

02 | `var x = y = 100`: 声明语句与语法改变了JavaScript语言核心性质

2019-11-13 周爱民

JavaScript核心原理解析

[进入课程 >](#)



讲述：周爱民

时长 21:12 大小 19.43M



你好，我是周爱民。

如果你听过上一讲的内容，心里应该会有一个问题，那就是——在规范中存在的“引用”到底有什么用？它对我们的编程有什么实际的影响呢？


当然，除了已经提及过的`delete 0`和`obj.x`之外，在今后的课程中，我还会与你讨论这个“引用”的其它应用场景。而今天的内容，就从标题来看，若是我要说与这个“引用”有关，你说不定得跳起来说我无知；但若说是跟“引用”无关的话呢，我觉得又不能把标题中的这一行代码解释清楚。

为什么这行代码看起来与规范类型中的“引用”无关呢？因为这行代码出现的时候，连 ECMAScript 这个规范都不存在。

我大概是在 JavaScript 1.2 左右的时代就接触到这门语言，在我写的最早的一些代码中就使用过它，并且——没错，你一定知道的：它能执行！

有很多的原因会促使你在 JavaScript 写出表达式连等这样的代码。从 C/C++ 走过来的程序员对这样的一行代码是不会陌生的。它能用，而且结果也与你的预期会一致，例如：

```
1 var x = y = 100;
2 console.log(x); // 100
3 console.log(y); // 100
```

 复制代码

它既没错、又好用，还很酷，你说我们为什么不用它呢？然而很不幸，这行代码可能是 JavaScript 中最复杂和最容易错用的表达式了。

所以今天我要与你一起好好地盘盘它。

声明

至今为止，除标签声明之外，JavaScript 中一共只有六条声明用的语句。注意，所有真正被定义“声明”的语法结构都一定是“语句”，并且都用于声明一个或多个标识符。这里的标识符包括变量、常量等。

严格意义上讲，JavaScript 只有变量和常量两种标识符，六条声明语句中：

letx...

声明变量 x。不可在赋值之前读。

constx...

声明常量 x。不可写。

varx...

声明变量 x。在赋值之前可读取到 undefined 值。

functionx...

声明变量 x。该变量指向一个函数。

classx...

声明变量 x。该变量指向一个类（该类的作用域内部是处理严格模式的）。

import ...

导入标识符并作为常量（可以有多种声明标识符的模式和方法）。

除了这六个语句之外，还有两个语句有潜在的声明标识符的能力，不过它们并不是严格意义上的声明语句（声明只是它们的语法效果）。这两个语句是指：

for (var|let|constx...) ...

for 语句有多种语法来声明一个或多个标识符，用作循环变量。

try ... catch (x) ...

catch 子句可以声明一个或多个标识符，用作异常对象变量。

总的来说，除上述的语法，用户是没有其它方式来在当前的代码上下文中“声明”出一个标识符来的。而我之所以在这里严格强调这一“汇总性的”结果，是因为下面的一个简单论断，所有的“声明”：

都意味着 JavaScript 将可以通过“静态”语法分析发现那些声明的标识符；

标识符对应的变量 / 常量“一定”会在用户代码执行前就已经被创建在作用域中。

这个标题中的`var x`就是一个声明。在这个声明的后半部分，使用“=”这个符号引导了一个初始化语法——通常情况下可以将它理解为一个赋值运算。

从读取值到赋值

声明是在语法分析阶段就处理的，并且因此它会使得当前代码上下文在正式执行之前就拥有了被声明的标识符，例如`x`。

这其实非常有趣，因为这表明JavaScript 虽然被称为是“动态语言”，但确实是拥有静态语义的。而在 JavaScript 的早期，这个静态语义其实并没有处理得太好，一个典型的问题就是所谓的“变量提升”。也就是，可以在变量声明之前访问该变量。例如：

 复制代码

```
1 console.log(x); // undefined
2 var x = 100;
3 console.log(x); // 100
```

这个“变量提升”还包括“变量被创建于声明它的语法块”之外的意思，但这并不是这里要讨论的内容，我会在今后的课程里，你只需要留意这个变量的读写过程就好了。那么，关于读取值，之前声明的变量与常量又有什么不同呢？

如上面已经说过的，由于标识符是在用户代码执行之前就已经由静态分析得到，并且创建在环境中，因此 `let` 声明的变量和 `var` 声明的变量在这一点上没有不同：它们都是在读取一个“已经存在的”标识符名。例如：

