链滴

function与感叹号

作者：K

原文链接：https://ld246.com/article/1445236168503

来源网站：链滴

#function与感叹号

最近有空可以让我静下心来看看各种代码，function与感叹号的频繁出现，让我回想起2个月前我回州最后参加团队会议的时候抛出的一样问题：*如果在function之前加上感叹号 (!) 会怎么样？*比如面的代码：

```
!function(){alert('iifksp')}()        // true
```

在控制台运行后得到的值时true，为什么是true这很容易理解，因为这个匿名函数没有返回值，默认回的就是undefined，求反的结果很自然的就是true。所以问题并不在于结果值，而是在于，为什么反操作能够让一个匿名函数的自调变的合法？

平时我们可能对添加括号来调用匿名函数的方式更为习惯：

```
(function(){alert('iifksp')})()        // true
```

或者：

```
(function(){alert('iifksp')}())        // true
```

虽然上述两者括号的位置不同，不过效果完全一样。

那么，是什么好处使得为数不少的人对这种叹号的方式情有独钟？如果只是为了节约一个字符未免太有必要了，这样算来即使一个100K的库恐怕也节省不了多少空间。既然不是空间，那么就是说也许有时间上的考量，事实很难说清，文章的最后有提到性能。

回到核心问题，为什么能这么做？甚至更为核心的问题是，为什么必须这么做？

其实无论是括号，还是感叹号，让整个语句合法做的事情只有一件，**就是让一个函数声明语句变成了个表达式。**

```
function a(){alert('iifksp')}        // undefined
```

这是一个函数声明，如果在这么一个声明后直接加上括号调用，解析器自然不会理解而报错：

```
function a(){alert('iifksp')}()        // SyntaxError: unexpected_token
```

因为这样的代码混淆了函数声明和函数调用，以这种方式声明的函数 a，就应该以 a(); 的方式调用。

但是括号则不同，它将一个函数声明转化成了一个表达式，解析器不再以函数声明的方式处理函数a而是作为一个函数表达式处理，也因此只有在程序执行到函数a时它才能被访问。

所以，**任何消除函数声明和函数表达式间歧义的方法，都可以被解析器正确识别。** 比如：

```
var i = function(){return 10}();        // undefined
1 && function(){return true}();        // true
1, function(){alert('iifksp')}();        // undefined
```

赋值，逻辑，甚至是逗号，各种操作符都可以告诉解析器，这个不是函数声明，它是个函数表达式。

且，对函数一元运算可以算的上是消除歧义最快的方式，感叹号只是其中之一，如果不在乎返回值，些一元运算都是有效的：

```
!function(){alert('iifksp')}()        // true
+function(){alert('iifksp')}()         // NaN
-function(){alert('iifksp')}()        // NaN
~function(){alert('iifksp')}()         // -1
```

甚至下面这些关键字，都能很好的工作：

```
void function(){alert('iifksp')}()        // undefined
new function(){alert('iifksp')}()        // Object
delete function(){alert('iifksp')}()        // true
```

最后，括号做的事情也是一样的，消除歧义才是它真正的工作，而不是把函数作为一个整体，所以无括号括在声明上还是把整个函数都括在里面，都是合法的：

```
(function(){alert('iifksp')})()        // undefined
(function(){alert('iifksp')}())        // undefined
```

说了这么多，实则在说的一些都是最为基础的概念——语句，表达式，表达式语句，这些概念如同指与指针变量一样容易产生混淆。虽然这种混淆对编程无表征影响，但却是一块绊脚石随时可能因为它头破血流。

最后讨论下性能。jsperf上有个简单的测试：http://jsperf.com/js-funcion-expression-speed ，可用不同浏览器访问，运行测试查看结果。

Testing in Chrome 45.0.2454.101 32-bit on Windows Server 2008 R2 / 7 64-bit :

