

06 | 作用域链：V8是如何查找变量的？

2020-03-28 李兵

《图解 Google V8》

课程介绍 >



讲述：李兵

时长 12:06 大小 8.32M



你好，我是李兵。

在前面我们介绍了 JavaScript 的继承是基于原型链的，原型链将一个个原型对象串起来，从而实现对象属性的查找，今天我们要聊一个和原型链类似的话题，那就是作用域链。

作用域链就是将一个个作用域串起来，实现变量查找的路径。讨论作用域链，实际就是在讨论按照什么路径查找变量的问题。

我们知道，作用域就是存放变量和函数的地方，全局环境有全局作用域，全局作用域中存放了全局变量和全局函数。每个函数也有自己的作用域，函数作用域中存放了函数中定义的变量。

当在函数内部使用一个变量的时候，V8 便会去作用域中去查找。我们通过一段在函数内部查找变量的代码来具体看一下：



```
1 var name = '极客时间'
2 var type = 'global'
3
4
5 function foo(){
6     var name = 'foo'
7     console.log(name)
8     console.log(type)
9 }
10
11
12 function bar(){
13     var name = 'bar'
14     var type = 'function'
15     foo()
16 }
17 bar()
```

在这段代码中，我们在全局环境中声明了变量 `name` 和 `type`，同时还定义了 `bar` 函数和 `foo` 函数，在 `bar` 函数中又再次定义了变量 `name` 和 `type`，在 `foo` 函数中再次定义了变量 `name`。

函数的调用关系是：在全局环境中调用 `bar` 函数，在 `bar` 函数中调用 `foo` 函数，在 `foo` 函数中打印出来变量 `name` 和 `type` 的值。

当执行到 `foo` 函数时，首先需要打印出变量 `name` 的值，而我们在三个地方都定义了变量 `name`，那么究竟应该使用哪个变量呢？

在 `foo` 函数中使用了变量 `name`，那么 V8 就应该先使用 `foo` 函数内部定义的变量 `name`，最终的结果确实如此，也符合我们的直觉。

接下来，`foo` 函数继续打印变量 `type`，但是在 `foo` 函数内部并没有定义变量 `type`，而是在全局环境中调用 `foo` 函数的 `bar` 函数中分别定义了变量 `type`，那么这时候的问题来了，你觉得 `foo` 函数中打印出来的变量 `type` 是 `bar` 函数中的，还是全局环境中的呢？


什么是函数作用域和全局作用域？

要解释清楚这个问题，我们需要从作用域的工作原理讲起。



每个函数在执行时都需要查找自己的作用域，我们称为函数作用域，在执行阶段，在执行一个函数时，当该函数需要使用某个变量或者调用了某个函数时，便会优先在该函数作用域中查找相关内容。

我们再来看一段代码：

 复制代码

```
1 var x = 4
2 var test
3 function test_scope() {
4     var name = 'foo'
5     console.log(name)
6     console.log(type)
7     console.log(test)
8     var type = 'function'
9     test = 1
10    console.log(x)
11 }
12 test_scope()
```

在上面的代码中，我们定义了一个 `test_scope` 函数，那么在 V8 执行 `test_scope` 函数的时候，在编译阶段会为 `test_scope` 函数创建一个作用域，在 `test_scope` 函数中定义的变量和声明的函数都会丢到该作用域中，因为我们在 `test_scope` 函数中定了三个变量，那么常见的作用域就包含有这三个变量。

你可以通过 Chrome 的控制台来直观感受下 `test_scope` 函数的作用域，先打开包含这段代码的页面，然后打开开发者工具，接着在 `test_scope` 函数中的第二段代码加上断点，然后刷新该页面。当执行到该断点时，V8 会暂停整个执行流程，这时候我们就可以通过右边的区域面板来查看当前函数的执行状态。



```
1 <!DOCTYPE html>
2 <html lang="en">
3
4 <head>
5   <title>Document</title>
6 </head>
7
8 <body>
9   <script>
10     var x = 4
11     var test
12     function test_scope() {
13       var name = 'foo'   name = "foo"
14       console.log(name)
15       console.log(type)  type = "function"
16       console.log(test)
17       var type = 'function' type = "function"
18       test = 1
19     console.log(x)
20   }
21   test_scope()
22 </script>
23 </body>
24
25 </html>
```

Paused on breakpoint

Watch

Call Stack

test_scope
(anonymous)

