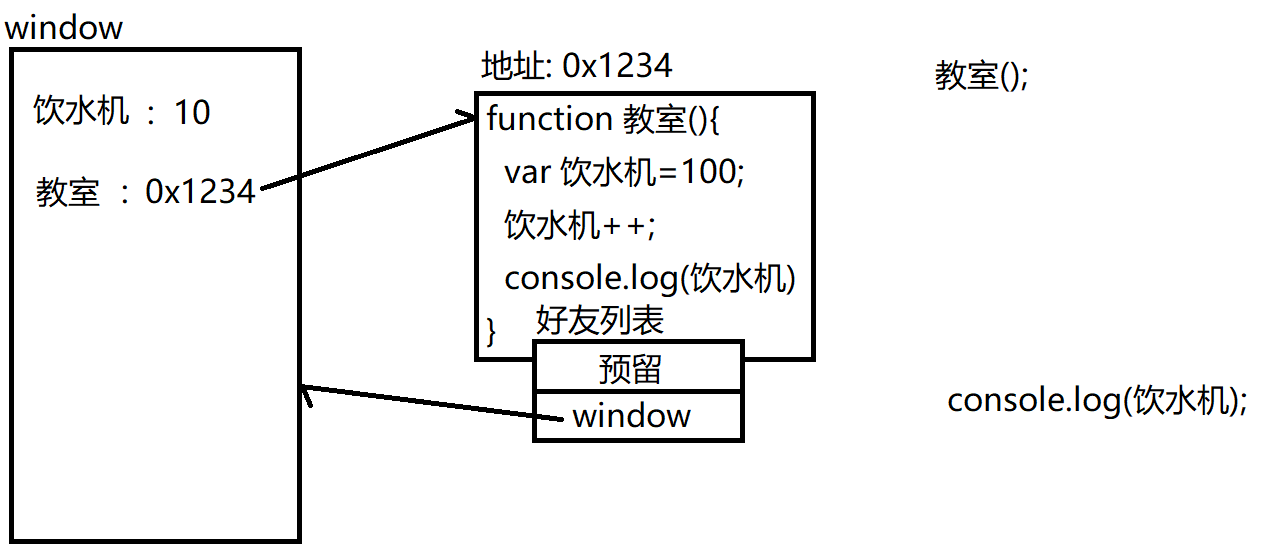
(1). 只要打开浏览器窗口，就会在内存中自动创建一个window对象，作为全局作用域对象。所有的全局变量和全局函数都会默认保存在window中

(2). 定义一个函数时，仅创建函数对象来保存函数的内容，暂时不会创建函数作用域对象和局部变量的！

其中: 每个函数对象都有一个"好友列表"，普通函数的好友列表中只有两个好友:

a. 离函数最近的位置: 暂时预留

b. 离函数稍远的位置: window



(3). 调动函数时，才临时创建本次函数调用所使用的函数作用域对象

a. 函数作用域对象中会根据函数内容的需要定义所有局部变量: 2种:

