准备: 微信小程序: WEB问题速查 小恐龙图标

1. 历届师兄师姐在今天课程中曾经问过的高频的经典的问题清单和解答: 小程序->首页->JSCORE->day03

自己可能没有发现问题，但是想知道别的同学都问过哪些问题

2. 今日课程对应的重点案例的视频以及鄙视面试所需的扩展知识视频：小程序->在线->JSCORE->day03

如果因为某些原因错过直播讲解，不用着急！直接看小程序中视频的回看即可。

如果tmooc不能用了，立刻看小程序视频，小程序视频比tmooc要全，甚至有扩展的新知识。

如果学有余力，将来有能力拿更高薪资，小程序中所有视频必须都要看。但是不必现在都看完。只要在笔试面试前都看完即可！

问题: 个人版小程序有功能限制：禁止在线播放视频

解决: 长按视频连接地址，完整复制视频连接地址和验证码到浏览器打开。通过百度网盘下载后观看。

3. 今日课程中包含的英语单词的翻译：小程序->单词->JSCORE->day03

英文不太好的同学，可以提前看一下第二天需要用到哪些英文单词。

约定: 2件事:

1. 第三阶段禁止上课跟着敲笔记和敲代码！谁抄笔记抄的勤，谁抄代码抄的勤，谁学的越烂！

2. 每个完整功能的案例，至少写三遍:

(1). 连代码+注释抄一遍 —— 明白这个例子是要做什么事儿

(2). 保留注释，自己试着看着注释，将代码填回来 —— 复习函数和对象

(3). 删除所有代码和注释，自己试着用自己的话，把注释写回来，再翻译为代码！—— 只有自己可以写出人话的注释，这个例子才算自己会！

如果没有问题的同学: （享受vip一对一答疑的时间还剩19天）

1. 上课时那句话没听懂，笔记中那句话看不懂，案例中那句话看不懂，立刻问！

2. 小程序问题清单中哪个问题或解答看不懂，立刻问！

3. 小程序视频中哪个视频看不懂，立刻问

4. 个人项目中哪个功能不会做，立刻问

5. 平时刷笔试题时，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！

从今天开始，未来的6天时间，是前端最重要的6天知识！—— 打通任督二脉！

正课: Function

1. 回顾函数:

2. 重载

3. 匿名函数

4. 作用域和作用域链

5. \*\*\*\*\*闭包:

一. 回顾函数：

1. 什么是: 程序中保存一段可重用的代码段的程序结构，再起一个名字

2. 何时: 只要一项任务的代码，可能被反复使用，都要封装为函数。

3. 如何定义: 声明方式:

function 函数名(形参变量列表){

要反复执行的代码段

return 返回值

}

4. 调用函数:

var 变量=函数名(实参值列表)

强调: 一个函数！只定义，不调用！是不执行的！

5. 示例: 定义一个做菜的函数，可以传入食材，并返回做好的饭菜

|  |
| --- |
| <script>  //想定义一个会做菜的函数  function cook(cai,dan,mian){ //只会做固定一种菜: 西红柿鸡蛋面  console.log(`炒${cai}`);  console.log(`炒${dan}`);  console.log(`${cai}和${dan}一起炒`);  console.log(`煮${mian}`);  return `香喷喷的${cai}${dan}${mian}`  };  //周一:  var wan=cook("西红柿","鸡蛋","面");  console.log(`一碗${wan}`);  //周二:  var wan=cook("辣条","螺蛳","粉儿");  console.log(`一碗${wan}`);  </script>  运行结果:  炒西红柿  炒鸡蛋  西红柿和鸡蛋一起炒  煮面  一碗香喷喷的西红柿鸡蛋面  炒辣条  炒螺蛳  辣条和螺蛳一起炒  煮粉儿  一碗香喷喷的辣条螺蛳粉儿 |

6. 鄙视: js中定义函数共有几种方式，差别是什么？3种:

对应小视频: 小程序->在线->JSCORE->day03 2. 3. 两个视频 三遍起

(1). 声明方式:

function 函数名(形参变量列表){

要反复执行的代码段

return 返回值

}

问题: 会被声明提前:

在程序开始执行前

js引擎会先将var声明的变量和function声明的函数，

提前到当前作用域的顶部，优先集中创建。

但是，赋值留在原地

缺点: 打乱程序的执行顺序，产生歧义

(2). 赋值方式:

a. var函数名=function (形参变量列表){

要反复执行的代码段

return 返回值

}

b. 优点: 不会被声明提前，保持了程序正常的执行顺序，不会产生歧义

c. 揭示了js中函数的本质:

1). js中函数名也只是一个普通的变量名而已（几乎和var 变量没差别）

2). 函数其实是一个对象

3). 函数名变量保存着函数对象

d. 示例: 声明提前笔试题

|  |
| --- |
| <script>  // function fun(){ console.log(1) }  // fun(); //2  // function fun(){ console.log(2) }  // fun(); //2  //就想输出1和2  var fun=function(){ console.log(1) }  fun(); //1  var fun=function(){ console.log(2) }  fun(); //2  </script>  运行效果：  1  2 |

(3). 用new创建: (因为格式极其晦涩，极少使用，了解即可)

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

b. 更揭示了函数的本质: 函数其实也是一个对象，函数名仅仅是一个变量。

c. 总结: function其实是new Function()的简写！

等效于

d. 示例: 验证函数的本质也是一个对象

|  |
| --- |
| <script>  //function add(a,b){//除了会被声明提前之外，和var fun=function()无差别  //var add=function(a,b){ return a+b; } //function其实是new Function()的简写  var add=new Function("a","b","return a+b");//最底层本质  console.log(add(3,5));//8  var sum=add;  //sum=function(a,b){return a+b};  console.log(sum(3,5));//8  var a=function(){ console.log(1) };//new Function() 被分配内存地址  var b=function(){ console.log(1) };//new Function() 被分配另一个新的内存地址  console.log(a==b);//false  //==比较两个对象时，不再隐式转换，而直接比较两个对象的地址是否相同。  </script>  运行效果：  8  8  false |

二. 重载(overload):

对应小视频: 小程序->在线->JSCORE->day03 4. 重载 overload pay

1. 什么是: 相同函数名，不同形参列表的多个函数，在调用时，可根据传入的实参值不同，自动选择匹配的函数版本来执行不同的逻辑。

2. 为什么: 减少函数名的个数，减轻调用者的负担！

3. 何时: 只要一件事，希望根据传入实参的不同，执行不同的逻辑时，都首选重载方式实现

4. 如何:

(1). 问题: js不支持传统的重载写法：js中不允许多个同名函数同时存在。如果同时存在，最后定义的同名函数会覆盖之前定义的所有同名函数

(2). 解决: 借助arguments对象来变通实现重载的效果

a. 什么是arguments对象:

1). 每个函数中自带的 —— 不用我们手工创建，就可直接使用！

2). 专门接收传入函数的所有实参值的 —— 作用/内容

3). 类数组对象 —— 长的像数组的对象

i. 相同点: ①下标, ②.length, ③遍历

ii. 不同点: 类型不同。类数组对象是Object家的孩子，类数组对象无法使用数组家的函数。

b. 如何实现重载:

1). 只定义一个函数，其中包含多种要执行的逻辑

2). 但是不要写任何形参变量，将来js中的函数即使一个形参也不写，arguments对象也可以自动替我们接住传入的所有实参值。

3). 可通过判断arguments中实参值的个数和实参值的内容不同来动态决定本次调用应该选择执行哪种逻辑

4). 可使用arguments[下标]方式获得本次调用时传入的第i个实参值

5. 示例: 定义一个支付函数pay，支持三种支付方式

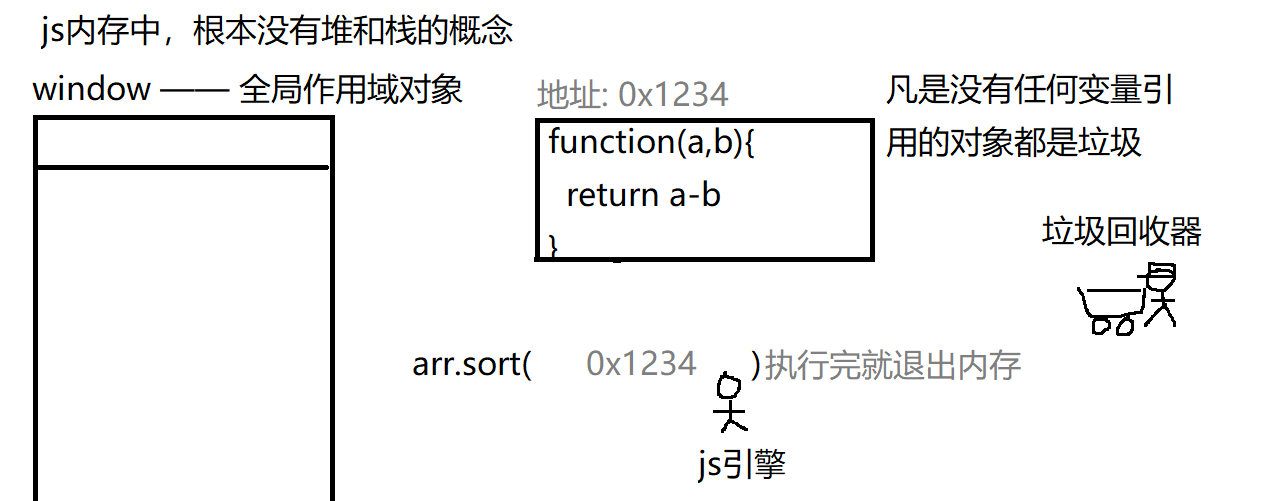
|  |
| --- |
| <script>  //定义一组支付的函数: 3种可能  //1. 手机支付: 不需要给收款员提供任何东西，就可手机支付  //2. 现金支付: 需要给收款员一笔现金  //3. 刷卡支付: 需要提供卡号和密码  //pay("6553 1234","123456");  //pay( 100 );  //pay( ↓ );  function pay( ){  // ↓ ↓ ↓  //arguments{ }.length  // 0 1 2...  //如果没有传入参数，就手机支付  if(arguments.length==0){  console.log(`手机支付...`);  }else if(arguments.length==1){//如果传入一个参数，就现金支付  console.log(`现金支付...收您${arguments[0]}元`)  }else{//否则就刷卡支付  console.log(`刷卡支付...从您卡号${arguments[0]}，扣款成功!`)  }  //本例: 只考虑最多2个实参值的情况。  }  //想手机支付:  pay();  //想现金支付  pay(100);  //想刷卡支付  pay("6553 1234","123456");  </script>  运行效果：  手机支付...  现金支付...收您100元  刷卡支付...从您卡号6553 1234，扣款成功! |

三. 匿名函数:

对应小程序视频: 小程序->在线->JSCORE->day03 5 匿名函数 防止全局污染

1. 什么是: 在定义函数时，不指定函数名的函数

2. 为什么: 匿名函数比有名称的函数更节约内存 —— 没有函数名，就没有变量引用，用完，内存就释放了！



3. 何时: 如果一个函数只使用一次时，都用匿名函数

4. 如何: 2种情况

(1). 回调函数:

a. 什么是: 我们定义的函数，但是我们自己不调用，而是交给另一个函数，由另一个函数根据需要自动调用。

b. 比如:

1). 数组排序: var arr=[12,3,123,23,1,2]; arr.sort(function(a,b){return a-b});

2). 高级替换:

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

c. 总结: 今后绝大多数回调函数，都优先使用匿名函数来定义

(2). 匿名函数自调:

a. 什么是匿名函数自调: 定义一个匿名函数后，立刻调用该函数，调用后立刻释放！

b. 为什么: 避免使用全局变量，造成全局污染！

c. 何时: 今后，所有js程序，都要放在一个匿名函数自调中包裹起来！

d. 如何: 标准写法: (function(){ ... })()

e. 说明: 用不用匿名函数自调，不影响功能的执行！只改变变量的性质而已。

f. 示例: 使用匿名函数自调避免使用全局变量

|  |
| --- |
| <script>  (function(){  //在页面开始加载时，弹出提示框: 开始加载页面内容,at:时间  //start变量就变成了局部变量  var start=new Date();//获得当前系统时间  alert(`开始加载页面内容,at:${start.toLocaleString()}`);  //复习第一阶段Date相关内容  })();//所有局部变量在匿名函数自调后，会被自动释放！因为局部变量没有保存在全局。（稍后讲）  </script>  <script>  //在页面加载完成是，弹出提示框：页面加载完成,at:时间  var end=new Date();//获得当前系统时间  alert(`页面加载完成,at:${end.toLocaleString()}`)  //复习第一阶段Date相关内容  //按理说: start变量和end变量在页面加载完之后，使命就结束了！  //但是: end变量却还遗留在window中，不会被释放！  </script>  运行效果： |

四. 作用域(scope)和作用域链:

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

1. 什么是作用域: 2种解释:

(1). 用法: 作用域是一个变量的可用范围

(2). 本质: 作用域是保存变量的特殊对象

2. 为什么: 避免不同范围的变量之间互相干扰

3. js中包括2种作用域:

(1). 全局作用域：

a. 什么是: 专门保存全局变量的特殊对象——window

b. 特点:

1). 在程序的任何一个角落都可以随时访问到全局作用域。

2). 全局作用域不会自动释放！除非关闭浏览器窗口。

c. 什么是全局变量: 保存在全局作用域中，可被程序任何角落随时访问到的变量

d. 全局变量特点: 2个

1). 随处可用！——缺点: 极易被污染！

2). 可反复使用 ——优点

(2). 函数作用域：

a. 什么是: 专门保存局部变量的特殊对象——？

b. 特点:

1). 当函数调用时才临时创建函数作用域对象，函数调用后函数作用域对象被自动释放！下次再调用同一个函数，也会重新创建一个全新的函数作用域对象.

