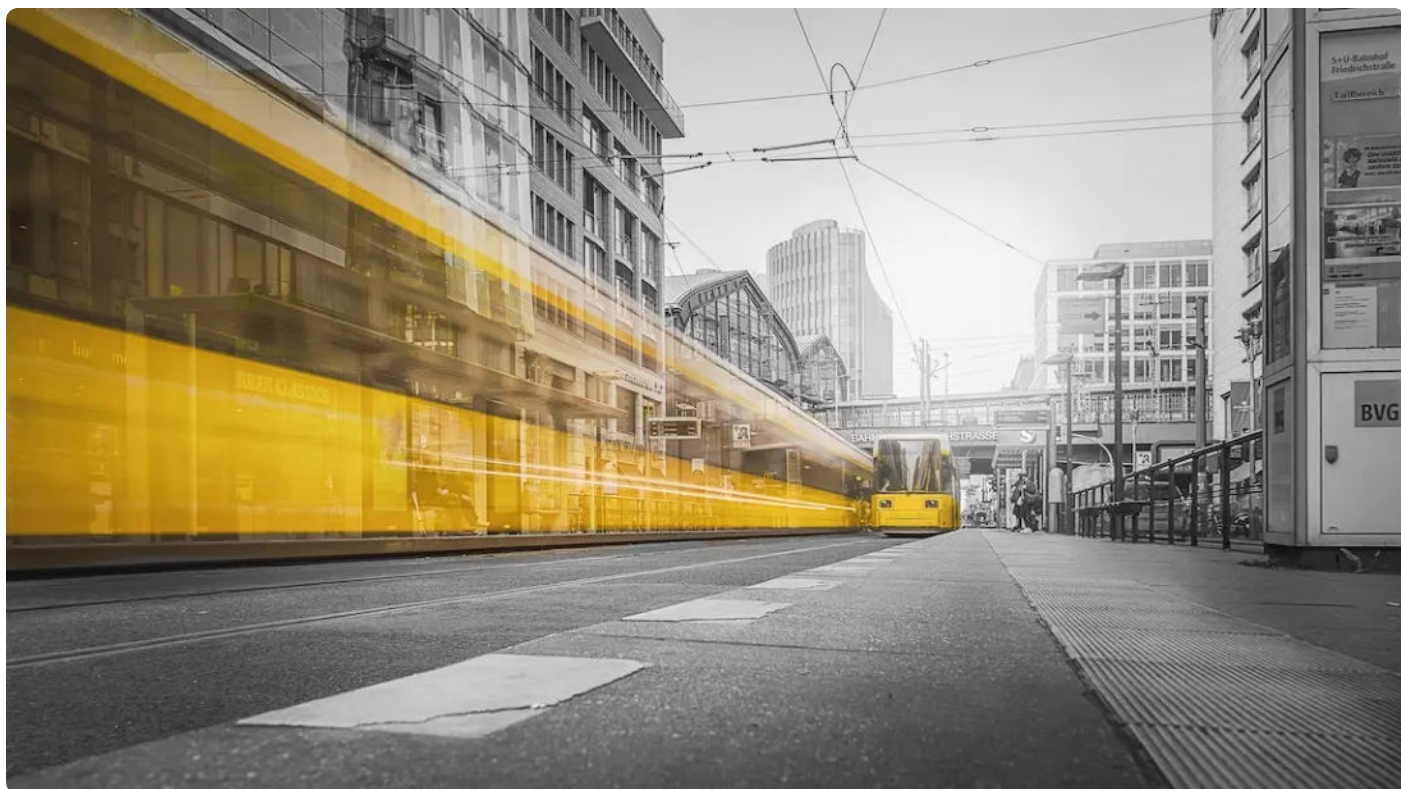


01 | delete 0: JavaScript中到底有什么是可以销毁的

2019-11-11 周爱民

《JavaScript核心原理解析》

[课程介绍 >](#)



讲述：周爱民

时长 18:10 大小 16.65M



你好，我是周爱民，感谢你来听我的专栏。

今天这个系列的第一讲，我将从 JavaScript 中最不起眼的、使用率最低的一个运算——delete 讲起。

你知道，JavaScript 是一门面向对象的语言。它很早就支持了 delete 运算，这是一个元老级的语言特性。但细追究起来，delete 其实是从 JavaScript 1.2 中才开始有的，与它一同出现的，是对象和数组的字面量语法。

有趣的是，JavaScript 中最具恶名的 typeof 运算其实是在 1.1 版本中提供的，比 delete 运算其实还要早。这里提及 typeof 这个声名狼藉的运算符，主要是因为 delete 的操作与类型的识别其实是相关的。



习惯中的“引用”

早期的 JavaScript 在推广时，仍然采用传统的数据类型的分类方法，也就是说，它宣称自己同时支持值类型和引用类型的数据，并且，所谓值类型中的字符串是按照引用来赋值和传递引用（而不是传递值）的。这些都是当时“开发人员的概念集”中已经有的、容易理解的知识，不需要特别解释。

但是什么是引用类型呢？

在这件事上，JavaScript 偷了个懒，它强行定义了“Object 和 Function 就是引用类型”。这样一来，引用类型和值类型就给开发人员讲清楚了，对象和函数呢，也就可以理解了：它们按引用来传递和使用。