 复制代码

```
1 var y = "outer";
2 function f() {
3   console.log(y); // undefined
4   console.log(x); // throw a Exception
5   let x = 100;
6   var y = 100;
7   ...
8 }
```

正是由于`var y`所声明的那个标识符在函数 `f()` 创建（它自己的闭包）时就已经存在，所以才阻止了`console.log(y)`访问全局环境中的`y`。类似的，`let x`所声明的那个`x`其实也已经存在 `f()` 函数的上下文环境中。访问它之所以会抛出异常（Exception），不是因为它不存在，而是因为这个标识符被拒绝访问了。

在 ECMAScript 6 之后出现的`let/const`变量在“声明（和创建）一个标识符”这件事上，与`var`并没有什么不同，只是 JavaScript 拒绝访问还没有绑定值的`let/const`标识符而已。

回到 ECMAScript 6 之前：JavaScript 是允许访问还没有绑定值的`var`所声明的标识符的。这种标识符后来统一约定称为“变量声明（`varDecls`）”，而“`let/const`”则称为“词法声明（`lexicalDecls`）”。JavaScript 环境在创建一个“变量名（`varName` in `varDecls`）”后，会为它初始化绑定一个 `undefined` 值，而“词法名字（`lexicalNames`）”在创建之后就没有这项待遇，所以它们在缺省情况下就是“还没有绑定值”的标识符。

NOTE：6 种声明语句中的函数是按 `varDecls` 的规则声明的；类的内部是处于严格模式中，但它的名字仍然是按 `varDecls` 来处理的；`import` 导入的名字是按 `const` 的规则来处理的。所以所有的声明本质上只有三种处理模式：`var` 变量声明，`let` 变量声明，和 `const` 常量声明。

所以，标题中的`var x = ...`在语义上就是为变量 `x` 绑定一个初值。在具体的语言环境中，它将被实现为一个赋值操作。

赋值

如果是在一门其它的（例如编译型的）语言中，“为变量 `x` 绑定一个初值”就可能实现为“在创建环境时将变量 `x` 指向一个特定的初始值”。这通常是静态语言的处理方法，然而，如前面说过的，JavaScript 是门动态的语言，所以它的“绑定初值”的行为是通过动态的执行过程来实现的，也就是赋值操作。

那么请你仔细想想，一个赋值操作在语法上怎么表达呢？例如：

```
1 变量名 = 值
```

这样对吗？不对！在 JavaScript 中，这样讲是非常不正确的。正确的说法是：

```
1 lRef = rValue
```

也就是将右操作数（的值）赋给左操作数（的引用）。它的严格语法表达是：

```
LeftHandSideExpression < = | AssignmentOperator > AssignmentExpression
```

也就是说，在 JavaScript 中，一个赋值表达式的左边和右边其实“都是”表达式！

向一个不存在的变量赋值

接下来我要给你介绍的是从 JavaScript 1.0 开始就遗留下来的一个巨坑，也就是所谓的“**变量泄漏**”问题。这在早期的 JavaScript 中的确是一个好用的特性：如果你向一个不存在的变量名赋值，那么 JavaScript 会在全局范围内创建它。

也就是说，代码中不需要显式地声明一个变量了，变量可以随用随声明，也不用像后来的 `let` 语句一样，还要考虑在声明语句之前能不能访问的问题了。这非常简单，在少量的代码中也相当易用。

但是，如果代码规模扩大，变成百千万行代码，那么“**一个全局变量是在哪里声明和创建的**”就变成一个非常要紧的问题。

如果随时都可能泄露一个代码给全局，或者随时都可能因为忘记本地的声明而读写了全局变量，那对调试除错将是一场灾难。另外，晚一些出现的运行期优化技术也不能很好地处理这种情况。所以从 ECMAScript5 开始的严格模式就禁止了这种特性，试图避免用户将变量泄露到全局环境。

然而现实中，即使在严格模式下这种泄露也未能避免。这称为“**间接执行**”，这将是另一个巨大的议题，并且是 ECMAScript6 之后开始的一种新的机制。但是现在这里发生的事情，

也就是这个“向不存在的变量赋值”的问题，是从 JavaScript 1.0 时代就遗留下来的问题，也是 ECMAScript 为 JavaScript 语言所填补的最大的设计漏洞之一。