1). 在函数内var出来的变量，是局部变量

2). 形参变量也会成为局部变量

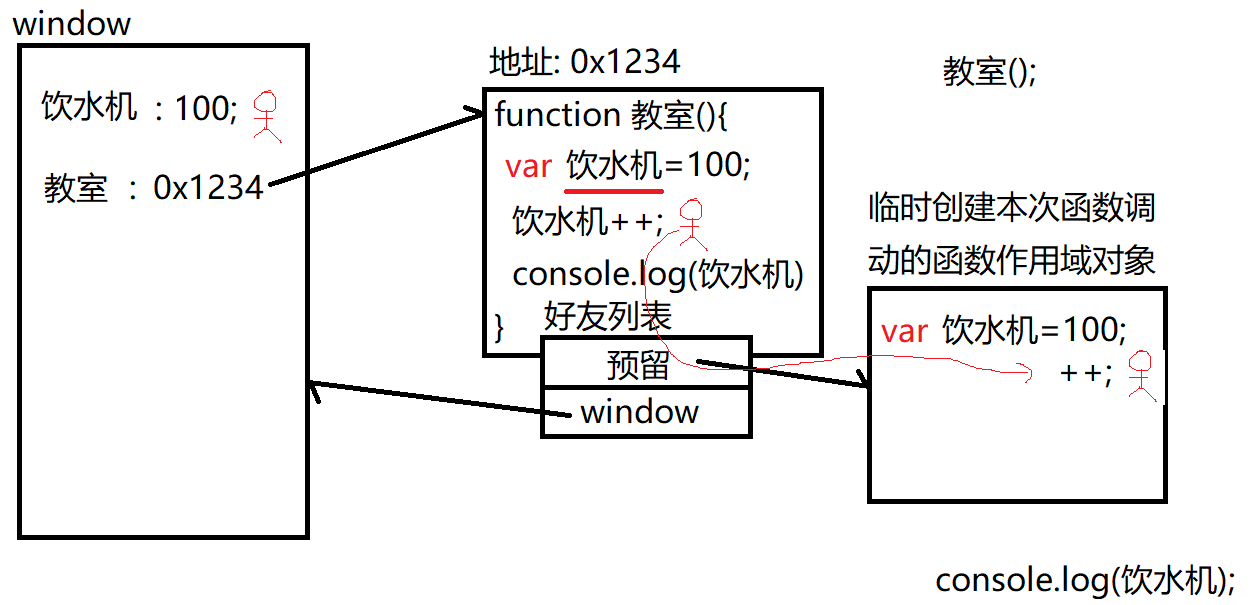
b. 将临时创建的函数作用域对象，保存到函数对象的好友列表中离函数最近的位置！

c. 执行函数中的操作时，一旦用到某个变量:

1). 首先：去离函数近的函数作用域对象中查找变量

2). 如果离函数近的函数作用域对象中没有所需的变量，才会想到去window找。

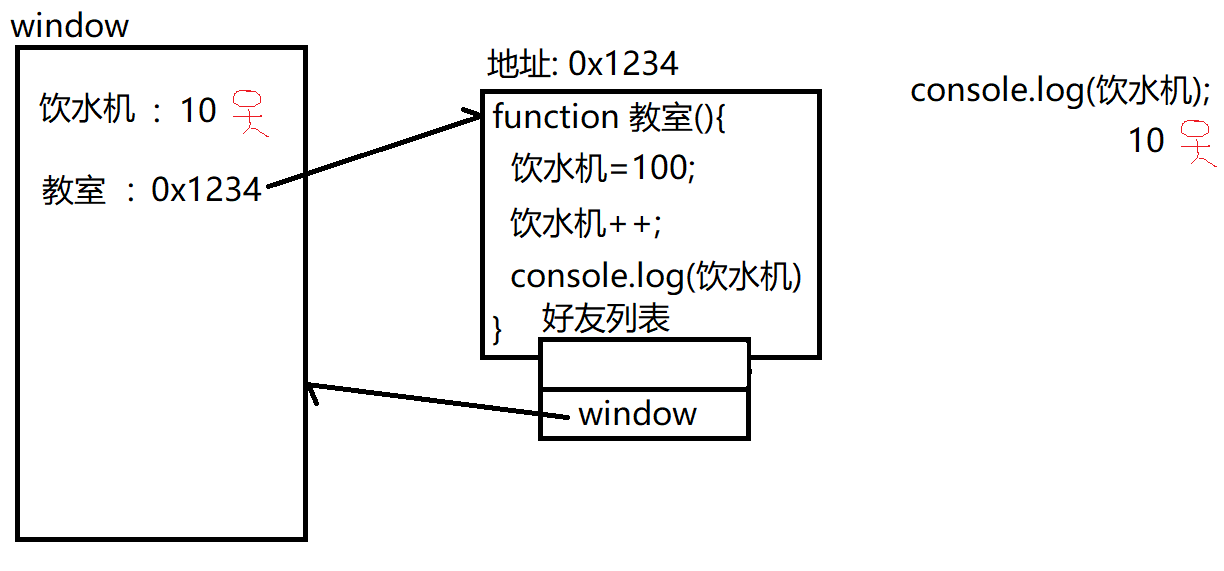
3). 最后，只有函数作用域对象中和window中都没有所需的变量，才会报错！



(4). 函数调用后，会释放临时创建的函数作用域对象

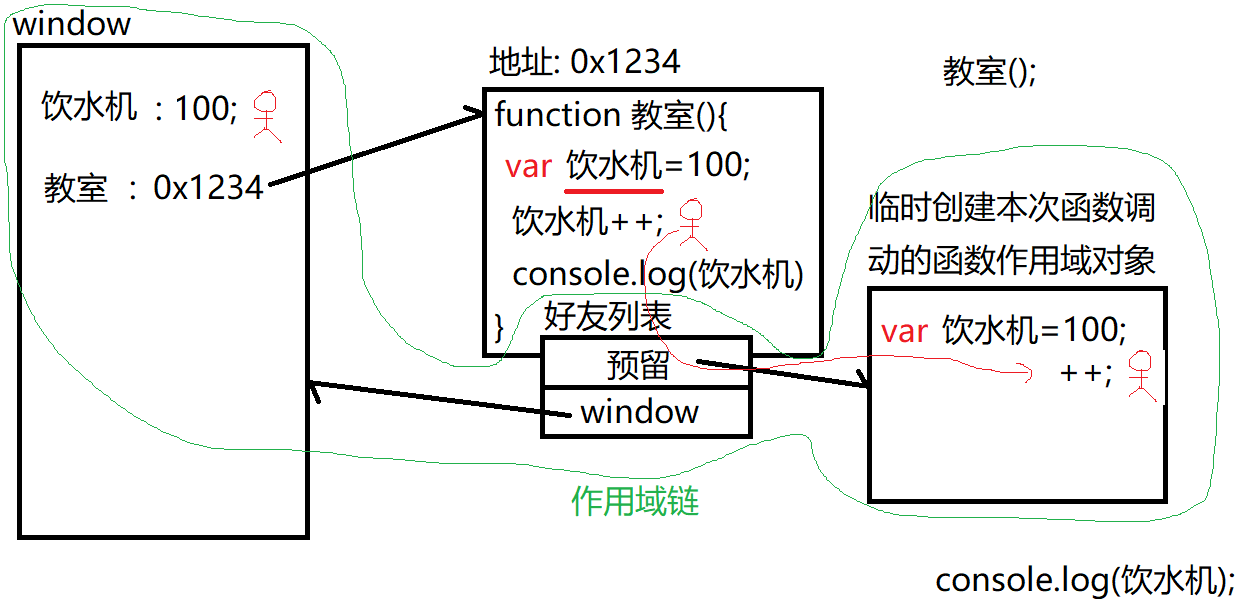
导致: 临时创建的局部变量也就跟着释放了！

但是函数对象，因为有函数名变量引用着，所以不会释放！



6. 作用域链:

(1). 什么是: 由多级作用域连在一起形成的链式结构



(2). 保存着: 一个函数可用的所有变量(包括局部变量和全局变量)

(3). 控制着变量的使用顺序: 先局部后全局

五. 闭包:

1. 什么是:

(1). 从用法来说: 既重用一个变量，又保护一个变量不被篡改的一种编程方式

(2). 本质: 外层函数的作用域对象被内层函数应用着无法释放，这样的作用域对象就称为闭包对象.

2. 为什么: 全局变量还是局部变量都有不可兼得的优缺点

(1). 全局变量:   
 a. 可重用！

b. 随处可用——全局污染

(2). 局部变量:

a. 仅函数内可用，不会造成全局污染

b. 不可重用！

3. 何时: 今后只要希望给一个函数保存一个专属的变量，既可重用，又不会被篡改时，必须用闭包！

4. 如何: 闭包3步:

(1). 用外层函数包裹要保护的变量和内层函数

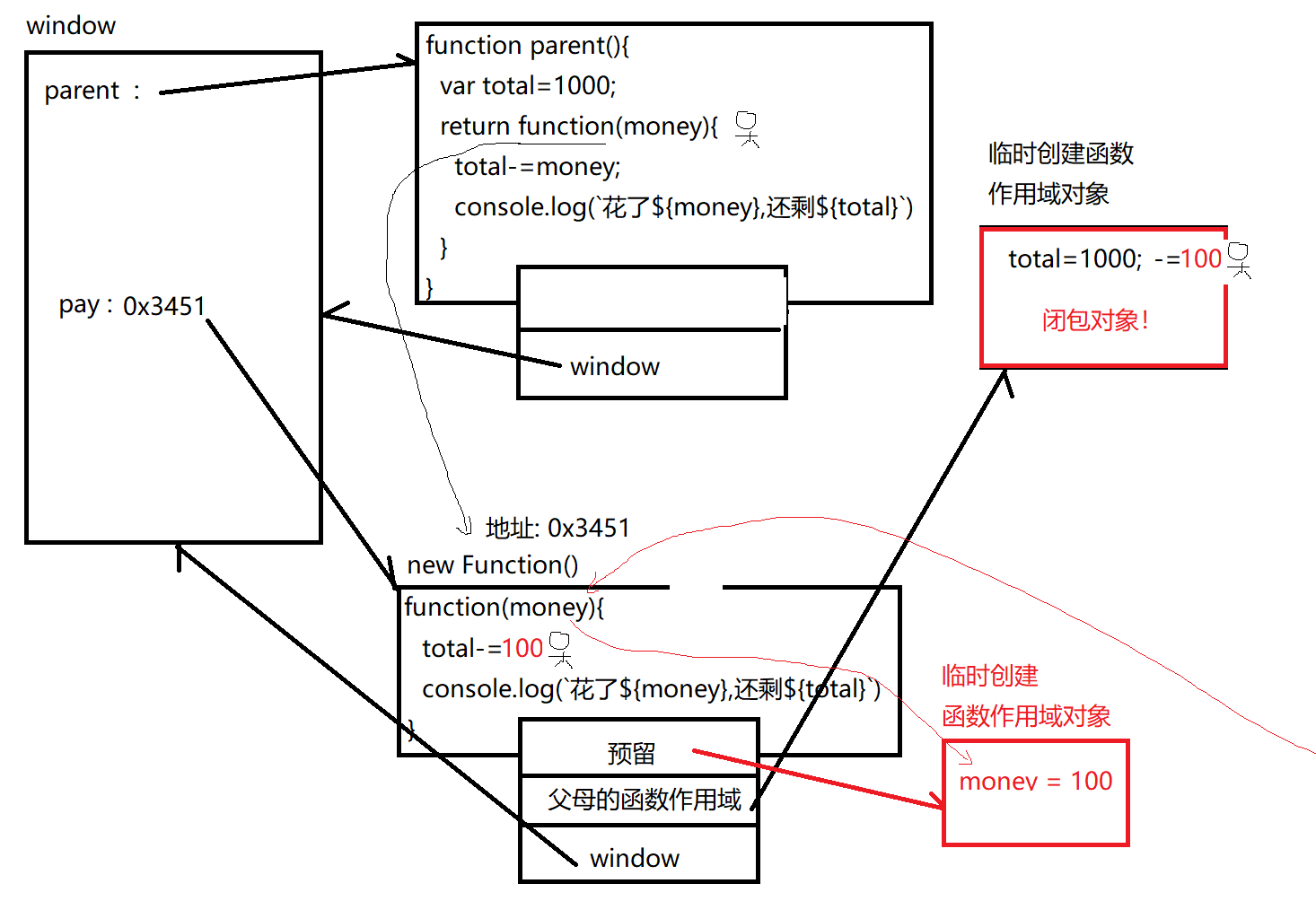
(2). 外层函数将内层函数return到外部

(3). 外部的使用者需要调用外层函数，才能获得返回出来的内层函数对象。

5. 示例: 定义一个函数，为小孩儿保管压岁钱

|  |
| --- |
| //定义一个函数为小孩儿管理压岁钱  //1. 用外层函数包裹要保护的变量和使用变量的内层函数  function parent(){    var total=1000;    //2. 外层函数将内层函数return到外部来！    return function(money){      total-=money;      console.log(`花了${money},还剩${total}`)    }  }  //3. 别人想用内层函数，必须调用外层函数parent，才能获得返回的内层函数对象  var pay=parent();  //pay: function(money){  //   total-=money;  //   console.log(`花了${money},还剩${total}`)  // }  //别人:  total=0;  pay(100); //花100，剩900  pay(100); //花100，剩800  pay(100); //花100，剩700 |

6. 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返回出来的内层函数引用着，无法释放！就行了闭包对象!