绝大多数情况下，这样解释起来是行得通的。但是到了 delete 运算这里，就不行。

因为这样一来，delete 0 就是删除一个值，而 delete x 就既可能是删除一个值，也可能是删除一个引用。然而，当时 JavaScript 又同时约定：那些在 global 对象上声明的属性，就“等同于”全局变量。于是，这就带来了第三个问题：delete x 还可能是删除一个 global 对象上的属性。而它在执行这个操作的时候，看起来却像是一个全局变量（的名字）。

这中间有哪些细节的区别呢？

delete 这个运算的表面意思，是该运算试图销毁某种东西。然而，delete 0 中的 0 是一个具体的、字面量表示的“值”。一个字面量值“0”如何在现实世界中销毁呢？假定它销毁了，那是不是说，在这个语言当前的运行环境中，就不能使用 0 这个值了呢？显然，这不合理。

所以，JavaScript 认为“**所有删除值的 delete 就直接返回 true**”，表明该行为过程中没有异常。很不幸，JavaScript 1.2 的时代并没有结构化异常处理（即 try...catch 语句）。所以，通过函数调用中返回 true 来表明“没有异常”，其实是很常规的做法。

然而，返回值只表明执行过程中没有异常，但实际的执行行为是“什么也没发生”。你显然不可能真的将“0”从执行系统中清理出去。

那么接下来，就只剩下删除变量和删除属性。由于全局变量实际上是通过全局对象的属性来实现的，因此删除变量也就存在识别这两种行为的必要性。例如：



```
1 delete x
```

[复制代码](#)

这行代码究竟是在删除什么呢？出于 JavaScript 是动态语言这项特性，从根本上来说，我们是没办法在语法分析期来判断x的性质的。所以现在，需要有一种方法在运行期来标识x的性质，以便进一步地处理它。

这就导致了一种新的“引用”类型呼之欲出。

到底在删除什么？

探索工作往往如此，是所谓“进五退一”，甚至是“进五退四”。在今后的专栏文章中，你往往会看到，我在碰触到一种新东西的时候会竭力向前，但随后又后退好几步，再来讨论一些更基础层面的东西。这是因为如果不把这些基础概念说得清楚明白，那么往前冲的那几步常常就被带偏了方向。

一如现在这个问题：**delete 0到底是在删除什么？**

对于一门编译型语言来说，所谓“0”，就是上面所述的一个值，它可以是基础值（Primitive values），也可以是数值类型。但如果将这个问题上升到编译之前的、所谓语法分析的阶段，那么“0”就会被称为一个记号（Tokens）。一个记号是没有语义的，记号既可以是语言能识别的，也可以是语言不能识别的。唯有把这二者同时纳入语言范畴，那么这个语言才能识别所谓的“语法错误”。

delete 不仅仅是要操作 0 或 x 这样的单个记号或标识符（例如变量）。因为这个语法实际起作用的是一个对象的属性，也就是“删除对象的成员”。那么它真正需要的语法其实是：

```
1 delete obj.x
```

[复制代码](#)

只不过因为全局对象的成员可以用全局变量的形式来存取，所以它才有了

```
1 delete x
```

[复制代码](#)

这样的语法语义而已。所以，这正好将你之前所认识的倒转过来，是删除 `x` 这个成员，而不是删除 `x` 这个值。不过终归有一点是没错的：既然没办法表达异常，而 `delete 0` 又不产生异常，那么它自然就该返回 `true`。

然而，如果你理解了 `delete obj.x`，那么就一定会想到：`obj.x`既不是之前说过的引用类型，也不是之前说过的值类型，它与 `typeof(x)` 识别的所有类型都无关。因为，它是一个表达式。

所以，`delete` 这个操作的正式语法设计并不是“删除某个东西”，而是“**删除一个表达式的结果**”：

```
1 delete UnaryExpression
```

 复制代码

表达式的结果是什么？

在 JavaScript 中表达式是一个很独特的东西，所有一切表达式运算的终极目的都是为了得到一个值，例如字符串。然后再用另外一些操作将这个值输出出来，例如变成网页中的一个元素（element）。这是 JavaScript 语言创生的原力，也是它的基础设计。也只是因为有了这种设计，它才变得既像面向对象的，又像函数式语言的样子。

表达式的执行特性，以及表达式与语句的关系等等细节，回头我放在第二阶段的内容中讲给你听。现在我们只需要关注一个要点，表达式计算的结果到底是什么？因为就像上面所说的，这个结果，才是 `delete` 这个操作要删除的东西。

在 JavaScript 中，有两个东西可以被执行并存在执行结果（Result），包括语句和表达式。比如你用 `eval()` 来执行一个字符串，那么实际上，你执行的是一个语句，并返回了语句的值；而如果你使用一对括号来强制一个表达式执行，那么这个括号运算得到的，就是这个表达式的值。

表达式的值，在 ECMAScript 的规范中，称为“引用”。

这是一种称为“规范类型”的东西。

规范中的“引用”



实际上这个概念出现得也很早。从 JavaScript 1.3 开始，ECMAScript 规范就在语言定义的层面，正式地将上述的天坑补起来，推出了上面说到的这个“（真正的）引用类型”。