Test		Ops/sec
!	!function(){;}()	58,066,858 ±10.04% fastest
+	+function(){;}()	19,809,475 ±0.00% 66% slower
-	-function(){;}()	23,140,500 ±0.00% 60% slower
~	~function(){;}()	25,184,462 ±0.00% 57% slower
(1)	(function(){;})()	24,721,284 ±0.00% 57% slower
(2)	(function(){;}())	24,484,500 ±0.00% 58% slower
void	void function(){;}()	18,754,500 ±0.00% 68% slower
new	new function(){;}()	203,331 ±31.40% 100% slower
delete	delete function(){;}()	21,986,410 ±0.00% 62% slower
=	var i = function(){;}()	30,423,683 ±0.00% 48% slower
&&	1 && function(){;}()	18,244,037 ±0.00% 69% slower
||	0 || function(){;}()	20,194,176 ±0.00% 65% slower
&	1 & function(){;}()	20,589,088 ±0.00% 65% slower
|	1 | function(){;}()	18,608,678 ±0.00% 68% slower
^	1 ^ function(){;}()	22,964,975 ±0.00% 60% slower
,	1, function(){;}()	20,368,629 ±4.33%

Testing in IE 8.0 on Windows Server 2008 R2 / 7 64-bit:

	Test	Ops/sec
!	`!function(){;}()`	1,944,404 ±1.06% fastest
+	`+function(){;}()`	1,893,450 ±0.65% 6% slower
-	`-function(){;}()`	1,886,197 ±1.22% 6% slower
~	`~function(){;}()`	1,924,617 ±0.53% fastest
(1)	`(function(){;})()`	1,936,444 ±0.90% fastest
(2)	`(function(){;}())`	1,984,268 ±0.95% fastest
void	`void function(){;}()`	1,963,969 ±0.68% fastest
new	`new function(){;}()`	553,332 ±1.23% 73% slower
delete	`delete function(){;}()`	ERROR
=	`var i = function(){;}()`	1,963,836 ±0.69% fastest
&&	`1 && function(){;}()`	1,907,606 ±0.66% fastest
||	`0 || function(){;}()`	1,912,608 ±0.86% fastest
&	`1 & function(){;}()`	1,865,614 ±0.73% 7% slower
|	`1 | function(){;}()`	1,949,419 ±0.89% fastest
^	`1 ^ function(){;}()`	1,924,477 ±0.89% fastest

Testing in Firefox 40.0 32-bit on Windows Server 2008 R2 / 7 64-bit

Testing in Firefox 40.0 32-bit on Windows Server 2008 R2 / 7 64-bit		
Test		Ops/sec
!	`!function(){;}()`	1,939,265,644 ±0.44% fastest
+	`+function(){;}()`	6,128,908 ±0.27% 100% slower
-	`-function(){;}()`	6,077,227 ±0.32% 100% slower
~	`~function(){;}()`	Infinity ±0.00% fastest
(1)	`(function(){;})()`	1,966,933,414 ±0.19% fastest
(2)	`(function(){;}())`	1,950,225,840 ±0.54% fastest
void	`void function(){;}()`	1,966,657,420 ±0.27% fastest
new	`new function(){;}()`	412,737 ±15.72% 100% slower
delete	`delete function(){;}()`	1,953,262,851 ±0.30% fastest
=	`var i = function(){;}()`	1,950,782,510 ±0.42% fastest
&&	`1 && function(){;}()`	1,960,888,144 ±0.29% fastest
||	`0 || function(){;}()`	1,960,110,711 ±0.31% fastest
&	`1 & function(){;}()`	1,960,353,558 ±0.27% fastest
|	`1 | function(){;}()`	1,964,858,909 ±0.23% fastest
^	`1 ^ function(){;}()`	1,972,422,415 ±0.24% fastest
,	`1, function(){;}()`	1,967,743,162 ±0.35% fastest

可见不同的方式产生的结果并不相同，而且，差别很大，因浏览器而异。

但我们还是可以从中找出很多共性：**new方法永远最慢**——这也是理所当然的。其它方面很多差距其不大，但有一点可以肯定的是，感叹号并非最为理想的选择。反观传统的括号，在测试里表现始终很，在大多数情况下比感叹号更快——所以平时我们常用的方式毫无问题，甚至可以说是最优的。加减在chrome表现惊人，而且在其他浏览器下也普遍很快，相比感叹号效果更好。

当然这只是个简单测试，不能说明问题。但有些结论是有意义的：括号和加减号最优。

原文链接：function与感叹号