闭包

JS 中的闭包是什么？

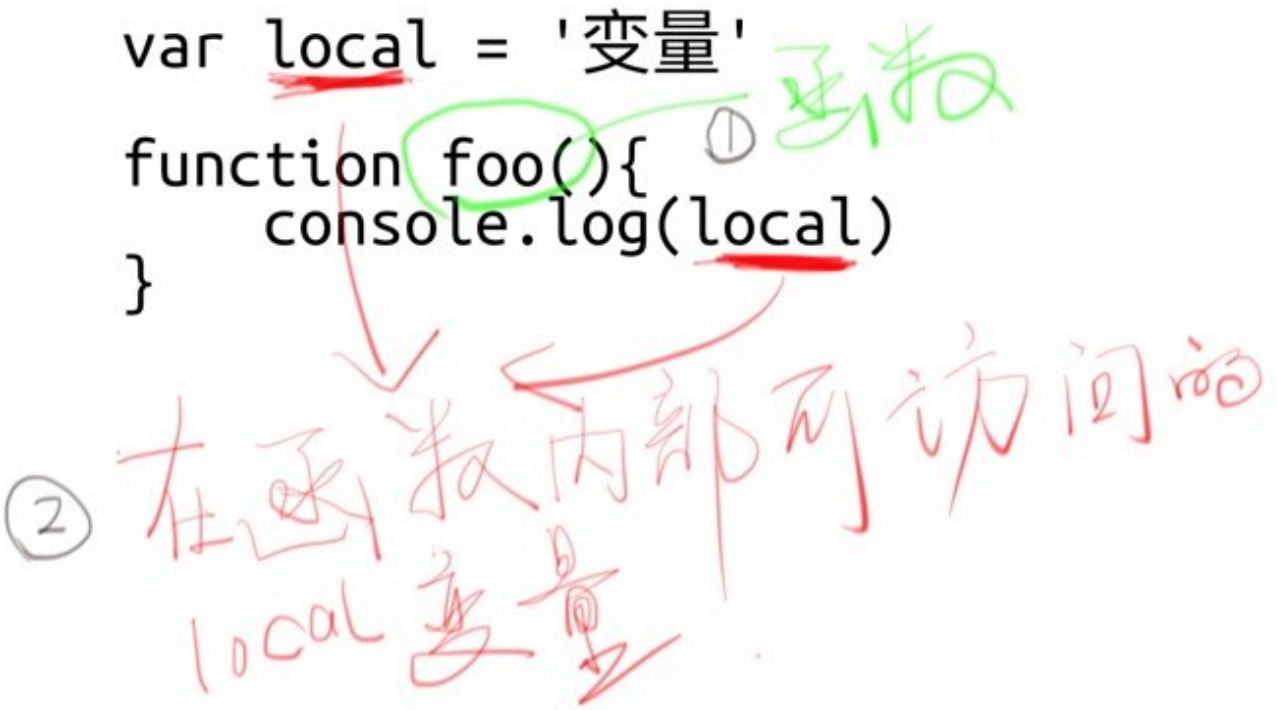
大名鼎鼎的闭包！这一题终于来了，面试必问。

请用自己的话简述

1.什么是「闭包」？

2.「闭包」的作用是什么？

首先来简述什么是闭包



上面三行代码在一个立即执行函数中。

三行代码中，有一个局部变量 local，有一个函数 foo，foo 里面可以访问到 local 变量。

好了这就是一个闭包：
「函数」和「函数内部能访问到的变量」（也叫环境）的总和，就是一个闭包。

就这么简单。

有的同学就疑惑了，闭包这么简单么？

「我听说闭包是需要函数套函数，然后 return 一个函数的呀！」

比如这样：

```
function foo(){
  var local = 1
  function bar(){
    local++
    return local
  }
  return bar
}

var func = foo()
func()
```

这里面确实有闭包，local 变量和 bar 函数就组成了一个闭包（Closure）。

为什么要函数套函数呢？

是因为需要局部变量，所以才把 local 放在一个函数里，如果不把 local 放在一个函数里，local 就是一个全局变量了，达不到使用闭包的目的——隐藏变量（等会会讲）。

这也是为什么我上面要说「运行在一个立即执行函数中」。

有些人看到「闭包」这个名字，就一定觉得要用什么包起来才行。其实这是翻译问题，闭包的原文是 Closure，跟「包」没有任何关系。

所以函数套函数只是为了造出一个局部变量，跟闭包无关。

为什么要 return bar 呢？
因为如果不 return，你就无法使用这个闭包。把 return bar 改成 window.bar = bar 也是一样的，只要让外面可以访问到这个 bar 函数就行了。