但是，由于这个时候规范的影响力在开发人员中并不那么大，所以开发人员还是习惯性地将对对象和函数称为引用，而其它类型就称为值，并且继续按照传统的理解来解释 JavaScript 中对数据的处理。

这种情况下，一个引用只是在语法层面上表达“它是对某种语法元素的引用”，而与在执行层面的值处理或引用处理没关系。所以，下面这行简短的语句：

```
1 delete 0
```

 复制代码

实际上是在说：JavaScript 将 0 视为一个表达式，并尝试删除它的求值结果。

所以，现在这里的 0，其实不是值（Value）类型的数据，而是一个表达式运算的结果（Result）。而在进一步的删除操作之前，JavaScript 需要检测这个 Result 的类型：

- 如果它是值，则按照传统的 JavaScript 的约定返回 true；
- 如果它是一个引用，那么对该引用进行分析，以决定如何操作。

这个检测过程说明，ECMAScript 约定：任何表达式计算的结果（Result）要么是一个值，要么是一个引用。并且需要留意的是，在这个描述中，所谓对象，其实也是值。准确地说，是“非引用类型”。例如：

```
1 delete {}
```

 复制代码

那么显然，这里要删除的一对大括号是表示一个字面量的对象，当它被作为表达式执行的时候，结果也是一个值。这也是我常常将所有这类表达式称为“单值表达式”的原因，这里并没有所谓的“引用”。



你可以像下面这样，非常细致而准确地解释这一行代码：单值表达式的运算结果返回那个“对象字面量”的单值。然后，`delete`运算发现它的操作数是“值 / 非引用类型”，就直接返回了 `true`。

所以，什么也没有发生。

还会发生什么

那么到底还会发生什么呢？

在 JavaScript 的内部，所谓“引用”是可以转换为“值”，以便参与值运算的。因为表达式的本质是求值运算，所以引用是不能直接作为最终求值的操作数的。这依赖于一个非常核心的、称为“`GetValue()`”的内部操作。所谓内部操作，也称为内部抽象操作（internal abstract operations），是 ECMAScript 描述一个符合规范的引擎在具体实现时应当处理的那些行为。

`GetValue()`是从一个引用中取出值来的行为。这有什么用呢？比如说下面这行代码：

```
1 x = x
```

 复制代码

我们上面说过，所谓 `x` 其实是一个引用。上面的表达式其实是一个赋值表达式，那么“引用 `x` 赋值给引用 `x`”有什么意义呢？其实这在语法层面来解释是非常直接的：

所有赋值操作的含义，是将右边的“值”，赋给左边用于包含该值的“引用”。

那么上面的`x=x`，其实就被翻译成：

```
1 x = GetValue(x)
```

 复制代码

来执行的。而 JavaScript 识别两个 `x` 的不同的方法，就称为“手性”，即是所谓“左手端 (*lhs*, *left hand side*)”和“右手端 (*rhs*)”。它本来是用来描述自然语言的语法中，一个修饰词应该是放在它的主体的前面或是后面的。而在程序设计语言中，它用来说明一个记号（Token）是放



在了赋值符号（例如“=”号）的左边或是右边。作为一个简单的结论，区别上例中的两个 x 的方法就是：

如果 x 放在左边作为 lhs，那么它是引用；如果放在右边作为 rhs，那么就是值。

所以x=x的语义并不是“x 赋给 x”，而是“**把值 x 赋给引用 x**”。

所以，“delete x”归根到底，是在**删除一个表达式的、引用类型的结果（Result）**，而不是在**删除 x 表达式**，或者这个**删除表达式的值（Value）**。

是的，在 JavaScript 中的delete是一个很罕见的、能直接操作“引用”的语法元素。由于这里的“引用”是在 ECMAScript 规范层面的概念，因此在 JavaScript 语言中能操作它的语法元素其实非常少。

然而很不幸，delete 就是其中之一。

告诉我这些有什么用

等等，我想你一定会问了：神啊，让我知道这些究竟有什么用呢？我永远也不会去执行 delete 0这样的操作啊！

是的。但是我接下来要告诉你的事实是：obj.x也是一个引用。对象属性存取是 JavaScript 的面向对象的基本操作之一，所以本质上我们早就在使用“引用”这个东西了，只不过它太习以为常，所以大家都视而不见。

“属性存取（"."运算符）”返回一个关于“x”的引用，然后它可以作为下一个操作符（例如函数调用运算“()”）的左手端来使用，这才有了著名的“**对象方法调用**”运算：

```
1 obj.x()
```

 复制代码

因为在对象方法调用的时候，函数 _x()_ 是来自于obj.x这个引用的，所以这个引用将obj这个对象传递给 x()，这才会让函数 _x()_ 内部通过 this 来访问到 obj。



根本上来说，如果`obj.x`只是值，或者它作为右手端，那么它就不能“携带”`obj` 这个对象，也就完成不了后续的方法调用操作。

对象存取 + 函数调用 = 方法调用

这是 JavaScript 通过连续表达式运算来实现新的语义 / 语法的经典示例。

而所谓“连续运算”其实是函数式运算范式的基本原则。也就是说，`obj.x()`是在 JavaScript 中集合了“引用规范类型操作”“函数式”“面向对象”和“动态语言”等多种特性于一体的一个简单语法。