那么，在具体技术细节上，这个变量声明是如何发生的呢？

事实上，这是因为在早期设计中，JavaScript 的全局环境是引擎使用一个称为“**全局对象**”东西管理起来的。

这个全局对象几乎类似或完全等同于一个普通对象。只不过，JavaScript 引擎将全局的一些缺省对象、运行期环境的原生对象等东西都初始化在这个全局对象的属性中，并使用这个对象创建了一个称为“**全局对象闭包**”的东西，从而得到了 JavaScript 的全局环境。

早期的 JavaScript 的引擎实现非常简洁，许多基础的技术组件都是直接复用的，例如这里的所谓全局环境、全局闭包，或者全局对象的实现方法，就与“**with 语句**”的效果完全相同——他们是相互复用的。

当向一个不存在的变量赋值的时候，由于全局对象的属性表是可以动态添加的，因此 JavaScript 将变量名作为属性名添加给全局对象。而访问所谓全局变量时，就是访问这个全局对象的属性。因此，实际效果就变成了“可以动态地向全局环境中添加一个变量”。并且，显然地，我们在第一讲已经讲过这个结果——你可以删除掉这个动态添加的“变量”，因为本质上就是在删除全局对象的属性。

那么现在（我是指 ECMAScript6 之后）的 JavaScript 的全局环境有什么不同吗？

为了兼容旧的 JavaScript 语言设计，现在的 JavaScript 环境仍然是通过将全局对象初始化为这样的一个全局闭包来实现的。但是为了得到一个“尽可能”与其它变量环境相似的声明效果（varDecls），ECMAScript 规定在这个全局对象之外再维护一个变量名列表

（varNames），所有在静态语法分析期或在 eval() 中使用 var 声明的变量名就被放在这个列表中。然后约定，这个变量名列表中的变量是“直接声明的变量”，不能使用 delete 删除。

于是，我们得到了这样的一种效果：

```
1 > var a = 100;
2 > x = 200;
3
4 # `a`和`x`都是 global 的属性
5 > Object.getOwnPropertyDescriptor(global, 'a');
6 { value: 100, writable: true, enumerable: true, configurable: false }
7 > Object.getOwnPropertyDescriptor(global, 'x');
8 { value: 200, writable: true, enumerable: true, configurable: true }
9
10 # `a`不能删除, `x`可以被删除
11 > delete a
12 false
13 > delete x
14 true
15
16 # 检查
17 > a
18 100
19 > x
20 ReferenceError: x is not defin
```

所以, 表面看起来“泄漏到全局的变量”与使用`var`声明的都是全局变量, 并且都实现为 `global` 的属性, 但事实上它们是不同的。并且当`var`声明发生在 `eval()` 中的时候, 这一特性又还有所不同, 例如:

```
1 # 使用 eval 声明
2 > eval('var b = 300');
3
4 # 它的性质是可删除的
5 > Object.getOwnPropertyDescriptor(global, 'b').configurable;
6 true
7
8 # 检测与删除
9 > b
10 300
11 > delete b
12 true
13 > b
14 ReferenceError: b is not define
```

这种情况下使用`var`声明的变量名尽管也会添加到 `varNames` 列表, 但它也可以从 `varNames` 中移除 (这是唯一一种能从 `varNames` 中移除项的特例, 而 `lexicalNames` 中

的项是不可移除的)。

发生了什么？

所以，现在回到今天讨论的这行代码`var x = y = 100`，在这行代码中，等号的右边是一个表达式`y = 100`，它发生了一次“向不存在的变量赋值”，所以它隐式地声明了一个全局变量`y`，并赋值为 100。

而一个赋值表达式操作的本身也是有“结果 (Result)”的，它是右操作数的值。注意，这里是“值”而非“引用”，例如下面的测试中的`a`将是一个函数，而不是带着“this 对象”信息的方法：

 复制代码

```
1 // 调用 obj.f() 时将检测 this 是不是原始的 obj
2 > obj = { f: function() { return this === obj } };
3
4 // false, 表明赋值表达式的“结果 (result)”只是右侧操作数的值，即函数 f
5 > (a = obj.f)();
6 false
```

到现在为止，我们讲述了整个语句的过程，也就是说，由于“`y = 100`”的结果值是 100，所以该值将作为初始值赋值“变量 `x`”。并且，从语义上来说，这是变量“`x`”的初始绑定。

之所以强调这一点，是因为相同的分析过程也可以用在 `const` 声明上，而 `const` 声明是只有一次绑定的，常量的初始绑定也是通过“执行赋值过程”来实现的。

