03 闭包—探索词法作用域模型

更新时间：2020-05-13 09:57:11

> 没有智慧的头脑，就像没有蜡烛的灯笼。——托尔斯泰

我们说过，作用域本质是一套规则。如果说上一节，我们是把这“规则”的内容，从头到尾给大家梳理清楚了。那么这节，我们要探讨的就是这“规则”的成因。

本节为相对较为高阶的内容，由各位同学根据自己当前情况选学。如果同学目前处在入门 / 初级阶段，没有晋级大厂 / 中高级工程师的强愿望，那么建议在有限的时间里，优先阅读下一篇《闭包的应用与真题集中解析》，时间投入回报比会更高。

词法作用域和动态作用域

相信很多同学可能看到标题已经懵了哈 —— 作用域就作用域，“词法”、“动态” 这些陌生的前缀是啥玩意儿？

事实上，当我们在 JavaScript 语言的范畴里讨论“作用域”这个概念的时候，确实不需要区分它是“词法”还是“动态”，因为我们 **JS 的作用域遵循的就是词法作用域模型**。当面试官抛出“词法作用域 ”这个概念的时候，完全不用慌，它指的就是你最最熟悉的 JS 作用域。

但是站在语言的层面来看，作用域其实有两种主要的工作模型：

- 词法作用域：也称为静态作用域。这是最普遍的一种作用域模型，也是我们学习的重点
- 动态作用域：相对“冷门”，但确实有一些语言采纳的是动态作用域，如：Bash 脚本、Perl 等

想要理解词法作用域本身，我们就不得不从 JS 的框框里跳出来，把它和它的对立面“动态作用域 ”放在一起来看。为了使两者的概念更加直观，我们直接来看一段代码：

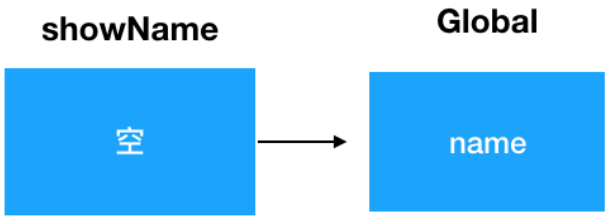
```
var name = 'xiuyan';

function showName() {
    console.log(name);
}

function changeName() {
    var name = 'BigBear';
    showName();
}

changeName();
```

这是一段 JS 代码，基于我们上节对 JS 作用域的学习，不难答出它的运行结果是 ‘xiuyan’。这是因为 JS 采取的就是词法（静态）作用域，这段代码运行过程中，经历了这样的变量定位流程：

- 在 showName 函数的函数作用域内查找是否有局部变量 name
- 发现没找到，于是根据**书写的位置**，查找上层作用域（全局作用域），找到了 name 的值是 xiuyan，所以结果会打印 xiuyan。

此时它的作用域关系示意如下：

运行时的作用域链关系如下：

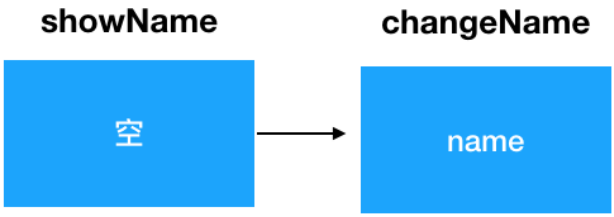


这里我们作用域的划分，是在**书写的过程中（例子中也就是在函数定义的时候，块作用域同理是在代码块定义的时候），根据你把它写在哪个位置来决定的**。像这样划分出来的作用域，遵循的就是词法作用域模型。

那什么是动态作用域呢？动态作用域机制下，同样的一段代码，会发生下面的事情：

- 在 showName 函数的函数作用域内查找是否有局部变量 name
- 发现没找到，于是**沿着函数调用栈、在调用了 showName 的地方**继续找 name。这时大家看看它找到哪去了？是不是就找到 changeName 里去了？ 刚好，changeName 里有一个 name，于是这个 name 就会被引用到 showName 里去。