而它对语言的基础特性的依赖，就在于：

- `delete 0`中的这个`0`是一个表达式求值；
- `delete x`中的`x`是一个引用；
- `delete obj.x`中`obj.x`是一组表达式连续运算的结果（Result/ 引用）；

于是，我们现在可以解释，当 `x` 是全局对象 `global` 的属性时，所谓`delete x`其实只需要返回`global.x`这个引用就可以了。而当它不是全局对象 `global` 的属性时，那么就需要从当前环境中找到一个名为`x`的引用。找到这两种不同的引用的过程，称为 `ResolveBinding`；而这两种不同的`x`，称为不同环境下绑定的标识符 / 名字。

知识回顾

下一讲我将给你讲述的，就是这个名字从声明到发现的全过程。至于现在，这一讲就要告一段落了。今天的内容中，有一些知识点我来带你回顾一下。

- `delete` 运算符尝试删除值数据时，会返回 `true`，用于表示没有错误（Error）。
- `delete 0` 的本质是删除一个表达式的值（Result）。
- `delete x` 与上述的区别只在于 `Result` 是一个引用（Reference）。
- `delete` 其实只能删除一种引用，即对象的成员（Property）。




所以，只有在`delete x`等值于`delete obj.x`时 `delete` 才会有执行意义。例如`with (obj) ...`语句中的 `delete x`，以及全局属性 `global.x`。

思考题

- `delete x` 中，如果 `x` 根本不存在，会发生什么？
- `delete object.x` 中，如果 `x` 是只读的，会发生什么？

希望你喜欢我的分享。

分享给需要的人，Ta购买本课程，你将得 20 元

 生成海报并分享

 赞 19  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 开篇词 | 如何解决语言问题？

下一篇 02 | `var x = y = 100`：声明语句与语法改变了JavaScript语言核心性质



JVM + NIO + Spring

各大厂面试题及知识点详解

限时免费 



精选留言 (89)

 写留言



海绵薇薇

2019-11-19

老师好，我又来了:-)

1.

`delete 0`

这里的0是一个值（就当前情况），而不是引用是吗？

2.

`delete x (x不存在)`

返回true

x 表达式返回的应该是一个引用，并且环境中并没有表示这个引用不能被删除，这个理解对吗？

但是文章中有提到delete只能删除属性这一种引用，糊涂了，估计这里的理解还是有问题。



3.

`delete null` 返回`true`

`delete undefined` 返回`false` 为啥啊？不都是值吗？

4. 还想知道昨天提问的1和2两条是不是漏洞百出啊，就想知道个结果😁。

作者回复: Oh~ 哈哈，你是说昨天有一个问题我只回复了3，没有回复1和2两条吗？那两条，是全对的，所以.....嗯嗯，我只是没有回复确认而已。你对ECMAScript中的“引用规范类型”的使用场景和过程推演都是正确的。

关于今天的前3个问题，1是正确的。

2你也是对的。但是有一点，这个`x`的确会得到一个引用，称为（`UnresolvableReference`）。而这一段逻辑在ECMAScript里面写的是“`if IsUnresolvableReference, then return true`”。也就是说，ECMAScript约定对于这种情况就是这么返回的，这属于规范约定（并且如果在这时发现是严格模式，就抛异常了）。关于这里，你可以看一看：

<https://tc39.es/ecma262/#sec-delete-operator-runtime-semantics-evaluation>

不过问题3，你倒是提到了一个“少有人知”问题，哈哈，这个问题我是漏讲了，而且其实还挺有趣、挺关键的。

是这样，早期的JavaScript中，`undefined`是一个特殊值，是在运行期中通过`void`运算，或者不返回值的函数，又或者一个声明了但未赋值的变量，等等类似这样的情况来“计算得到”的。所以在JavaScript的早期版本中，你没有办法直接判断“`undefined`是`undefined`”，例如无法写出“`x === undefined`”这样的代码，而你只能写类似“`typeof(x) === 'undefined'`”这样的代码。

后来（其实也没有太久），规范就约定把`undefined`作为可以缺省访问的名字，类似于`null`。但是这个时候就带来了一个矛盾，因为这个`undefined`很重要，早期的绝大多数框架或引擎都把它作为一个“全局名字”给声明了。也就是说，ECMAScript现在既没有办法将它规范成一个`keyword`，也没有办法处理成保留字等等，它看起来像`null`，但又没有办法在规范层面强制它。所以.....ECMAScript就搞了一个“奇招”：

> 我们把`undefined`声明成全局的属性，怎么样？！

嗯嗯，很好。所以你看，现在的引擎上面`undefined`看起来长得跟`null`值差不多，而且在ECMAScript规范中它们都还是平级的（是原始值），而且它们的作用也很接近，最后他们都还是从最初的JavaScript 1.x中就存在的概念，但是`undefined/null`两者却在实现上完全不同：`undefined`是一个全局属性，而`null`是一个关键字。



由于undefined是全局属性，所以`delete undefined`其实就是`delete global.undefined`，是删除引用，而不是删除值。而这个属性是只读的，所以就返回false了。

例如你可以试试下面的代码：

```
> Object.getOwnPropertyDescriptor(global, 'undefined')
{ value: undefined,
  writable: false,
  enumerable: false,
  configurable: false }
```

共 10 条评论 >

👍 89



海绵薇薇

2019-11-22

hello 老师好，感谢老师之前的回答：)

突然想到，访问不存在的变量x报ReferenceError错误，其实是对x表达式的Result引用做getVale的时候报的错误，然后为啥typeof x和delete x不报错，因为这两个操作没有求值。

作者回复: 强烈点赞！你这个就属于一通百通的例子。弄明白了Result用来做引用和值的方法/原理，一些具体现象就迎刃而解了！

^^.