知识回顾

`var` 等声明语句总是在变量作用域（变量表）或词法作用域中静态地声明一个或多个标识符。

全局变量的管理方式决定了“向一个不存在的变量赋值”所导致的变量泄露是不可避免的。

动态添加的“`var` 声明”是可以删除的，这是唯一能操作 `varNames` 列表的方式（不过它并不存在多少实用意义）。

变量声明在引擎的处理上被分成两个部分：一部分是静态的、基于标识符的词法分析和
管理，它总是在相应上下文的环境构建时作为名字创建的；另一部分是表达式执行过
程，是对上述名字的赋值，这个过程也称为绑定。

这一讲标题里的这行代码中，`x` 和 `y` 是两个不同的东西，前者是声明的名字，后者是一个
赋值过程可能创建的变量名。

思考题

根据今天讲解的内容，我希望你可以尝试回答以下问题：

严格来说，声明不是语句。但是，是哪些特性决定了声明不是“严格意义上的”语句
呢？

在下一讲中我会来讲一讲 JavaScript 社区中的一个历史悬案，这桩悬案与今天讨论的这行
代码的唯一区别在于：它不是声明语句，而是赋值表达式。

小程序学习打卡邀请

打卡 46 天，彻底搞定 JavaScript



扫一扫参与小程序打卡



新版升级：点击「 请朋友读」，20位好友免费读，邀请订阅更有**现金**奖励。

© 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 01 | `delete 0`: JavaScript 中到底有什么是可以销毁的

下一篇 03 | `a.x = a = {n:2}`: 一道被无数人无数次地解释过的经典面试题

精选留言 (21)

写留言



铭

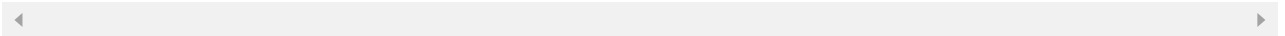
2019-11-13

〈以下是小生愚见〉

概念纷繁，建议老师将讲解重心放到这门语言的现有特性，贯之历史脉络，是否（怎样）解决了某种设计缺陷。这样，知识纵深感更强，并可指导实际工作以避免踩坑。适当穿插示例代码和图文更佳。

展开 ▾

作者回复: 多谢。我在后面的课程中尽量注意这个 ^^.



6



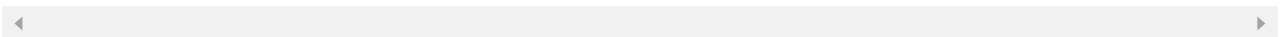
fatme

2019-11-14

声明和语句的区别在于发生的时间点不同，声明发生在编译期，语句发生在运行期。声明发生在编译期，由编译器为所声明的变量在相应的变量表，增加一个名字。语句是要在运行时执行的程序代码。因此，如果声明不带初始化，那么可以完全由编译器完成，不会产生运行时执行的代码。

展开 ▾

作者回复: +1 ^^.



3



Mr_Liu

2019-11-13

思考题: 小白的我，没有太明确的答案，暂时还不能明确自己理解究竟是否正确，希望听老师后续的课程能够明白

读完今天的这篇理解了昨天的提问，为什么`var x = '123' delete x`是false，即使是...