Scope

Local
name: "foo"
type: "function"
this: Window

Global

Breakpoints
test.html:19
console.log(x)

XHR/fetch Breakpoints

DOM Breakpoints

观察作用域

你可以参考图中右侧的 Scope 项，然后点击展开该项，这个 Local 就是当前函数 test_scope 的作用域。在 test_scope 函数中定义的变量都包含到了 Local 中，如变量 name、type，另外系统还为我们添加了另外一个隐藏变量 this，V8 还会默认将隐藏变量 this 存放到作用域中。

另外你还需要注意下，第一个 test1，我并没有采用 var 等关键字来声明，所以 test1 并不会出现在 test_scope 函数的作用域中，而是属于 this 所指向的对象。（this 的工作机制不是本文讨论的重点，不展开介绍。如果你感兴趣，可以在《浏览器工作原理与实践》专栏中《[11 | this：从 JavaScript 执行上下文的视角讲清楚 this](#)》这一讲查看。）

那么另一个问题来了，我在 test_scope 函数使用了变量 x，但是在 test_scope 函数的作用域中，并没有定义变量 x，那么 V8 应该如何获取变量 x？

如果在当前函数作用域中没有查找到变量，那么 V8 会去全局作用域中去查找，这个查找的线路就称为作用域链。

全局作用域和函数作用域类似，也是存放变量和函数的地方，但是它们还是有点不一样：全局作用域是在 V8 启动过程中就创建了，且一直保存在内存中不会被销毁的，直至 V8 退出。而函数作用域是在执行该函数时创建的，当函数执行结束之后，函数作用域就随之被销毁掉了。



全局作用域中包含了很多全局变量，比如全局的 `this` 值，如果是浏览器，全局作用域中还有 `window`、`document`、`opener` 等非常多的方法和对象，如果是 `node` 环境，那么会有 `Global`、`File` 等内容。

V8 启动之后就进入正常的消息循环状态，这时候就可以执行代码了，比如执行到上面那段脚本时，V8 会先解析顶层 (Top Level) 代码，我们可以看到，在顶层代码中定义了变量 `x`，这时候 V8 就会将变量 `x` 添加到全局作用域中。

作用域链是怎么工作的？

理解了作用域和作用域链，我们再回过头来看文章开头的那道思考题：“`foo` 函数中打印出来的变量 `type` 是 `bar` 函数中的呢，还是全局环境中的呢？”我把这段代码复制到下面：

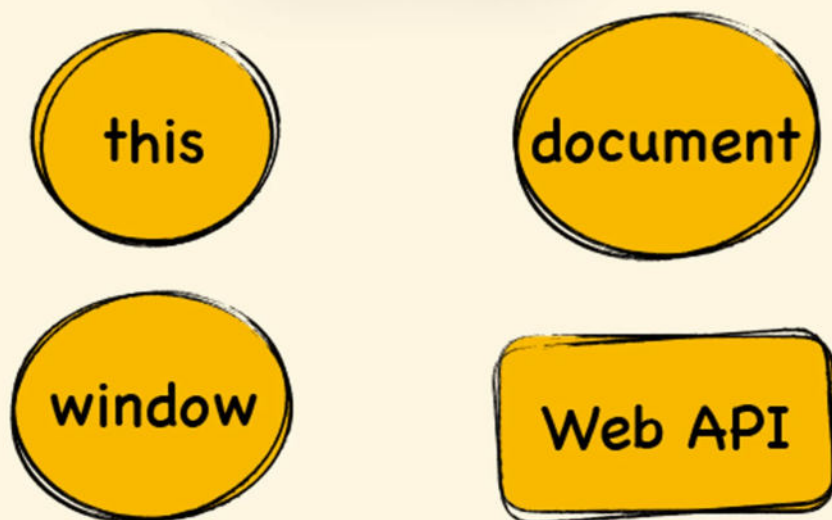
 复制代码

```
1 var name = '极客时间'
2 var type = 'global'
3
4
5 function foo(){
6     var name = 'foo'
7     console.log(name)
8     console.log(type)
9 }
10
11
12 function bar(){
13     var name = 'bar'
14     var type = 'function'
15     foo()
16 }
17 bar()
```

现在，我们结合 V8 执行这段代码的流程来具体分析下。首先当 V8 启动时，会创建全局作用域，全局作用域中包括了 `this`、`window` 等变量，还有一些全局的 Web API 接口，创建的作用域如下图所示：