所以 return bar 只是为了 bar 能被使用，也跟闭包无关。

闭包的作用
闭包常常用来「间接访问一个变量」。换句话说，「隐藏一个变量」。

假设我们在做一个游戏，在写其中关于「还剩几条命」的代码。

如果不用闭包，你可以直接用一个全局变量：

```
window.lives = 30 // 还有三十条命
```

这样看起来很不妥。万一不小心把这个值改成 -1 了怎么办。所以我们不能让别人「直接访问」这个变量。怎么办呢？

用局部变量。

但是用局部变量别人又访问不到，怎么办呢？

暴露一个访问器（函数），让别人可以「间接访问」。

代码如下：

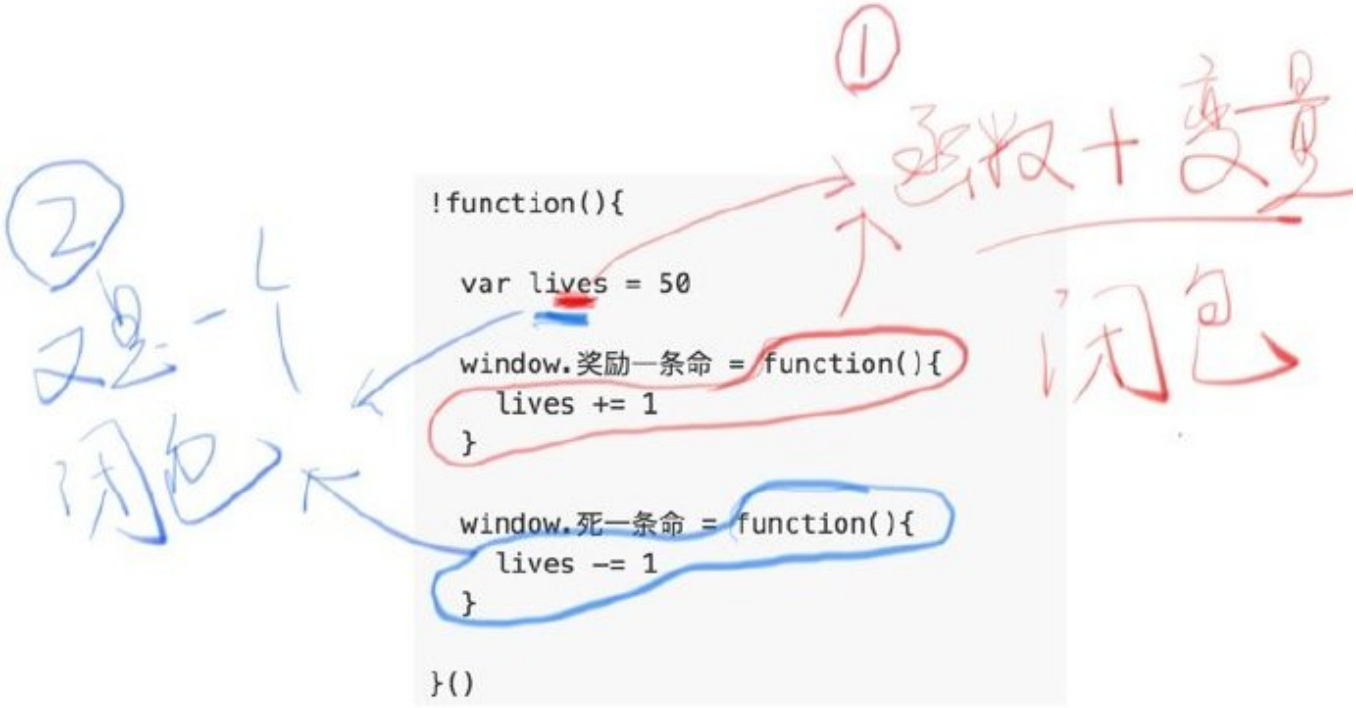
```
!function(){

  var lives = 50

  window.奖励一条命 = function(){
    lives += 1
  }

  window.死一条命 = function(){
    lives -= 1
  }
```

```
12.
13.   }()
```