2). 只有函数调用时，函数内的代码才能使用函数作用域对象。

c. 什么是局部变量: 保存在函数作用域当中，仅函数内的代码可以访问的变量

d. 局部变量特点:

1). 仅函数内可用！——优点: 不会被污染！

2). 不可重用 ——缺点

4. 作用域链:

(1). 什么是作用域链: 由多级作用域，逐级引用，形成的链式结构

(2). 函数从定义到调用，再到调用完成所经历的过程:

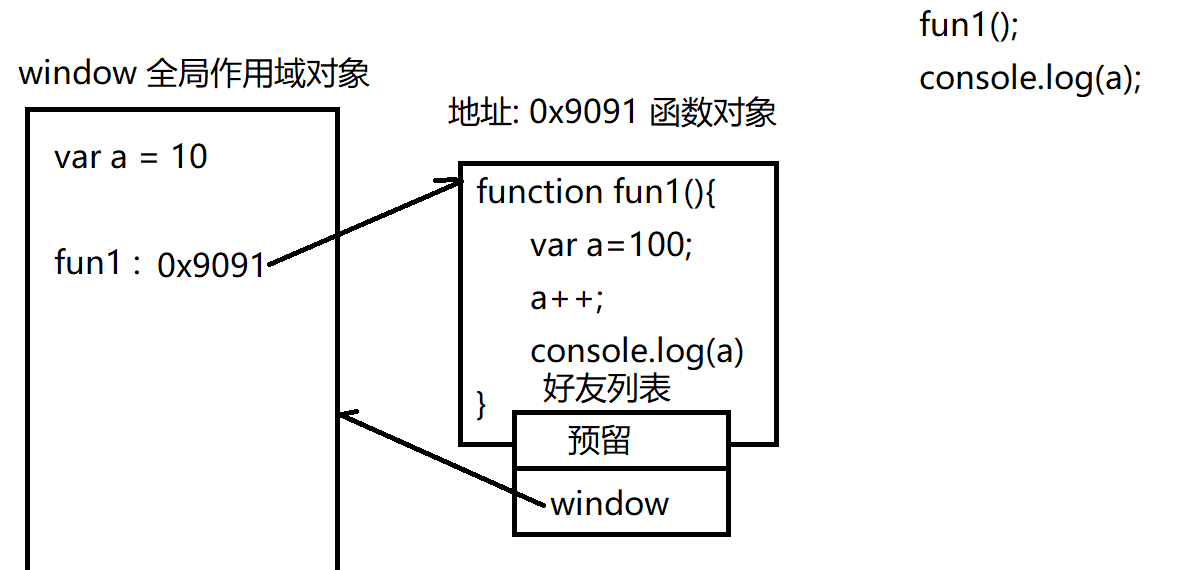
a. 定义函数时:

1). 创建函数对象保存函数内的代码片段

2). 每个函数对象身上都有一个"好友列表"，记录了每个函数可用的所有作用域对象，普通函数的好友列表包含2个格子:

i. 离函数自己近的格子，暂时预留

ii. 离函数自己稍远的格子存window。



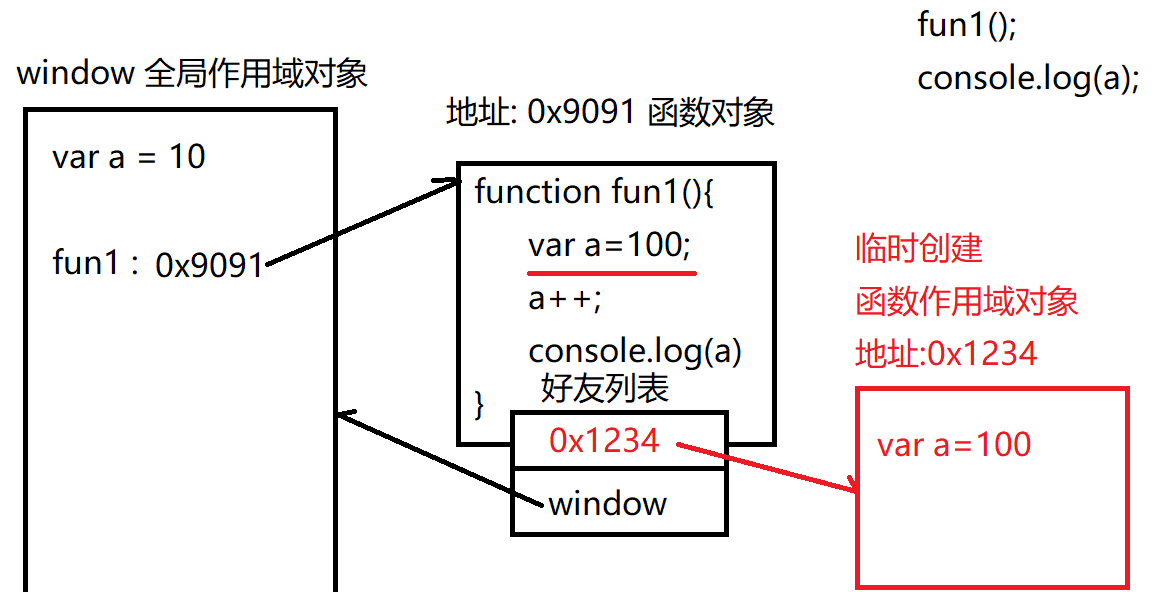
b. 调用函数时:

1). 临时创建本次函数调用的函数作用域对象

2). 将函数体中定义的局部变量临时创建到函数作用域对象中保存

强调: 局部变量有2种: ①在函数内var出的变量; ②形参也属于局部变量

3). 将临时创建的函数作用域对象的地址，临时保存到函数的好友列表中离函数自己近的格子中.



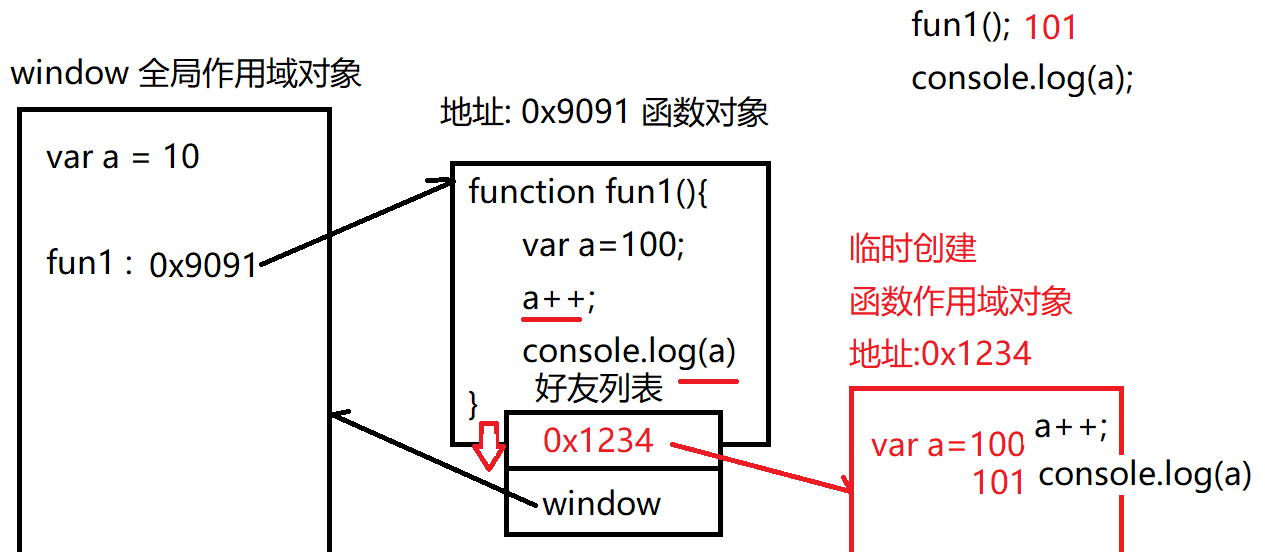
c. 执行函数过程中:

1). 如果函数体中用到了某个变量，函数都会去好友列表中查找该变量

2). 按顺序: 就近原则: 先局部，后全局。

i. 如果局部有该变量，则优先使用局部变量

ii. 如果局部没有该变量，才被迫去全局作用域查找使用。



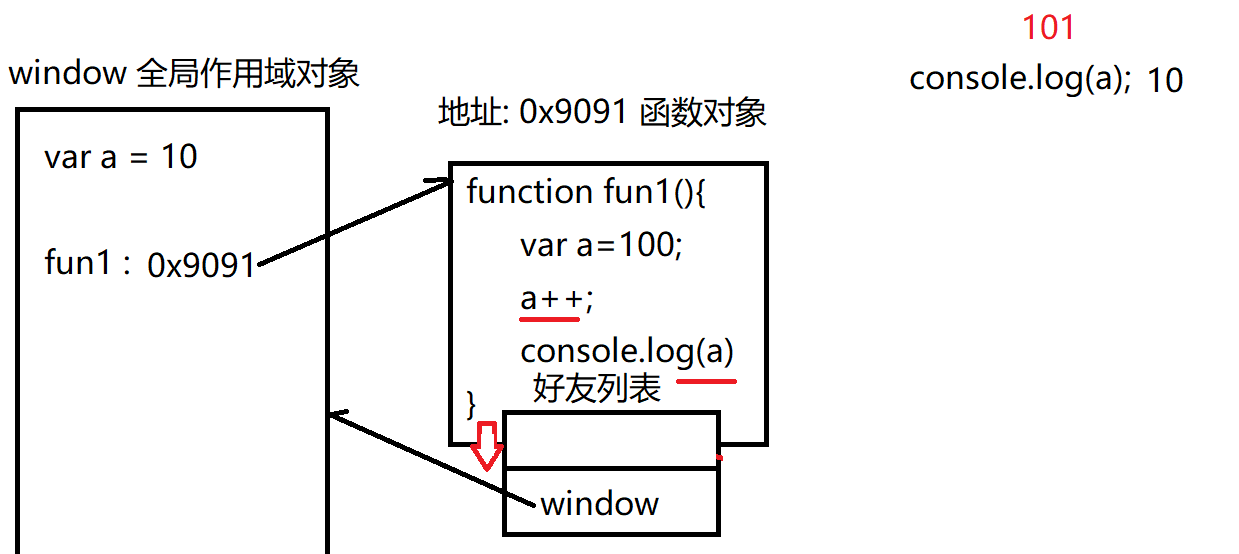
d. 函数调用后:

1). 好友列表中离自己近的格子会清空z

2). 导致临时创建的函数作用域对象被释放

但是保存着函数定义的函数对象是不会被释放的！

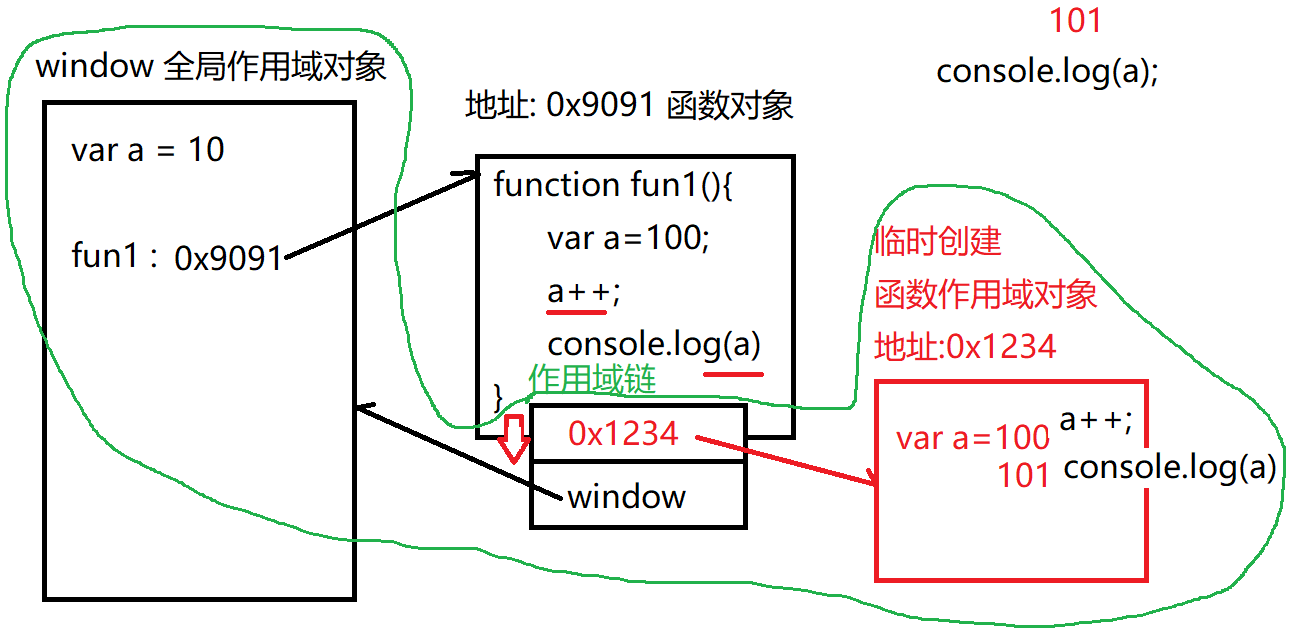
3). 导致函数本次使用的所有局部变量一同被释放——局部变量不可重用！



(3). 以上过程中函数对象的"好友列表"，学名就叫作用域链:

a. 保存着: 一个函数可用的所有的作用域以及变量

b. 控制着: 变量的使用顺序: 先局部后全局——就近原则



五. 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: 2个概念

(1). 专门重用一个变量，又保护变量不被篡改一种编程方法

(2). 本质: 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

2. 为什么: 全局变量和局部变量都有不可兼得的优缺点:

(1). 全局变量: 优点: 可重用, 缺点: 随处可用，极易被污染

(2). 局部变量: 优点: 只能在函数内使用，不会被污染, 缺点: 不可重用

3. 何时: 今后只要想重用一个变量，但是又要保护变量不被篡改，都要用闭包！

4. 如何: 3步 —— 判断闭包的三个特点

(1). 用外层函数包裹要保护的变量和使用该变量的函数

(2). 外层函数将内层函数return到外层函数外部，让外部可以使用

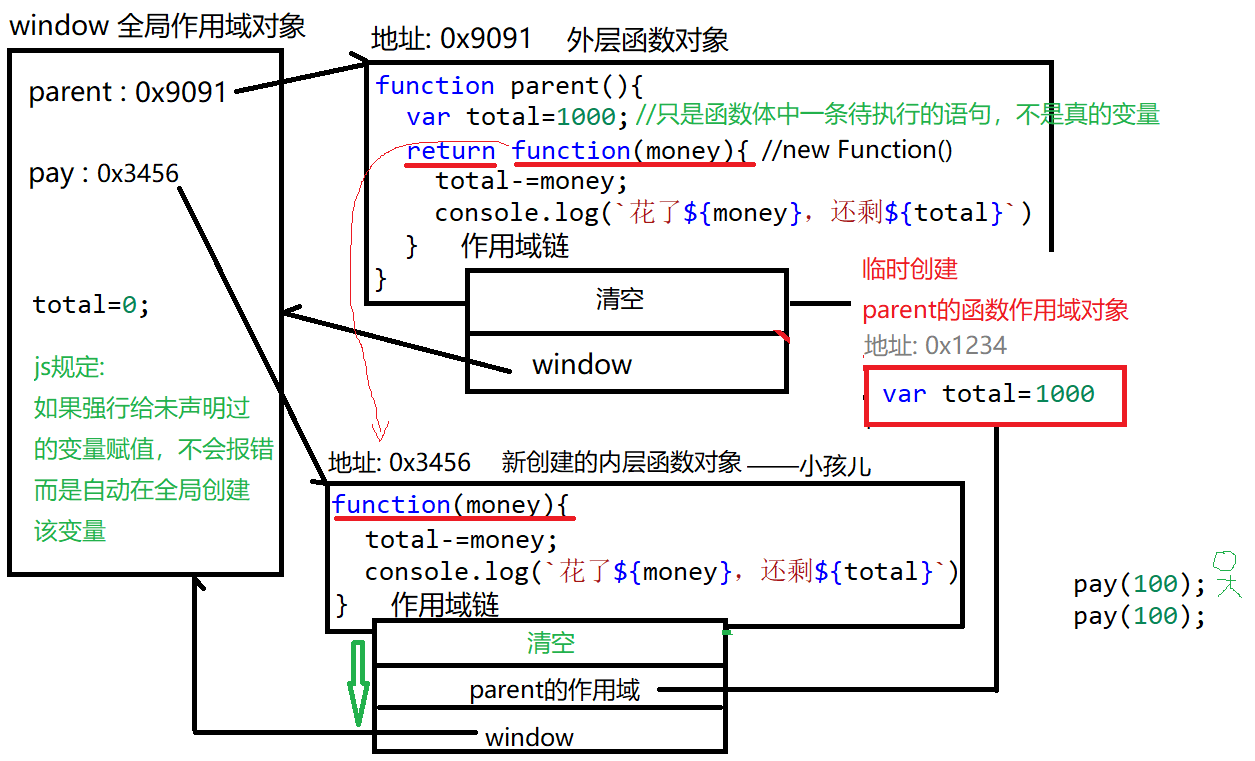
说明: 因为内层函数被返回到外部之后，一定会被外边的变量接住。所以，内层函数没必要起名！就算起了名字，将来外部也不一定使用这个名字！

