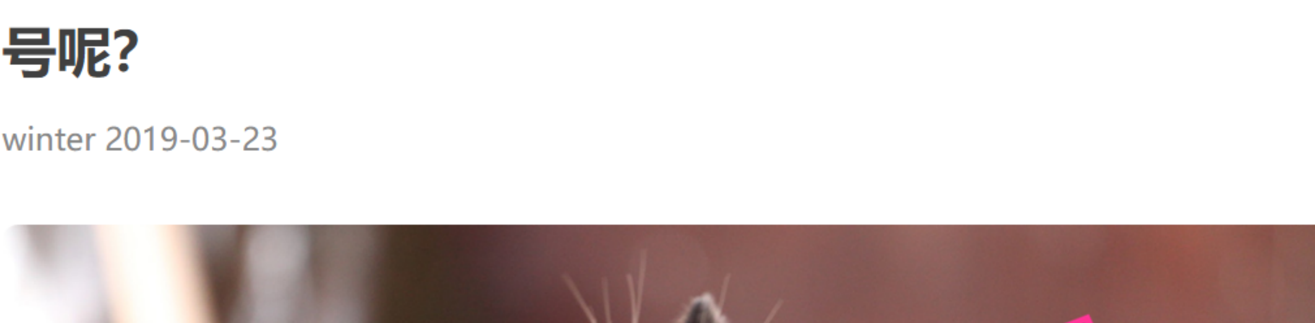
28 | JavaScript语法（预备篇）：到底要不要写分号呢?

winter 2019-03-23



仅供本群内部学习使用
防断更微信：699250

00:00 10:56
讲述：winter 大小：10.02M

你好，我是 winter。

在我们介绍 JavaScript 语法的全局结构之前，我们先要探讨一个语言风格问题：究竟要不要写分号。

这是一个非常经典的口水问题，“加分号”党和“不写分号”党之间的战争，可谓是经久不息。

实际上，行尾使用分号的风格来自于 Java，也来自于 C 语言和 C++，这一设计最初是为了降低编译器的工作负担。

但是，从今天的角度来看，行尾使用分号其实是一种语法噪音，恰好 JavaScript 语言又提供了相对可用的分号自动补全规则，所以，很多 JavaScript 的程序员都是倾向于不写分号。

这里要特意说一点，在今天的文章中，我并不希望去售卖自己的观点（其实我是属于“加分号”党），而是希望比较中立地给你讲清楚相关的知识，让你具备足够的判断力。

我们首先来了解一下自动插入分号的规则。

自动插入分号规则

自动插入分号规则其实独立于所有的语法产生式定义，它的规则说起来非常简单，只有三条。

- 要有换行符，且下一个符号是不符合语法的，那么就尝试插入分号。
- 有换行符，且语法中规定此处不能有换行符，那么就自动插入分号。
- 源代码结束处，不能形成完整的脚本或者模块结构，那么就自动插入分号。

这样描述是比较难以理解的，我们一起看一些实际的例子进行分析：

```
let a = 1
void function(a){
    console.log(a);
}(a);

```

在这个例子中，第一行的结尾处有换行符，接下来 void 关键字接在 1 之后是不合法的，这命中了我们的第一条规则，因此会在 void 前插入换行符。

```
var a = 1, b = 1, c = 1;
a
++
b
++
c

```

这也是个著名的例子，我们看第二行的 a 之后，有换行符，后面遇到了 ++ 运算符，a 后面跟 ++ 是合法的语法，但是我们看看 JavaScript 标准定义中，有 [no LineTerminator here] 这个字样，这是一个语法定义中的规则，你可以感受一下这个规则的内容（下一小节，我会给你详细介绍 no LineTerminator here ）：

```
UpdateExpression[Yield, Await]:
    LeftHandSideExpression[?Yield, ?Await]
    LeftHandSideExpression[?Yield, ?Await][no LineTerminator here]++
    LeftHandSideExpression[?Yield, ?Await][no LineTerminator here]--
    ++UnaryExpression[?Yield, ?Await]
    --UnaryExpression[?Yield, ?Await]

```

于是，这里 a 的后面就要插入一个分号了。所以这段代码最终的结果，b 和 c 都变成了 2，而 a 还是 1。

```
(function(a){
    console.log(a);
})()
(function(a){
    console.log(a);
})()

```

仅供本群内部学习使用
防断更微信：699250

这个例子是比较有实际价值的例子，这里两个 function 调用的写法被称作 IIFE（立即执行的函数表达式），是个常见技巧。

这段代码意图上显然是形成两个 IIFE。

我们来看第三行结束的位置，JavaScript 引擎会认为函数返回的可能是个函数，那么，在后面再跟括号形成函数调用就是合理的，因此这里不会自动插入分号。

这是一些鼓励不写分号的编码风格会要求大家写 IIFE 时必须在行首加分号的原因。

```
function f(){
    return/*
        This is a return value.
    */1;
}
f();