简明起见，我用了中文

那么在其他的 JS 文件，就可以使用 window.奖励一条命() 来涨命，使用 window.死一条命() 来让角色掉一条命。

看到闭包在哪了吗？

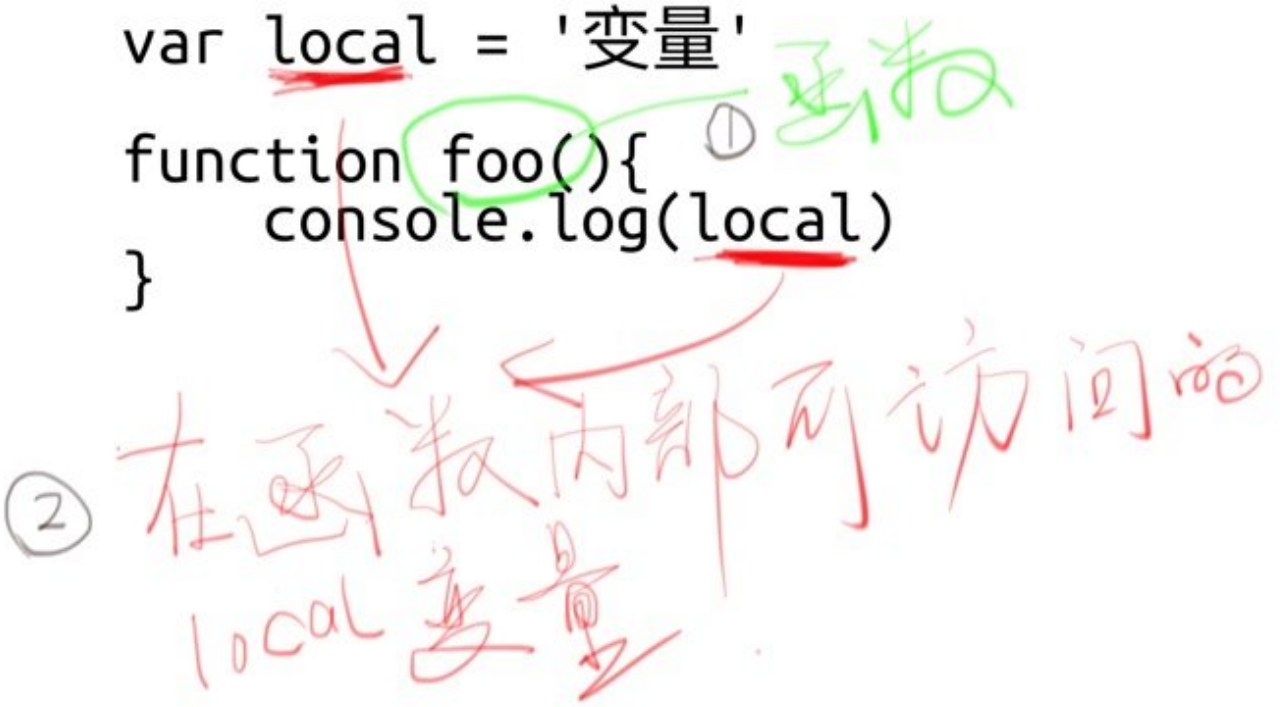


闭包到底是什么？

同学们可以百度搜索看一下哦！！！

五年前，我也被这个问题困扰，于是去搜了 stackoverflow 并总结下来。你在百度搜闭包，详细了解一下。当时我还是新手，一直不理解为什么大家口中的闭包这么模糊、这么琢磨不定呢。

我们重新来审视一下闭包的代码：



第一句是变量声明，第二句是函数声明，第三句是 console.log。

每一句我都学过，为什么合起来我就看不出来是闭包？

我告诉你答案，你根本不需要知道闭包这个概念，一样可以使用闭包！

闭包是 JS 函数作用域的副产品。

换句话说，正是由于 JS 的函数内部可以使用函数外部的变量，所以这段代码正好符合了闭包的定义。而不是 JS 故意要使用闭包。

很多编程语言也支持闭包，另外有一些语言则不支持闭包。

只要你懂了 JS 的作用域，你自然而然就懂了闭包，即使你不知道那就是闭包！

所谓闭包的作用

如果我们在写代码时，根本就不知道闭包，只是按照自己的意图写，最后，发现满足了闭包的定义。

那么请问，这算是闭包的作用吗？

这个问题，留给你思考。

关于闭包的谣言

闭包会造成内存泄露？

错。

说这话的人根本不知道什么是内存泄露。内存泄露是指你用不到（访问不到）的变量，依然占居着内存空间，不能被再次利用起来。

闭包里面的变量明明就是我们需要的变量（lives），凭什么说是内存泄露？

这个谣言是如何来的？

因为 IE。IE 有 bug，IE 在我们使用完闭包之后，依然回收不了闭包里面引用的变量。

这是 IE 的问题，不是闭包的问题。

一个小经验

编程界崇尚以简洁优雅为美，很多时候如果你觉得一个概念很复杂，那么很可能是你理解错了。