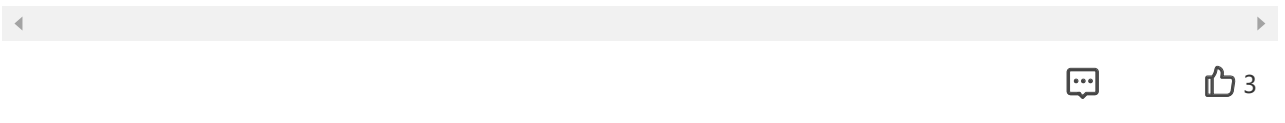
展开 ▾

作者回复: 是的。

`delete`从不删除值。`delete`只能删除引用，例如`obj.x`，或者`with(obj) delete x`，这些都是以引用

的方式得到的，所以delete才能删除它们。

``delete 0``这种行为并不真的能够发生，它什么也没做，只是返回了true而已。

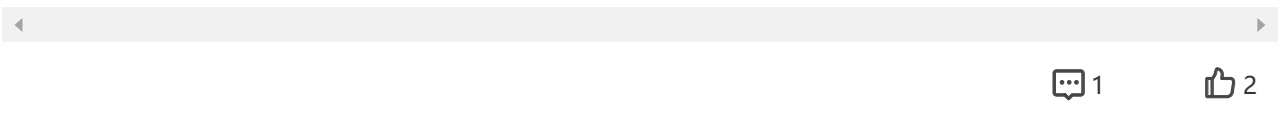


佳民

2019-11-13

思考题：var声明会声明提升，在语法解析（静态分析）阶段进行，不是在运行阶段执行，这样理解对吗？

作者回复: 是的。不过所有的6种声明都是如此，非独var声明。



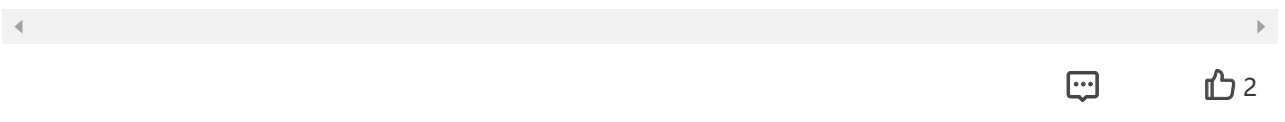
Geek_kkkkkkkk

2019-11-13

老师，您文章中讲到“ECMAScript 规定在这个全局对象之外再维护一个变量名列表（var Names），所有在静态语法分析期或在 eval() 中使用var声明的变量名就被放在这个列表中。然后约定，这个变量名列表中的变量是“直接声明的变量”，不能使用delete删除。”，这里面的意思我理解了 eval() 中使用var声明的变量名，不可用delete删除，但是下面的代码中eval()声明了b,configurable是true,可删除，是否矛盾了？还是我理解的不...
展开

作者回复: > 这种情况下使用var声明的变量名尽管也会添加到varNames列表，但它也可以从var Names中移除（这是唯一一种能从varNames中移除项的特例...

文章中是解释过的。可能我没有足够地强调这个特例，或者它的特殊性吧。



蓝配鸡

2019-11-13

醍醐灌顶，但是有一些疑问：

文中说，

"如果是在一门其它的（例如编译型的）语言中，“为变量 x 绑定一个初值”就可能实现为“在创建环境时将变量 x 指向一个特定的初始值”。这通常是静态语言的处理方法，然而，如前面说过的，JavaScript 是门动态的语言，所以它的“绑定初值”的行为是通过...

展开

作者回复: 动态语言的诸多细节, 其实要到18讲之后才会讨论到。不过有个概念上的问题, 并不是说有动态类型就是动态语言, 或者说支持动态执行就是动态语言。有很多方面的“动态语言的特性”, 这个需要详细解析。

仅是说初始化这一项, 使用动态赋值的原因是因为这个值必须要到执行期环境中才能确定下来, 而在静态语法分析阶段是确认不了的。——所以它不可能“先于环境”而执行。

1

2



Makus

2019-11-13

附「语句和声明」的链接: <https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Statements>

严格意义上来说, 除了本篇中出现的六种和链接中的五大类都不能算作语句吧。

例如: 标题中的 `y = 100`

麻烦老师指正一下, 不太懂

展开

作者回复: 不着急。会慢慢讲到的。^^.

还有, 我所讲的有些会与MDN有出入, 这个你可以先存疑。不必急着得到“YES/NO”这样的答案, 慢慢地听到后面就会知道我为什么这么讲了。

2

2



陆昱嘉

2019-11-17

老师, 一个赋值表达式的左边和右边其实“都是”表达式, 那么`var x=(var y=100);`这样就报错, 原因是什么? `varNames`里面的冲突?

展开

作者回复: “`var y = 100`”不是表达式, 而是语句。

所以“`var x = y = 100`”里面:

- “`var x ...`”是语句语法 (Variable Statement)
- “`= ...`”是初始化器 (Initializer, 严格来说, 也不是表达式, 而是语法的一部分)
- “`y = 100`”是表达式 (expression)。

所以你列的问题，报错的原因是“语句在语法上不支持这种写法”，即在“=...”中的“...”上面，必须是一个用来做“初始器”的表达式，而不能“再是一个语句”。

参见ECMAScript:

<https://tc39.es/ecma262/#sec-variable-statement>

<https://tc39.es/ecma262/#prod-VariableDeclaration>



1



Marvin

2019-11-14

相当于

```
var/let/const x = (y = 100)
```

再拆就是

```
y = 100 // 变量泄漏
```

```
var x = y // 模拟表达式返回值赋值
```

展开 ∨

作者回复: 是的。简洁，正确。^^.