7. 闭包缺点: 闭包比普通的函数多占用一块内存(外层函数的作用域对象)！不容易被释放，容易造成内存泄漏（以为释放了，但是却还存在）

8. 解决: 一旦一个闭包不再使用，应该将保存内层函数对象的变量赋值为null

比如: pay=null

导致小函数对象被释放

导致外层函数的作用域对象一起被释放

结果整个闭包就释放了

总结:

1. 创建正则表达式对象: 2种:

(1). 如果正则表达式是固定不变的，不需要在程序运行时动态生成:

var reg=/正则/ig;

(2). 如果正则表达式不是固定的，需要在程序运行时根据其他内容动态生成：

var reg=new RegExp("正则","ig")

(3). 冲突: 无论用//还是用new RegExp()，只要正则中的内容和关键词或关键标点符号冲突了！都要用\转义！

2. 查找敏感词: 4种:

(1). 查找一个固定的敏感词出现的位置: var i=str.indexOf("敏感词",fromi)

(2). 用正则查找一个敏感词的位置: var i=str.search(/正则/i);

(3). 获得敏感词的内容: 2种:

a. 只获得一个敏感词的内容和位置: var arr=str.match(/正则/i)

b. 获得所有敏感词的内容(不包含位置): var arr=str.match(/正则/ig);

(4). 既获得每个敏感词的内容，又获得每个敏感词的位置:

do{

var arr=reg.exec(str);

if(arr!=null){

console.log(arr)

}

}while(arr!=null);

(5). 返回值:

a. 如果函数返回下标位置i，则找不到返回-1

b. 如果函数返回数组arr，则找不到返回null

3. 替换敏感词: 2种:

(1). 将所有敏感词都替换为统一的新值:

str=str.replace(/正则/ig, "新值");

(2). 根据每次找到的敏感词不同，动态选择不同的新值替换

str=str.replace(/正则/ig, function(kw){

return 根据本次kw接住的敏感词不同，动态返回不同的新值

} )

(3). 删除敏感词: 其实就是将敏感词替换为""

4. 切割字符串:2种

(1). 切割符是固定不变的:

var arr=str.split("固定的切割符");

(2). 切割符不是固定的

var arr=str.split(/正则/i);

(3). 打散字符串为字符的数组:

var arr=str.split("");

5. 验证字符串格式:

var bool=reg.test(/^正则$/)

作业：

1. (必须)看小程序视频复习声明提前:

看小程序->在线->JSCORE->day03: 1. 声明提前和3. 声明提前...

2. (必须)看小程序问题清单实现add函数，能够计算任意多个数值的和

小程序->首页->搜"重载" 结果: 作业: 重载 定义一个函数add...

3. (必须)看小程序问题清单学习作用域笔试题:

小程序->首页->搜"foo" 结果: 笔试题 var a=10 foo...

4. (必须)看小程序视频复习作用域和按值传递:

小程序->在线->JSCORE->day03:

作业: 高频笔试题 按值传递

和

6. 简单 作用域 按值传递

5. (必须)看小程序问题清单实现一个取号机函数:

getNum() 1

getNum() 2

i=0 //假设用i保存当前序号

getNum() 3

getNum() 4

小程序->首页->搜"getNum" 结果: 作业: 闭包 实现一个取号机函数

6. (扩展)看小程序视频学习作用域笔试题:

小程序->在线->JSCORE->day03: 作业: 作用域 笔试题

7. (扩展)看小程序视频学习闭包笔试题

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

8. (扩展)看小程序视频学习闭包笔试题:

小程序->在线->JSCORE->day03: 作业: 闭包 高频鄙视题 funs