全局作用域

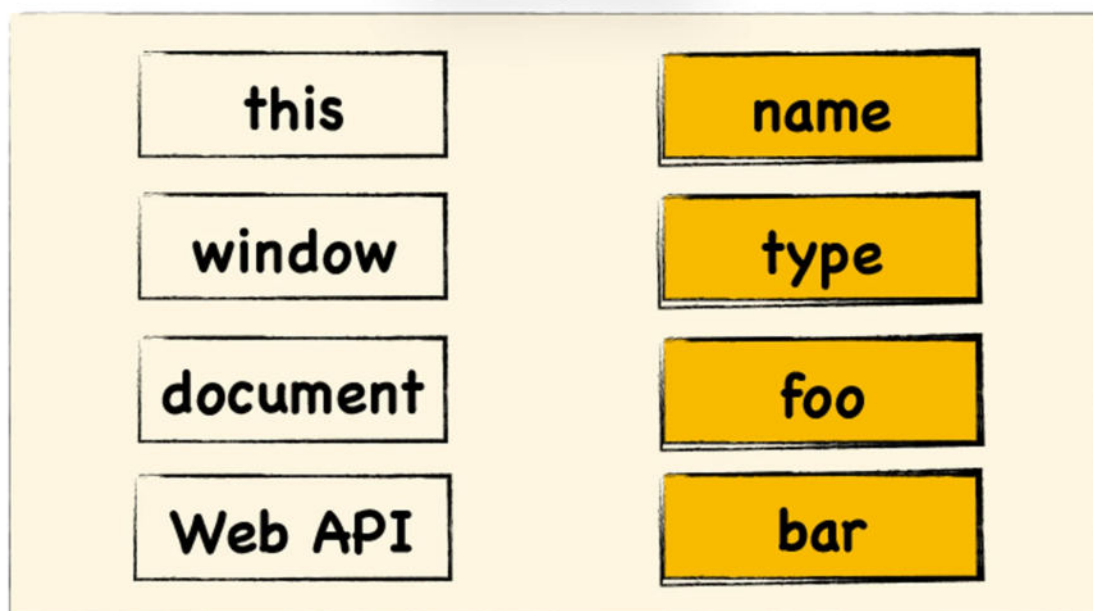


全局作用域

V8 启动之后，消息循环系统便开始工作了，这时候，我输入了这段代码，让其执行。

V8 会先编译顶层代码，在编译过程中会将顶层定义的变量和声明的函数都添加到全局作用域中，最终的全局作用域如下图所示：


全局作用域



全局作用域



全局作用域创建完成之后，V8 便进入了执行状态。前面我们介绍了变量提升，因为变量提升的原因，你可以把上面这段代码分解为如下两个部分：

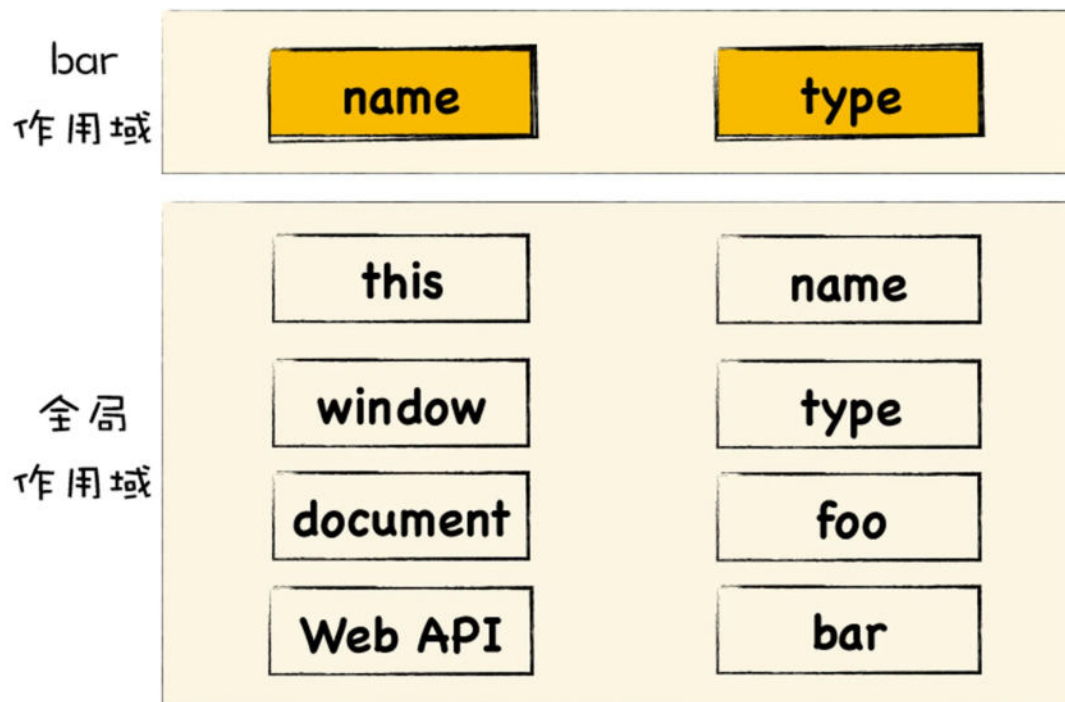
 复制代码

```
1 //=====解析阶段--实现变量提升=====
2 var name = undefined
3 var type = undefined
4 function foo(){
5     var name = 'foo'
6     console.log(name)
7     console.log(type)
8 }
9 function bar(){
10     var name = 'bar'
11     var type = 'function'
12     foo()
13 }
14
15
16
17
18 //====执行阶段=====
19 name = '极客时间'
20 type = 'global'
21 bar()
```