此时它的作用域链关系示意如下：



所以如果是动态作用域，那么这段代码运行的结果就会是 'BigBear' 了～

我们总结一下，词法作用域和动态作用域的区别其实在于**划分作用域的时机**：

- 词法作用域： 在代码书写的时候完成划分，作用域链沿着它**定义的位置**往外延伸
- 动态作用域： 在代码运行时完成划分，作用域链沿着它的**调用栈**往外延伸

（如果你对"调用栈"这个概念感到陌生或者不舒服，现在也不用着急，在《深入理解 JS 上下文和调用栈》这一节，我们会有更加深入和详细的解析。

修改词法作用域

在相对高阶的前端面试中，有时面试官会抛出这样的问题：如何"欺骗"词法作用域？

大家不要被"欺骗"这个新奇的说法给唬到了，这里"欺骗"就是"改变"的意思。

面试官询问你改变作用域的方法，一般不是真的希望你在写代码的时候去改变作用域规则（这样做往往需要付出性能的代价），而是在摸底，想知道你对词法作用域到底了解到了什么程度。

如何理解“修改”这个动作？JS 遵循词法作用域模型已成定局，难道我还能把它扳成动态作用域不成？别说，还真行。你 JS 不是只在书写阶段对作用域进行划分吗？那么我偏要在运行过程中把你划分好的作用域改掉 —— 到底是谁这么牛？我们请出 eval 和 with：

eval 对作用域的修改

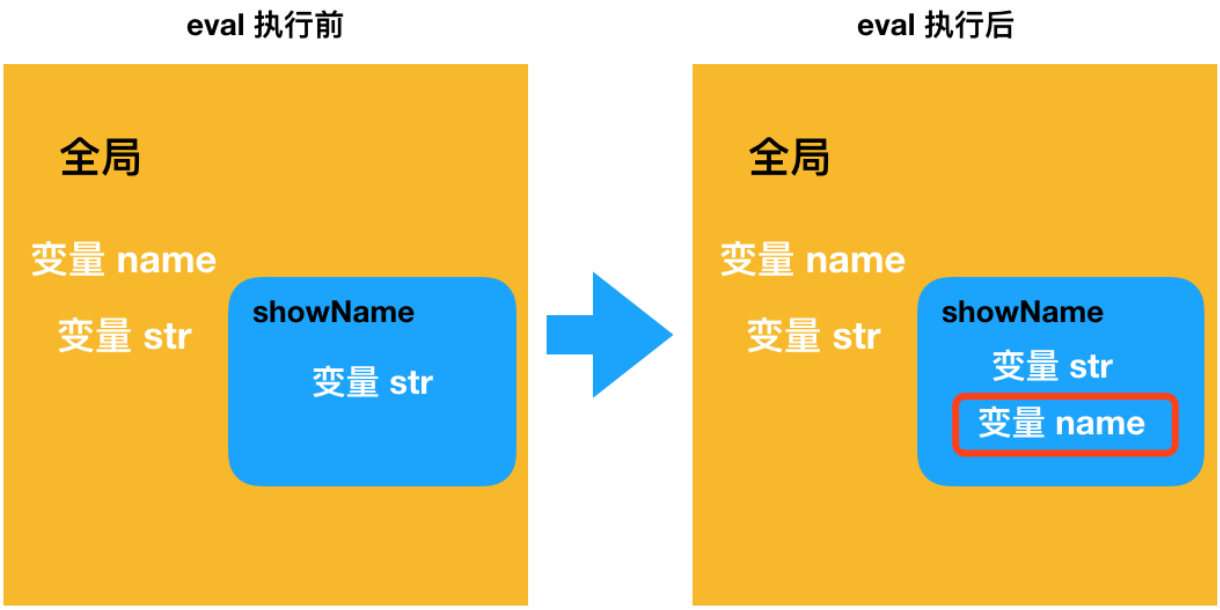
开头我们先来复习一下 eval 函数的用法：

```
function showName(str) {
  eval(str)
  console.log(name)
}

var name = 'xiuyan'
var str = 'var name = "BigBear"'

showName(str) // 输出 BigBear
```