共 6 条评论 >

👍 53



潇潇雨歇

2019-11-11

1、如果x根本不存在，delete x什么也不做，返回true

2、如果x只读，delete object.x不能删除掉x属性，返回false；如果在严格模式下，会报错：TypeError: Cannot delete property 'c'

作者回复: 赞的！+1

其实第1个问题的潜在问题是：这种情况下，x是什么呢？它显然是语法可以识别的东西，但如果这样，在语法上它是什么，且在执行环境中它又是什么？

而第二个问题的答案，其实也会回到第一个问题上。如果是在严格模式上，第一个问题的答案是什么？并且，为什么它们不同？

所以，呵呵，其实细一点的看，这两个问题还可以挖更多的呢。^^.



**潇潇雨歇**

2019-11-11

关于delete的知识，大家可以看下MDN的讲解：<https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Operators/delete>

以及这篇深入delete博客：<http://perfectionkills.com/understanding-delete/>

作者回复: 谢谢 @潇潇雨歇

共 7 条评论 >

👍 20

**SOneDiGo**

2019-11-22

想问下老师如何理解用delete处理array element实际上在底层是如何操作的？

例如：array = [1,2,'1']

为什么 delete array[2] 后数组就成了[1,2,undefined/empty]？

作者回复: 对于array来说，你理解为一个普通对象就可以了，只是一些array原型上的方法能帮助你处理array.length这个属性而已。

有多少个有效的element，那么就有多少个同名的（数字下标的）属性；而array.length记录着这个最大值。除了这一点，没有任何与其它对象不同。

所以你用array.pop()或array.push()等操作，甚至直接使用array[i]都可以影响到array.length这个属性——因为这些操作内部都会处理它。但是，你用delete去根本不会处理这个属性——因为delete是把array[i]当一个一般属性处理的，根本不知道array.length的存在。

例如：

...

```
> x = new Array(8)
> x.length
8
> x[x.length] = 8 // add to last
> x.length
9
> x.push(10) // push
10

> x.pop() // pop
> x.length
9
```



```
> delete x[8]
true
> x.length
9
...
```

至于删除`delete array[2]`，则`array[2]`位置上是`undefined`，这个与`delete`操作无关。而是因为你“读取一个不存在的属性，它的值就是`undefined`”。



👍 17



海绵薇薇

2019-11-18

感谢老师指点😊

ref：语法上的引用

我又看了几遍文章并根据提供的连接，得出如下结论：

1.

```
var x
```

```
x = 0
```

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

`x` 表达式返回的是一个`ref`（`{referencedName: 'x', base: Environment Record}`），然后计算值`getValue(ref)`得到具体的值，具体的值会分为传统意义上的基本类型和引用类型

2. 衍生出下面的猜想

```
var obj
```

```
obj = {a: 1}
```

```
console.log(obj.a)
```

`obj.a` 也是一个`ref`（`{referencedName: 'a', base: obj}`），然后计算值的时候`getValue(ref)`得到具体的值1

3. 关于表达式的结果`Result`的疑问。



文中说：表达式的值，在 ECMAScript 的规范中，称为“引用”。（表达式的结果（Result）是引用。）

但是后文说Result可能是引用/值。

这里的值我不能很好的理解。值指的是另一种引用的格式吗？例如链接文档中提到的base其实有很多种值 undefined, Object, a Boolean, a String, a Symbol, a Number。值指的是{base: 0}这种引用吗？如果不是这样的话base的Boolean等基本值类型有啥用啊？

还是说 0 这个表达式的Result就是0这个值？

期待老师的指点😁

作者回复: 关于3，我一般是用Result来表达它是表达式执行结果的“未决状态”。就是执行出结果来了，但没确定是作为lrs还是rhs，所以这种情况下，它就是未决的。

当你确定了一个Result用作lrs，那么它就是引用；如果确定它用作rhs，那么它就是值（将由引擎隐式地调用`GetValue()`）。



👍 17



铭

2019-11-11

乍一读，云里雾里。翻了文档并做测试，总结如下：

delete 操作符用于删除对象的属性，它接收一个表达式，该表达式应返回对象属性的引用。

1. 如果表达式返回的结果是引用：

当该引用是 let 或 const 定义的，delete 执行结果总是 false；

当引用作为对象的属性不存在时，delete 对象的属性，执行结果为 true，表示未处理；

当该引用为 window 对象的属性且是 var 定义的，delete window 对象的属性，执行结果为 false，表示处理失败（获取属性描述符时为不可配置）；

如果在全局环境下显示定义一个属性描述符为可配置的全局属性，执行 delete，结果是 true，表示操作成功；

当该引用为非 window 对象的属性且是 var 定义的，delete 非 window 对象的属性，执行结果为 true，表示处理成功（获取属性描述符时为可配置）。