第一部分是在编译过程中完成的，此时全局作用域中两个变量的值依然是 undefined，然后进入执行阶段；第二部代码就是执行时的顺序，首先全局作用域中的两个变量赋值“极客时间”和“global”，然后就开始执行函数 bar 的调用了。

当 V8 执行 bar 函数的时候，同样需要经历两个阶段：编译和执行。在编译阶段，V8 会为 bar 函数创建函数作用域，最终效果如下所示：

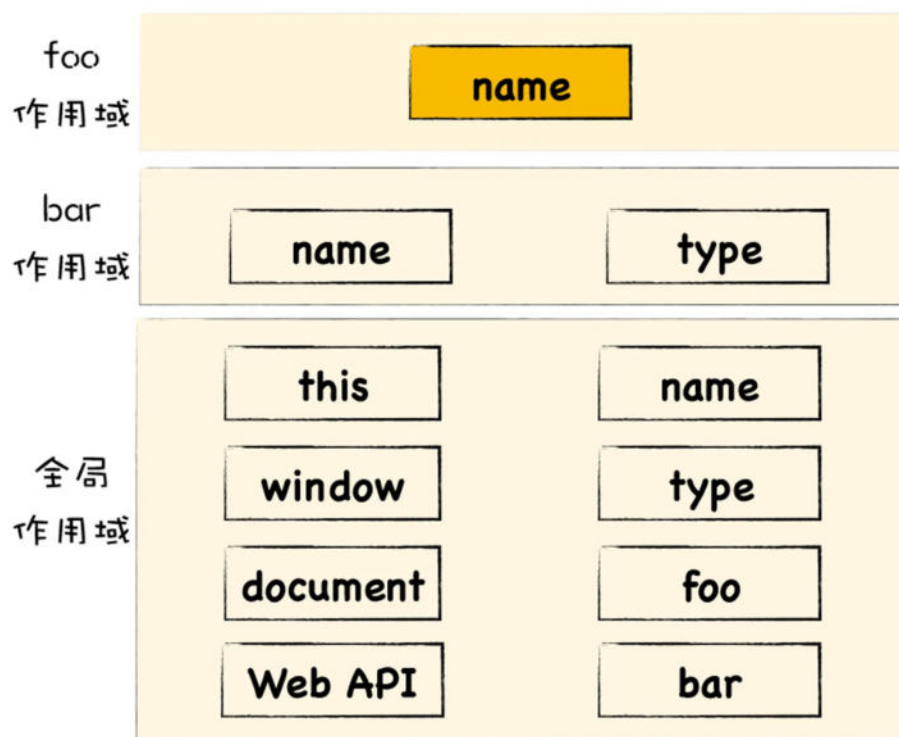




bar函数作用域

然后进入了 `bar` 函数的执行阶段。在 `bar` 函数中，只是简单地调用 `foo` 函数，因此 V8 又开始执行 `foo` 函数了。

同样，在编译 `foo` 函数的过程中，会创建 `foo` 函数的作用域，最终创建效果如下图所示：



foo函数作用域



好了，这时候我们就有了三个作用域了，分别是全局作用域、bar 的函数作用域、foo 的函数作用域。

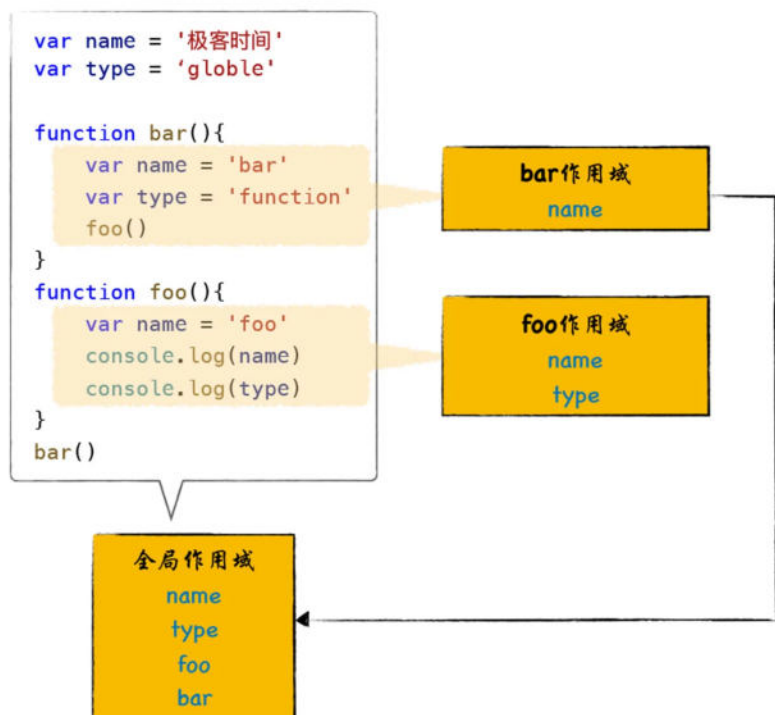
现在我们可以将刚才提到的问题转换为作用域链的问题了：foo 函数查找变量的路径到底是什么？

- 沿着 foo 函数作用域→bar 函数作用域→ 全局作用域；
- 还是，沿着 foo 函数作用域→ 全局作用域？

因为 JavaScript 是基于词法作用域的，词法作用域就是指，查找作用域的顺序是按照函数定义时的位置来决定的。bar 和 foo 函数的外部代码都是全局代码，所以无论你是在 bar 函数中查找变量，还是在 foo 函数中查找变量，其查找顺序都是按照当前函数作用域→ 全局作用域这个路径来的。