```

在这个例子中，return 和 1 被用注释分隔开了。

根据 JavaScript 自动插入分号规则，**带换行符的注释也被认为是有换行符**，而恰好的是，return 也有 [no LineTerminator here] 规则的要求。所以这里会自动插入分号，f 执行的返回值是 undefined。

no LineTerminator here 规则

好了，到这里我们已经讲清楚了分号自动插入的规则，但是我们要想彻底掌握分号的奥秘，就必须要对 JavaScript 的语法定义做一些数据挖掘工作。

no LineTerminator here 规则表示它所在的结构中的这一位置不能插入换行符。

自动插入分号规则的第二条：有换行符，且语法中规定此处不能有换行符，那么就自动插入分号。跟 no LineTerminator here 规则强相关，那么我们就找出 JavaScript 语法定义中的这些规则。

为了方便你理解，我把产生式换成了实际的代码。

下面一段代码展示了，带标签的 continue 语句，不能在 continue 后插入换行。

```
outer:for(var j = 0; j < 10; j++)
    for(var i = 0; i < j; i++)
        continue /*no LineTerminator here*/ outter

```

break 跟 continue 是一样的，break 后也不能插入换行：

```
outer:for(var j = 0; j < 10; j++)
    for(var i = 0; i < j; i++)
        break /*no LineTerminator here*/ outter

```

我们前面已经提到过 return 和后自增、后自减运算符。

```
function f(){
    return /*no LineTerminator here*/1;
}

```

```
i/*no LineTerminator here*/++
i/*no LineTerminator here*/--

```

以及，throw 和 Exception 之间也不能插入换行符：

```
throw/*no LineTerminator here*/new Exception("error")

```

凡是 async 关键字，后面都不能插入换行符：

```
async/*no LineTerminator here*/function f(){

}
const f = async/*no LineTerminator here*/x => x*x

```

仅供本群内部学习使用
防断更微信：699250

箭头函数的箭头前，也不能插入换行

```
const f = x/*no LineTerminator here*/=> x*x

```

yield 之后，不能插入换行

```
function *g(){
    var i = 0;
    while(true)
        yield/*no LineTerminator here*/i++;
}

```

到这里，我已经整理了所有标准中的 no LineTerminator here 规则，实际上，no LineTerminator here 规则的存在，多数情况是为了保证自动插入分号行为是符合预期的，但是令人遗憾的是，JavaScript 在设计的最初，遗漏了一些重要的情况，所以有一些不符合预期的情况出现，需要我们格外注意。

不写分号需要注意的情况

下面我们来看几种不写分号容易造成错误的情况，你可以稍微注意一下，避免发生同样的问题。

以括号开头的语句

我们在前面的案例中，已经展示了一种情况，那就是以括号开头的语句：

```
(function(a){
    console.log(a);
})()/* 这里没有被自动插入分号 */
(function(a){
    console.log(a);
})()

```

这段代码看似两个独立执行的函数表达式，但是其实第三组括号被理解为传参，导致抛出错误。

以数组开头的语句

除了括号，以数组开头的语句也十分危险：

```
var a = [[]]/* 这里没有被自动插入分号 */
[3, 2, 1, 0].forEach(e => console.log(e))

```

这段代码本意是一个变量 a 赋值，然后对一个数组执行 forEach，但是因为没有自动插入分号，被理解为下标运算符和逗号表达式，我这个例子展示的情况，甚至不会抛出错误，这对于代码排查问题是个噩梦。

以正则表达式开头的语句

正则表达式开头的语句也值得你去多注意一下。我们来看这个例子。

```
var x = 1, g = {test:()=>0}, b = 1/* 这里没有被自动插入分号 */
/(a)/g.test("abc")
console.log(RegExp.$1)

```

这段代码本意是声明三个变量，然后测试一个字符串中是否含有字母 a，但是因为没有自动插入分号，正则的第一个斜杠被理解成了除号，后面的意思就都变了。

注意，我构造的这个例子跟上面的例子一样，同样不会抛错，凡是这一类情况，都非常致命。

以 Template 开头的语句

以 Template 开头的语句比较少见，但是跟正则配合时，仍然不是不可能出现：

```

var f = function(){
  return "";
}
var g = f/* 这里没有被自动插入分号 */
`Template`.match(/(a)/);
console.log(RegExp.$1)

```

仅供本群内部学习使用
防断更微信：699250

这段代码本意是声明函数 f，然后赋值给 g，再测试 Template 中是否含有字母 a。但是因为没有自动插入分号，函数 f 被认为跟 Template 一体的，进而被莫名其妙地执行了一次。

总结

这一节课，我们讨论了要不要加分号的问题。

首先我们介绍了自动插入分号机制，又对 JavaScript 语法中的 no line terminator 规则做了个整理，最后，我挑选了几种情况，为你介绍了不写分号需要注意的一些常见的错误。

最后留给你一个问题，请找一些开源项目，看看它们的编码规范是否要求加分号，欢迎留言讨论。