大家知道，eval 函数的入参是一个字符串。当 eval 拿到一个字符串入参后，它会把这段字符串的内容当做一段 js 代码（不管它是不是一段 js 代码），插入自己被调用的那个位置。所以上面这个例子里，被 eval “改造”过后的 showName 函数其实长这样了：

```
function showName(str) {
  var name = 'BigBear'
  console.log(name)
}
```

此时当我们尝试输出 name 的时候， 函数作用域内的 name 已经被 eval 传入的这行代码给修改掉了，所以作用域内 name 的值就从 ‘xiuyan’ 变成了 ‘BigBear’（eval 带来的改变如下图所示）。而这个改变确实只有在 eval (str) 这行代码被执行后才发生 ——eval 在运行时改变了作用域的内容，它成功地“修改”了词法作用域规则约束下在书写阶段就划分好的作用域。



with 对作用域的修改

with 对大家来说可能比 eval 要陌生一些。它的作用就是帮我们“偷懒”，当我们不想重复地写一个对象名作为前缀的时候，with 可以帮到我们：

```
var me = {
  name: 'xiuyan',
  career: 'coder',
  hobbies: ['coding', 'footbal']
}

// 假如我们想输出对象 me 中的变量，没有 with 可能会这样做：
console.log(me.name)
console.log(me.career)
console.log(me.hobbies)

// 但 with 可以帮我们省去写前缀的时间
with(me) {
  console.log(name)
  console.log(career)
  console.log(hobbies)
}
```

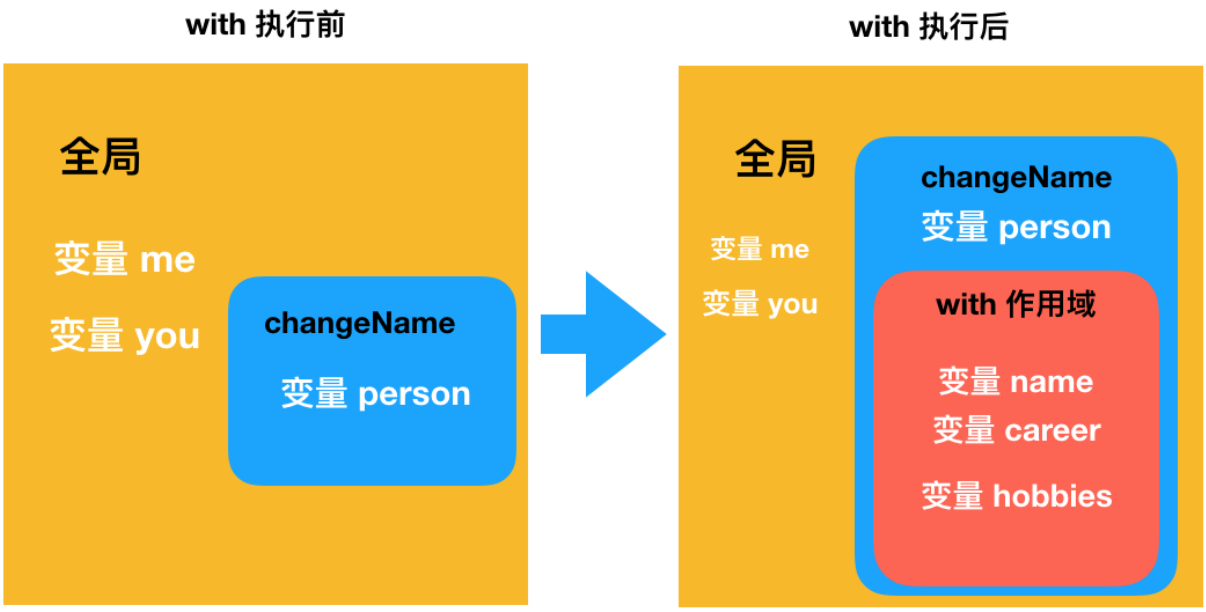
没错， with 就是当我们希望去引用一个对象内的多个属性的时候，一个“偷懒”的办法。