由于我们代码中的 foo 函数和 bar 函数都是在全局下面定义的，所以在 foo 函数中使用了 type，最终打印出来的值就是全局作用域中的 type。

你可以参考下面这张图：



另外，我再展开说一些。因为词法作用域是根据函数在代码中的位置来确定的，作用域是在声明函数时就确定好的了，所以我们将词法作用域称为静态作用域。

和静态作用域相对的是动态作用域，动态作用域并不关心函数和作用域是如何声明以及在何处声明的，只关心它们从**何处调用**。换句话说，作用域链是基于调用栈的，而不是基于函数定义的位置的。（动态作用域不是本文讨论的重点，如果你感兴趣，可以参考《浏览器工作原理与实践》专栏中的《[🔗10 | 作用域链和闭包：代码中出现相同的变量，JavaScript 引擎是如何选择的？](#)》这一节。）

总结

今天，我们主要解释了一个问题，那就是在一个函数中，如果使用了一个变量，或者调用了另外一个函数，V8 将会怎么去查找该变量或者函数。

为了解释清楚这个问题，我们引入了作用域的概念。作用域就是用来存放变量和函数的地方，全局作用域中存放了全局环境中声明的变量和函数，函数作用域中存放了函数中声明的变量和函数。当在某个函数中使用某个变量时，V8 就会去这些作用域中查找相关变量。沿着这些作用域查找的路径，我们就称为作用域链。

要了解查找路径，我们需要明白词法作用域，词法作用域是按照代码定义时的位置决定的，而 JavaScript 所采用的作用域机制就是词法作用域，所以作用域链的路径就是按照词法作用域来实现的。