(3). 外部要想使用内层函数，必须先调用外层函数，来获得返回的内层函数保存到变量中，才能反复使用！

5. 示例: 定义一个管理压岁钱的函数，要求压岁钱总数不能被外界随意篡改

|  |
| --- |
| <script>  //需求: 定义一个函数帮小孩儿管理压岁钱  //假设小孩儿有1000块钱压岁钱  //1. 用外层函数包裹要保护的变量和内层函数  function parent(){//父母帮小孩儿保管压岁钱  var total=1000;//——压岁钱  //2. 外层函数将内层函数return到外层函数外部，让外部可以使用  return function(money){//——小孩儿  //从总钱数中-本次花的钱  total-=money;  //输出花了多少，还剩多少余额  console.log(`花了${money}，还剩${total}`)  }  }  //3. 外部要想使用内层函数，必须先调用外层函数，来获得返回的内层函数保存到变量中，才能反复使用！  var pay=parent();  //pay:function(money){ ... }  //测试:  pay(100);  total=0;//尝试篡改total  pay(100);  pay(100);  </script>  运行结果:  花了100，还剩900  花了100，还剩800  花了100，还剩700 |

6. 一句话闭包形成的原因: 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，就形成了闭包！



7. 闭包缺点: 比一般的函数多占用了一块内存空间——外层函数的函数作用域对象

8. 解决: 一旦一个闭包不再使用，应该立刻释放:

保存内层函数的变量=null

导致内层函数对象被释放

导致外层函数的作用域对象(闭包)一同被释放

约定: 看视频时，必须跟着亲手画图(在操作系统的画图工具里)！至少画三遍以上！

总结:

1. String正则相关的函数: 3件事

(1). 查找关键词: 4种情况

a. 查找一个固定的关键词出现的位置:

var i=str.indexOf("关键词", fromi)

b. 用正则模糊查找多种敏感词的位置:

var i=str.search(/正则/i)

c. 查找关键词的内容:

1). 如果只查找一个关键词的内容和位置：

var arr=str.match(/正则/i)

2). 如果查找所有关键词的内容，不关心位置:

var arr=str.match(/正则/ig)

d. 既查找每个关键词的内容，又查找每个关键词的位置:

reg.exec()

(2). 替换关键词: 2种情况:

a. 简单替换: str=str.replace(/正则/ig, "新值")

b. 高级替换: str=str.replace(/正则/ig, function(keyword){

return 根据keyword不同，动态选择不同的新值返回并替换到字符串

})

c. 删除关键词: str=str.replace(/正则/ig, "")

(3). 切割字符串: 2种情况:

a. 简单切割: var arr=str.split("切割符")

b. 复杂切割: var arr=str.split(/正则/i)

c. 固定套路: var arr=str.split("")//将字符串打散为字符数组

2. RegExp对象:

(1). 创建:

a. 如果正则表达式是动态拼接而成的: var reg=new RegExp("正则","ig")

b. 如果正则表达式固定不变的: var reg=/正则/ig

(2). 验证格式:

var reg=/^正则$/;

var bool=reg.test(str)

(3). 既查找每个敏感词内容，又查找每个敏感词位置:

var reg=/正则/g

do{

var arr=reg.exec(str);

...

}while(arr!=null); //只有本次找到了敏感词才有必要继续执行下一次查找

上午案例对应视频:

小程序->在线->JSCORE->day03 2. 3. 两个视频 三遍起

小程序->在线->JSCORE->day03 4. 重载 overload pay

小程序->在线->JSCORE->day03 5 匿名函数 防止全局污染

小程序->在线->JSCORE->day03 7. 原理 作用域和作用域链

小程序->在线->JSCORE->day03 8. 闭包 closure 压岁钱

中午作业: 小程序->在线->JSCORE->day03 6 简单 作用域 按值传递

作业:

1. 复习今日问题清单(小程序->首页->JSCORE->DAY03)，预习下节课问题清单(小程序->首页->JSCORE->DAY04)

2. 看小程序问题清单学习使用重载实现规定的效果：

小程序->首页 搜: add， 作业: 重载 定义一个函数add...

3. 看小程序问题清单学习使用闭包实现固定的功能

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

4. 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

5. 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...

6. 看小程序视频学习预习用{}创建对象和this

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