为什么说 with 可以“改变”词法作用域呢？我们再来看一个例子：

```
function changeName(person) {
  with(person) {
    name = 'BigBear'
  }
}

var me = {
  name: 'xiuyan',
  career: 'coder',
  hobbies: ['coding', 'footbal']
}

var you = {
  career: 'product manager'
}

changeName(me)
changeName(you)
console.log(name) // 输出 'BigBear'
```

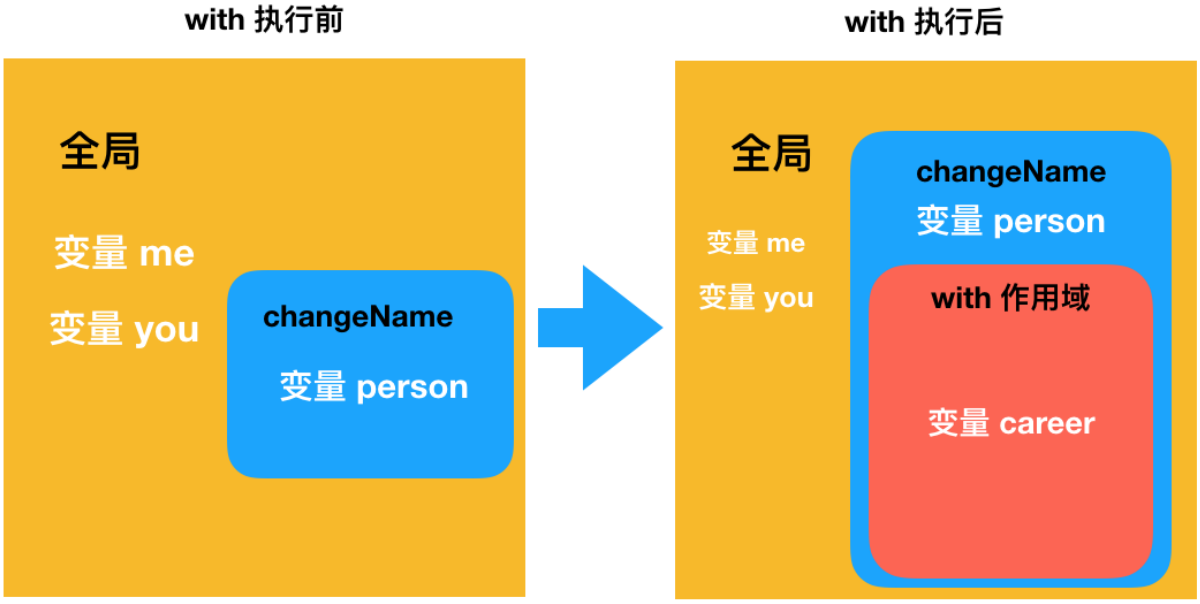
我们惊奇地发现，在执行了两次 changeName 后，竟然多出一个全局变量 name ！

这其实就是 with 在“捣鬼”。其实大家通过使用 with 的过程不难感受出来， with 做的事情其实就是**凭空创建出了一个新的作用域**。比如单说第一次执行 changeName 的过程，它是这样的：