思考题

我将文章开头那段代码稍微调整了下，foo 函数并不是在全局环境中声明的，而是在 bar 函数中声明的，改造后的代码如下所示：

 复制代码

```
1 var name = '极客时间'
2 var type = 'global'
3 function bar() {
4     var type = 'function'
5     function foo() {
6         console.log(type)
7     }
8     foo()
9 }
10 bar()
```



那么执行这段代码之后，打印出来的内容是什么？欢迎你在留言区与我分享讨论。

感谢你的阅读，如果你觉得这一讲的内容对你有所启发，也欢迎把它分享给你的朋友。

分享给需要的人，Ta订阅超级会员，你将得 50 元

Ta单独购买本课程，你将得 20 元

生成海报并分享

赞 6 提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 05 | 原型链：V8是如何实现对象继承的？

下一篇 07 | 类型转换：V8是怎么实现1+“2”的？

学习推荐

JVM + NIO + Spring

各大厂面试题及知识点详解

限时免费





Geek_f

2020-05-19

老师，下面这题困扰很久了，不知道作用域链该怎么画，想请教下：

```
var a = [];  
for(let i = 0;i<10;i++){  
  a[i]=function(){  
    console.log(i)  
  }  
};  
a[2]();
```

作者回复: let定义的i会运行for的块级作用域中，每次执行一次循环，都会创建一个块级作用域。

在这个块级作用域中，你又定义了一个函数，而这个函数又引用了函数外部的i变量，那么这就产生了闭包，也就是说，所有块级作用域中的i都不会被销毁，你在这里执行了10次循环，那么也就创建了10个块级作用域，这十个块级作用域中的变量i都会被保存在内存中。

那么当你再次调用该a[n]()时，v8就会拿出闭包中的变量i，并将其打印出来，因为每个闭包中的i值都不同，所以a[n]()时，打印出来的值就是n，这个就非常符合直觉了。

但是如果你将for循环中的i变量声明改成var，那么并不会产生块级作用域，那么函数引用的i就是全局作用域中的了，由于全局作用域中只有一个，那么在执行for循环的时候，i的值会一直被改变，最后是10，所以最终你执行a[n]()时，无论n是多少，打印出来的都是10. 那么这就是bug之源了。

共 2 条评论 >

👍 38



杨越

2020-03-28

老师，昨天面试小红书，有个问题请教下：

function f(){setTimeout(fn,0)}面试官问我这种调用会不会导致内存溢出？

作者回复: fn是啥？是函数f吧？

如果fn是f的话，那么不会溢出啊，因为这是异步调用，下次执行f函数时，已经在新的栈中执行了，所以当前栈不会发生溢出！

理解这个问题核心是理解事件循环和消息队列这套机制，这个专栏会有几篇文章介绍事件循环系统的，另外我的上个《浏览器专栏》对这块介绍的比较详细！



共 6 条评论 >

👍 27



刘大夫

2020-04-07

和 this 对比就很好记了，可以简单的理解为 this 是看函数的调用位置，作用域是看函数的声明位置。除了箭头函数等那些特殊的情况

作者回复: 对，可以认为this是用来弥补JavaScript没有动态作用域特性的🤔

共 3 条评论 >

👍 26



Bazinga

2020-03-29

老师，在大量数据时(百万级别)，foreach循环比for循环的执行效率低，是因为什么

作者回复: 因为foreach有函数回调过程啊，每次回调都要额外创建新的额外的栈页，新的上下文，那么效率也就随之下来了

共 3 条评论 >

👍 13



Jack.Huang

2020-06-29

根据ECMAScript最新规范，函数对象有一个[[Environment]]内部属性，保存的是函数创建时当前正在执行的上下文环境，当函数被调用并创建执行上下文时会以[[Environment]]的值初始化作用域链，所以从规范也可以得知函数的作用域只跟函数创建时的当前上下文环境有关。规范中关于[[Environment]]的描述：<https://tc39.es/ecma262/#sec-ecmascript-function-objects>

作者回复: 赞



👍 11



WinfredH

2020-04-02

请教老师一个困惑了很久的问题：分不清楚作用域和执行上下文中的环境记录。函数的作用域跟函数执行上下文中的环境记录有什么区别呢？两者都是用于在编译和执行js代码阶段保存函数中声明的变量吗？变量查找到到底是往环境记录中找还是作用域中找？