2. 如果表达式返回的结果是值，如数字、字符串等，delete 执行结果为 true，表示未处理。



作者回复: 其实这一讲的核心是关于“引用/值”在ECMAScript规范类型中的使用与理解, 而不是 (不仅仅是) delete 的使用。所以呢, 解释delete这个操作的种种现象, 最好是在ECMAScript规范所讨论的语言模型中来叙述, 这样更容易讲得清楚。

比如说, `x` 如果是一个属性 (包括是global的属性), 那么`delete x` 的是否成功就取决于属性描述符, 以及属性存取的过程 (是否在严格模式中等等)。这样就Ok了, 而不需要细致地列举每一种情况。

共 4 条评论 >

👍 16



潇潇雨歇

2019-11-16

看的第三遍。还是要去看看规范加深理解。

如果x根本不存在, delete x操作时, x首先是一个表达式, 语义上是一个引用, 然后去寻找该引用的result, 但是x根本不存在, 是找不到的。也就做不了什么, 返回ture。

如果obj.x是只读的或者不可配置的, 表示他是不能删除的, 但是他是实实在在的引用, 是可以求值得到Result的, 所以返回false。表示不能删除。

作者回复: :)

+1



👍 14



Wiggle Wiggle

2019-11-12

即便 obj.x 是一个 function, 当 obj.x 作为右手端时, 也会被 GetValue 方法抽取出值来, 而这个“值”并不是直觉上的数字或字符串。这里是有恍然大悟的感觉的, “值”和“引用”应当从严格的规范定义层面理解, 而不能从直觉上来理解, 只要满足定义, 那就是“值”/“引用”。

作者回复: 赞的! 就是这样!



👍 13



隔夜果酱

2019-11-11

既然delete这么鸡肋, 只能删除对象的成员.

那么后来的版本中为什么不进行改进呢?

比如限定其只能用delete obj.x这种语法格式.

或者加入trycatch, 对删除value的操作直接报错呢?



作者回复: 这个问题就牵扯得大了。

最早javascript中是没有明确、显式的global这个对象的，在宿主环境（例如浏览器中）你可以用window.x去访问它，这算是宿主在实现引擎的时候的约定。但是，仅仅从引擎的角度上来说，既没有window，也没有global，更没有Global。所以，全局的变量虽然是作为全局属性名存在着，却没有办法写成global.x这样的引用。

而global这个全局名字，直到现在在ECMAScript中都还是个没被规范的东西。TC39有一个提案（<https://github.com/tc39/proposal-global>）专门来定义它，现在到了stage3，应该不会被否决了。但即使如此，这个东东也不叫global，而改名成了globalThis。——原本提案阶段是叫global的，但应用中有问题，所以就改了。

关于globalThis这个说法，又得是一段历史了。因为早期的JavaScript约定普通函数在“不作为对象方法调用”的时候，this值默认指向这个全局的global。所以，这也就是著名的代码“global = (new Function('return this'))”，或“global = Function('return this')()”的由来。

^^.



13



桃翁

2020-03-19

我突然 明白了 (obj.func=obj.func)()这种方式会丢掉obj里面的this，因为等号右边的obj.func是值，所以得到的仅仅是个函数这个值，而不是引用。老师我理解得是对的吗？

作者回复: Yes! +5

共 3 条评论 >

10



渭河

2019-11-21

这句话要怎么理解呀

所谓值类型中的字符串是按照引用来赋值和传递引用（而不是传递值）的

作者回复: 这就是“传统中的‘引用’”用来解释这类现象的时候出现的麻烦。很典型的一个例子，话表达的是正确的，内容是正确的，说法也正确，就是特别特别难于理解。

首先，“值类型中的字符串”是指什么呢？是指typeof(x) === 'string'中的那个`x`。在传统的javascript概念中，这样的x是值类型，而不是引用类型。



那么值“该怎么赋值和传递”呢？如果x的值是1，那么 $y = x$ 的话，就是把1这个值“抄写”到y里面去。这是“正常的值”的处理方法，但是如果“字符串值”也这么处理，就完蛋了，因为字符串可能无数多个字符，那么当 $y = x$ 按照“正常的值处理方法”来实现的话，这个“值的复制”的开销就受不了。

所以：

1. “值类型中的字符串”，是指照
2. “引用来赋值和传递引用”的；且，
3. 它是只传递引用（而不是传递值）的。

如上。只是说起来特别麻烦而已。

共 2 条评论 >

👍 10



余文郁

2019-11-11

老师，JS是基于对象的语言，不是面象对象的语言吧，感觉第二段这有点不妥，虽然ES6增加了class语法，但只是原型的语法糖而已

作者回复：在后面我会再着重地讲到JavaScript对面向对象的理解。

如今我们对OOP的理解其实添加了太多应用的色彩。事实上，JavaScript对OOP的理解是很精彩、很学术，以及很完整的。不过这些内容大概要到第11讲之后了。

至于“面向对象”还是“基于对象”，其实JavaScript 1.0是有类而无继承的，而JavaScript 1.1才开始使用原型来实现继承，这个时候它又抛弃了（严格意义上的）类。

当然，上面看起来有点儿绕着你的问题在讲。所以，如果再确切地、准确无误地回复你的问题，那么应该是说：所谓面向对象的三个原则（封装、继承与多态），在严格意义上，后两者是多余的。所以不必过度去强调这些性质之于面向对象的重要性。

共 4 条评论 >

👍 10



ssala

2019-11-14

关于delete，搜集了一些资料，结合代码测试，我目前是这样理解的：delete为一元操作符，其操作数为一个表达式，如果表达式的求值结果是一个值，那么`delete 值`直接返回true，表示该操作没有异常。如果表达式求值结果是一个引用，那么`delete 引用`则会有如下表现，如果引用是可删除的，则直接删除该引用，返回true，否则返回false。



关于属性/property的可删除特性，参照这篇文章：<http://perfectionkills.com/understanding-delete/>

关于引用和值的理解，我用段代码说明，如下：

...

```
var x = {a: 20}
```

...