1



ssala

2019-11-13

老师，像变量提升这种不符合直觉的设计，难道规范不能将其移除吗？不移除是为了兼容老代码吗？我想应该没有代码会依赖这个特性吧？

展开 ∨

作者回复: 一是因为向下兼容，这是规范订制过程中面临的主要挑战之一。第二个原因，在于按照JavaScript的核心设计，在原理上这就很难绕过去。例如就包括不受兼容性影响的import，其实也是有变量提升效果的。例如下面这样的代码也是成立的：

```
...
```

```
console.log(x);
```

```
import x from './test.js';
```

```
...
```



1



凯文1985
2019-11-13

对于js在词法分析和语法分析中发生的事情还不够了解 希望也能借这个课程讲解一下：)



1



Smallfly
2019-11-13

老师文中说变量声明分为两步，静态分析和动态绑定，JS 是在进入每个作用域后，执行前进行词法分析的么？

作者回复: 是在加载.js文件之后，执行第一行用户代码之前，就已经完成了全部的词法分析（当然，除了eval执行的）。



1



旺旺
2019-11-13

简单语法语句背后还有这么多门道.

声明不是语句，声明只是在告诉编译器，并没有对运行产生影响，是这样吗？

展开 ∨

作者回复: 是的。



1



blockman-luz
2019-11-15

这一节不知主题是什么?这样讲会造成知识点东一快西一快没有发挥系统学习的效果，更像很多很散的集合

作者回复: 有主题的哦~ ^^.

目录中对这一个段落的定义是“JavaScript语言是如何构建起来的”。基本上是讲静态语言语义，以及它在引擎级别上的实现。5个小节，从值、引用等基础，到模块和语句，基本上覆盖构成语言的核心组件。

可不是在乱讲哦。^^.





Ada

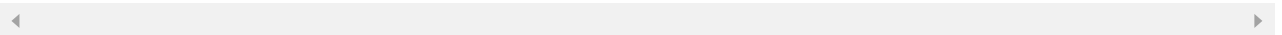
2019-11-15

请教老师一个问题。现在大部分的JS框架和NODEJS环境都是默认STRICT MODE。也就说任何变量都必须有声明，而像`var x=y=100` 这样的语句执行就会报错。如果未来ECMAJS都是默认STRICT MODE方向，那像全局变量污染问题还是否需要担心呢？

展开 ▾

作者回复: strict不能杜绝全局变量污染。例如`new Function()`一样可能操作全局。

就目前的情况来说，strict会对程序员使用这门语言有很好的制约作用，对语言将来的发展也很好。但是，JavaScript的底层设计本来就是那样，多范型带来的特性之间的碰撞不可避免。例如strict其实是静态语言的需求，而`new Function`去是动态语言的特性。



一步

2019-11-14

JS 6种声明语句

1. let 声明
2. const 声明
3. var 声明
4. function x {} 声明...

展开 ▾

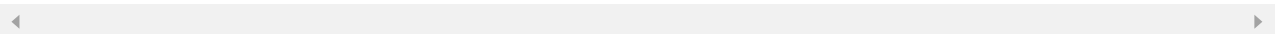


一步

2019-11-14

语句都会返回一个引用，声明不会，可不可以这样理解？

作者回复: 不是。^^.



九酒大人

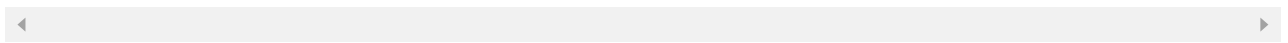
2019-11-14

为什么eval里var的变量可以删除？还是没明白

展开 ▾

作者回复: 因为它在global中声明的属性的configurable为true。

varNames其实并不限制删除，但global.x删除后会同步删除掉varNames。所以如果configurable为false那么就删不掉属性，于是也就删除不掉varNames中的名字（对于使用var声明来说）。



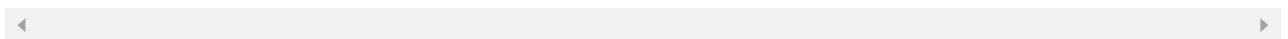
Wiggly Wiggle

2019-11-14

提问：为什么我执行 "var x=100; console.log(Object.getOwnPropertyDescriptor(global, 'x'));" 输出结果是 undefined呢？

这节课的收获就是加深了对“声明”的理解，而且让我意外的是js居然也有静态语义。对于思考题：声明是针对静态编译的，而“非声明”的语句则是运行时的具体执行，个人...
展开 ∨

作者回复: 或许是shell环境的问题？



许童童

2019-11-13

老师讲得太好了，让我又学到了很多以前不知道的东西，感觉学无止境啊。加油