共 3 条评论 >

👍 5



bright



2020-03-29

词法作用域查找作用域的顺序是按照函数定义时的位置来决定的,foo函数的外层是bar函数,所以打印出来的是'function',有意思的是如果打印的是this.type的话就是'global'了,经常被这个东西搞的头疼。

共 1 条评论 >

👍 5



潇潇雨歇

2020-03-28

思考题: 打印的是function, 根据定义时的位置查找作用域链, foo函数查找到的是bar, 而bar函数作用域内是有type变量的。



👍 3



Z_CHP

2020-03-28

"另外你还需要注意下, 第一个 test1, 我并没有采用 var 等关键字来声明, 所以 test1 并不会出现在 test_scope 函数的作用域中, 而是属于 this 所指向的对象。"这句话有点疑惑, 文中代码的部分没有看到有test1这个变量名, 这个test1说的是哪里的呢?

共 6 条评论 >

👍 3



零维

2020-05-19

老师, 请问一下我下面关于作用域的理解是否正确:

如果我运行一个 js 文件, 在解释阶段生成 AST 树之后, 紧接着, 这个 js 文件的所有的作用域(函数作用域, 块级作用域, 全局作用域)就都已经确定了, 就算有某些函数没有被执行, 它的作用域内含有哪些变量也已经确定了, 但是这些变量还都不会真实存在栈或堆中。也就是说, 某个未执行函数的执行上下文中的变量环境和词法环境现在也已经确定了。

这样的理解对吗?

作者回复: 正常情况下是这样的, 单是执行eval的情况, 这个eval方法很有破坏性, 因为在执行eval之前, 引擎并不知道eval要执行的内容, 也就没有办法提前做预解析



👍 3



chengl

2020-04-09

老师你好, 为了更好的理解效果, 建议有打印信息的代码, 在代码底部增加打印结果, 便于结合理解, 有些代码虽然很简单, 但是就怕理解错了。



作者回复: 好, 回头我修订的时候我来补充



👍 1



非洲大地我最凶

2020-03-28

作用域链是基于调用栈的, 而不是基于函数定义的位置的? 这里说的是动态作用域链吗。静态作用域不是基于函数定义的位置吗

共 2 条评论 >

👍 2



离人生巅峰还差一只猫...

2020-06-30

老师你好, 有个问题想请教下:

之前提到在编译阶段就会生成作用域和AST, 在本节中又提到函数在执行时才会创建作用域。那么编译时创建的作用域具体是哪些作用域, 因为通过d8 print-scopes发现都所有作用域都存在

作者回复: 是打印所有的作用域, 编译的时候就编译什么代码就创建什么代码的作用域。

比如执行全局代码的时候, 只会生成全局作用域, 函数的作用域就不会被生成, 当执行某个函数时, 就会生成函数的作用域了。



👍 2



于你

2020-04-09

之前看《浏览器运行原理》时候, 说的是作用域链是在定义代码的时候决定的, 当时的有点迷糊, 今天突然看明白了, 看来还是需要反复巩固知识啊, 给老师点个赞!

作者回复: 👍



👍 1



zlxag

2020-04-03

老师你这个图片怎么画的呀

作者回复: keynote

共 2 条评论 >

👍 1





一步

2020-03-28

在解析阶段会进行语法分析，词法分析，这时候词法分析会生成作用域，也会基于代码定义位置生成对应的作用域链（所有称为词法作用域）



1



Geek_5311d4

2021-05-22

请问可不可以这样理解呢。作用域是一个抽象的编译时概念，规范中并没有其定义。执行上下文是个运行时概念，是ES标准所规定的。作用域链本质上是通过执行上下文中的词法环境和外部指针实现的。



Geek_81a93b

2021-03-11

当函数执行结束之后，函数作用域就随之被销毁掉了。
如果这个函数被多次调用执行，会多次创建作用域并销毁吗？



Kevin

2020-07-02

输出 "function";

执行foo()首先在foo函数的作用域中找type，没有找到就会沿着作用域链查找type，词法作用域是根据定义函数的地方决定的，所以现在bar函数作用域中查找，找到了type='function'返回了；



刹那

2020-06-09

老师可以讲下作用域跟作用域链在堆栈中的存储方式吗？按照我的理解，作用域内的变量应该是根据类型存放，基本数据类型当栈里面，对象放堆里面。他们之间是怎么链起来的？