代码中，x是引用，它"指向"执行系统中{a: 20}的一个对象，而{2: 20}则是值，它对应执行系统中内存上的一块区域。x.a是引用，它"指向"执行系统中内存20这个值，而20是值，它也对对应执行系统中内存上的一块区域。因此：

...

`delete x // `false``，x为表达式，求值结果为`global.x`，且该属性是用`var`来声明的，其特性是不可删除

`delete 20 // `true``，当执行系统遇到20字面量时，认为其为表达式，对其求值以后得到20这个值，`delete` 值返回`true`

`delete x.a // `true`` x.a 为引用，且可以删除

...

另外关于`delete x`，若x不存在，我的解释是：x为表达式，由于未定义，表达式求值结果是未定义的，但是虽然未定义，但求值结果仍然是值，而`delete` 值就返回`true`。不知这种解释是否正确？

作者回复: Yes. 对的。

其实只要理解到`delete {}`中的对象字面量其实是“值”，那么就一通百通了。

共 2 条评论 >

👍 8



blueBean

2020-02-21

表达式的值，在 ECMAScript 的规范中称为“引用”。

ECMAScript 约定：任何表达式计算的结果（Result）要么是一个值，要么是一个引用。

上面这两句话矛盾了吧

作者回复: 并不是矛盾，只是这里解释起来比较别扭。因为“Value”和“Result”，以及“值和引用”在上下文中都存在多种含义。

【第一句】

...

表达式的值，在 ECMAScript 的规范中称为“引用”。



这一句讲的时候，上下文中是将表达式与语句放在一起讨论的。原文是“（你）执行的是一个语句，那么……；而如果你使用……表达式执行，那么……”。前者是语句的值，后者表达式的值。——它们都分别是“一个称为结果（Result）的东西”。

这样对比来讲的时候，我向来会解释成：

> 语句和表达式都是有值的，语句的值是“完成（规范类型）”，而表达式的值是“引用（规范类型）”。

这个区别在后面的章节里面还会有，而且也还会这么讲。主要是这样讲起来清晰、简单，分别起来也很容易。——但是，这样讲并不“准确”。因为事实上表达式的“结果（Result）”也可以是完成类型，而语句的结果还包括一个所谓的“Empty”值。

在第一章中，要把所有关系到的概念讲清楚是很难的，真要那样讲概念，大概也让人读不下去。所以这里说的是一个简单的区分，也就是如何区别“表达式的值 vs 语句的值”。

【第二句】

ECMAScript 约定：任何表达式计算的结果（Result）要么是一个值，要么是一个引用。

这一句是完整而正确的。但是如同上面的讨论中所说的，它其实也并不“绝对完整”，因为有一部分表达式事实上是在返回“完成（规范类型）”。只不过当这种情况发生时，后续的计算过程会从“完成（规范类型）”中直接取值，因此在计算过程中感觉不到“非值”的结果（Result），这是一种中间状态。

总之，这些内容在后续的章节中还会介绍。会逐渐更新和补全。第一章，以及前几章的内容，要通贯起来看，有很多地方的写法或者讲法，是不得以而（暂且）为之的。关于这一点，我在“加餐（选学的章节）”里面说过，也就可以先略过去，看不明白，或者看起来矛盾的地方，后面再读到的时候，就了解了。



7



仰望星空

2019-11-11

老师的英语发音delete偏差的有点多

作者回复: 这个这个，惭愧呀惭愧~

我的口语不是一点半点的糟糕（当然，其实不仅仅只是口语糟糕）。我尽量……注意……后面的课程



~ 多谢多谢~

惭愧呀~

:(~



👍 7



Mr_Liu

2019-11-12

思考题1: delete x x不存在返回的是true

2: 删除会返回false,严格模式会报错

第一遍听感觉有些云里雾里的感觉，又听了一遍加实践。但是有一点不理解或者不知道理解的对不对，希望老师解答一下

问题一：

例如var a = '123' delete a 返回的是false，

再次输入a 得到结果依然是 '123'，

这是说明delete 没有起作用，其没有起作用的原因是因为 var a = '123' 中的a 是基本数据类型，不是引用类型，所以删除a 元素失败，借此印证了所讲的delete 删除的是表达式或者引用类型的结果。印证这句话的另一个例子是：

```
var obj = {  
  a: '123'  
},
```

```
var b = obj.a
```

delete b 返回false，因为b = obj.a 属于一个赋值语句，b 也是个基本数据类型，所以也不起作用

那么修改成

```
var obj = {  
  a: '123',  
  b: {  
    name: '123'  
  }  
}
```

```
var val = obj.b
```

delete val 返回的依然是false 后来会读了一下，有这样一句话：delete 其实只能删除一种引用，即对象的成员（Property）

那么 delete x 还有什么存在的意义么。

问题二：

接着我使用delete obj.a 返回的是true，再次输入 obj.a 返回的就是undefined

但如果我使用



```
var val = obj.b
```

delete obj.b 返回的是true

然后打印 obj.b 为undefined; val 为 {name: '123'}

，那老师的那句delete实际上是删除一个表达式的、引用类型的结果（Result），而不是在删除 x 表达式，或者这个删除表达式的值（Value）。是否可以理解为实际是删除一直引用呢。