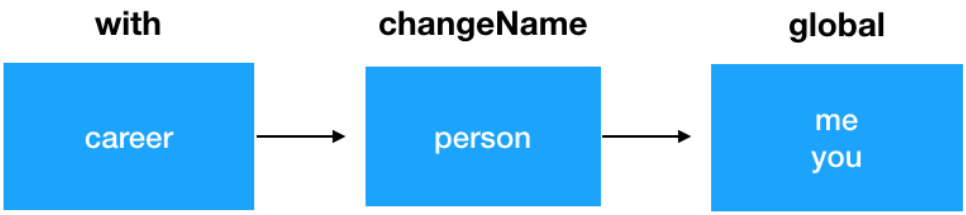


我们把 with 这种创建新作用域的能力代入到两次 changeName 的执行里，就不难理解为什么会多出一个全局 name 了。事情是这样的：

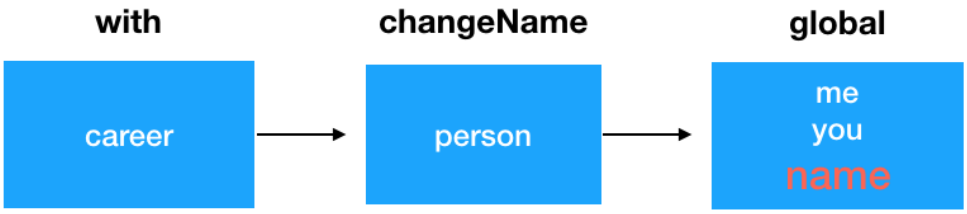
- 第一次 changeName 调用， with 为 me 这个对象创建了一个新作用域，使得我们可以在这个作用域里直接访问 name、 career、hobbies 等对象属性。过程就是我们上面这张图所示。到这里都还没啥毛病。
- 第二次 changeName 调用， with 为 you 这个对象也创建了一个新作用域，使得我们可以在这个作用域里直接访问 career 这个对象属性（如下图）。



结果我们试图访问的竟然是 name —— 一个当前作用域里没有的变量。这时会发生什么？大家注意， with 对作用域的改变，仅仅在于“创建”这个动作。**当这个作用域被创建出来之后，它的查询规则仍然遵循我们词法作用域的查询规则**，所以它本能地“探出头去”、去自己的上层作用域 —— 全局作用域查询 name 了，发现依然找不到（作用域链关系如下图）。



注意我们这时处于非严格模式下，非严格模式下，就算全局作用域里找不到 name，系统也会为你自动在全局作用域创建一个 name （这里如果感到不太理解的小伙伴，需要好好复习一下 JS 基础）。于是 name = ‘BigBear’ 就这么顺利地执行了，全局变量 name 横空出世～（过程如下图）



一切水落石出。我们赶紧总结下 with 改变作用域的方式：

- with 会原地创建一个全新的作用域，这个作用域内的变量集合，其实就是传入 with 的目标对象的属性集合。
- 因为“创建”这个动作，是在 with 代码实际已经被执行后发生的，所以这个新作用域确实是在运行时被添加的， with 因此也实现了对书写阶段就划分好的作用域进行修改。

这里面需要注意的是，“改变”仅仅是描述“创建”这个动作 —— 创建出来的这个新的作用域。因此它的作用域查询机制仍然是遵循词法作用域模型的。

tips：不要用 with 和 eval 写代码

大家学到这里，要保持头脑清醒：我们这里提到 with 和 eval，仅仅是为了拓宽大家的知识面，确保大家在面试时能够言之有物、不会被问及盲区，而**绝不是为了建议大家使用 with 和 eval**。

事实上， with 和 eval 因为其恼人的副作用（比如对语言性能的拖累、比如我们上面“横空出世”的全局变量等等），一直是我们 JS 程序员眼中的过街老鼠。实际编码中早就没人用了，我也极力推荐大家不要用。

在面试过程中，若面试官试图追问类似于“请讲讲你在实际项目中对 with、 eval 的应用”之类的问题，一率回答“我不用 with 和 eval 写代码” 就可以了。不用担心追问，正常的面试官不会追问（不正常的面试官咱理他干啥？：））。

```
}
```

02 闭包—从编译原理的角度理解作用域

04 闭包面试真题集中解析