作者回复: 问题1中，你的思考方向错了。`delete a`不起作用的原因是`var`声明导致的，而不是因为`a`是基本数据类型。举例来说，

```
`with (x = {a: 100}) delete a;`
```

这个例子的结果中x.a是不存在的，但`a`也是基本数据类型`呀。所以是无关的。

“delete x 还有什么存在的意义么”这个问题我之前回复过另一个留言，你找找。

关于问题二，关键在于你所理解的“引用与值”，跟JavaScript内部所理解的“引用与值”是不一样的。也正是因此，我在这一讲的一开始用大量文字讨论了二者的区别。简单地来说，如果有表达式`x = x`，那么同一个变量`x`，在上述表达式中，左侧的这个是它的引用，左侧的是它的值。如果放在代码中看：

```
x = 5; // 在JavaScript语言中，'5'是“值类型”
```

```
x = x; // 在ECMAScript规范中，左侧是“引用x”，右侧是“值x”。
```

我一直用“结果（Result）”来强调表达式“表达式计算的结果”，就是因为对于`x = x`来说，左侧和右侧都是表达式，左侧的结果是“lhs/引用（reference）”，而右侧的结果是“rhs/值（value）”。

所以所谓“结果（Result）”，在不明确它的手性或用处之前，是二个意思都包含的。

共 2 条评论 >

👍 6



Smallfly

2019-11-27

在 ECMAScript 规范中，引用的构成至少需要 base value、referenced name、strict reference flag，具体引擎实现应该会把它们封装成一种数据结构来，从而来操作引用。

而文中把 $x = x$ 中的 x 叫做一个引用，应该不是很精确， x 只是引用的 referenced name。

因为我们代码层面无法获取引用，也就没有名字，所以这里用 x 指代规范中的引用。

请问老师这样理解对么？



作者回复: > 在 ECMAScript 规范中, 引用的构成至少需要 base value、referenced name、strict reference flag, 具体引擎实现应该会把它们封装成一种数据结构来, 从而来操作引用。

在ECMAScript中, 它就是一种数据结构啊。我们称byte/word/array/map为数据结构, 为什么“Specification Types”就不是呢? 你看ECMAScript规范里面, “Specification Types”就是在“Data Types”这一章中的一节啊。

> 而文中把 $x = x$ 中的 x 叫做一个引用, 应该不是很精确, x 只是引用的 referenced name。不对。左侧的 x 就是引用, 而不仅仅是referenced name。只是代码文本 (在静态读代码的情况下) 它是个名字 x 而已。

> 因为我们代码层面无法获取引用, 也就没有名字, 所以这里用 x 指代规范中的引用。代码层面是可以获取所谓“引用”的, 而且是“规范类型”中的引用。例如delete obj.x中的`obj.x`整体上就是一个引用。



👍 5



Marvin

2019-11-20

关于文中delete x的解释, 我有一点疑问。

文中是这样说的:

于是, 我们现在可以解释, 当 x 是全局对象 global 的属性时, 所谓delete x其实只需要返回global.x这个引用就可以了。而当它不是全局对象 global 的属性时, 那么就需要从当前环境中找到一个名为 x 的引用。找到这两种不同的引用的过程, 称为 ResolveBinding; 而这两种不同的 x , 称为不同环境下绑定的标识符 / 名字。

如果把 x 解释为引用, 而且先寻找global.x, 当不是全局属性再寻找当前环境的话:

...

```
window.apple = 10;
let apple = 10;
delete apple; // false
...
```

上面的代码应该先去全局寻找apple引用, 那么删除就成功了, 应该返回true才对, 而不是false。

作者回复: 哦。确实是这样的。

但是这个问题与delete运算符无关, 这个取决于`delete`将`apple`作为一个名字被“发现(resolving)”的过程。

由于全局环境的作用域是由global对象和一个词法环境 (共同) 构成的, 所以它查找上面这个名字的顺序是先词法声明, 然后才是global对象上的属性的。这个部分请参见ECMAScript:



<https://tc39.es/ecma262/#sec-global-environment-records-getbindingvalue-n-s>

```
> If DclRec.HasBinding then return DclRec.GetBindingValue();  
> else return ObjRec.GetBindingValue()
```



👍 5



半橙汁

2019-11-11

在《你不知道JavaScript-上》中，看到过关于lhs和rhs的相关介绍，涉及到很多编译，语法解析的知识，真的很难肯...

希望通过对老师专栏的学习，能够更加顺畅地去啃另外的中、下两本😂😂😂

作者回复: 那三本书很不错! 值得一读。^^.



👍 5

