

# JavaScript词法：为什么12.toString会报错？

winter 2019-03-19



你好，我是 winter。

在前面的文章中，我们已经从运行时的角度了解过 JavaScript 的知识内容，在接下来的几节课，我们来了解一下 JavaScript 的文法部分。

文法是编译原理中对语言的写法的一种规定，一般来说，文法分成词法和语法两种。

词法规定了语言的最小语义单元：token，可以翻译成“标记”或者“词”，在我的专栏文章中，我统一把 token 翻译成词。

从字符到词的整个过程是没有结构的，只要符合词的规则，就构成词，一般来说，词法设计不会包含冲突。词法分析技术上可以使用状态机或者正则表达式来进行，我们的课程主要是学习词法，关于它们实现的细节就不多谈了。

## 概述

我们先来看一看 JavaScript 的词法定义。JavaScript 源代码中的输入可以这样分类：

WhiteSpace 空白字符

LineTerminator 换行符

Comment 注释

Token 词

- IdentifierName 标识符名称，典型案例是我们使用的变量名，注意这里关键字也包含在内了。
- Punctuator 符号，我们使用的运算符和大括号等符号。
- NumericLiteral 数字直接量，就是我们写的数字。
- StringLiteral 字符串直接量，就是我们用单引号或者双引号引起来的直接量。
- Template 字符串模板，用反引号`括起来的直接量。


这个设计符合比较通用的编程语言设计方式，不过，JavaScript 中有一些特别之处，我下面就来讲讲特别在哪里。

首先是除法和正则表达式冲突问题。我们都知道，JavaScript 不但支持除法运算符“/”和“/=", 还支持用斜杠括起来的正则表达式“/abc/”。

但是，这时候对词法分析来说，其实是没有办法处理的，所以 JavaScript 的解决方案是定义两组词法，然后靠语法分析传一个标志给词法分析器，让它来决定使用哪一套词法。

JavaScript 词法的另一个特别设计是字符串模板，模板语法大概是这样的：

```
1 `Hello, ${name}`
```

 复制代码

理论上，“\${}”内部可以放任何 JavaScript 表达式代码，而这些代码是以“}”结尾的，也就是说，这部分词法不允许出现“}”运算符。

是否允许“}”的两种情况，与除法和正则表达式的两种情况相乘就是四种词法定义，所以你在 JavaScript 标准中，可以看到四种定义：

InputElementDiv;

InputElementRegExp;

InputElementRegExpOrTemplateTail;

InputElementTemplateTail。

为了解决这两个问题，标准中还不得不把除法、正则表达式直接量和“}”从 token 中单独抽出来，用词上，也把原本的 Token 改为 CommonToken。

但是我认为，从理解的角度上出发，我们不应该受到影响，所以在本课，我们依然把它们归类到 token 来理解。

对一般的语言的词法分析过程来说，都会丢弃除了 token 之外的输入，但是对 JavaScript 来说，不太一样，换行符和注释还会影响语法分析过程，这个我们将会语法部分给你详细讲解（所以要实现 JavaScript 的解释器，词法分析和语法分析非常麻烦，需要来回传递信息）。

接下来我来给你详细介绍一下。

## 空白符号 Whitespace

说起空白符号，想必给大家留下的印象就是空格，但是实际上，JavaScript 可以支持更多空白符号。

<HT>(或称<TAB>) 是 U+0009，是缩进 TAB 符，也就是字符串中写的 \t 。

<VT>是 U+000B，也就是垂直方向的 TAB 符 \v，这个字符在键盘上很难打出来，所以很少用到。

<FF>是 U+000C，Form Feed，分页符，字符串直接量中写作 \f，现代已经很少有打印源程序的事情发生了，所以这个字符在 JavaScript 源代码中很少用到。

<SP>是 U+0020，就是最普通的空格了。

<NBSP>是 U+00A0，非断行空格，它是 SP 的一个变体，在文字排版中，可以避免因为空格在此处发生断行，其它方面和普通空格完全一样。多数的 JavaScript 编辑环境都会把它当做普通空格（因为一般源代码编辑环境根本就不会自动折行……）。HTML 中，很多人喜欢用的 &nbsp; 最后生成的就是它了。

<ZWNBSP>(旧称<BOM>) 是 U+FEFF，这是 ES5 新加入的空白符，是 Unicode 中的零宽非断行空格，在以 UTF 格式编码的文件中，常常在文件首插入一个额外的 U+FEFF，解

析 UTF 文件的程序可以根据 U+FEFF 的表示方法猜测文件采用哪种 UTF 编码方式。这个字符也叫做“bit order mark”。

此外，JavaScript 支持所有的 Unicode 中的空格分类下的空格，我们可以看下表：

字符	名称
U+0020	SPACE
U+00A0	NO-BREAK SPACE
U+1680	OGHAM SPACE MARK
U+180E	MONGOLIAN VOWEL SEPARATOR
U+2000	EN QUAD
U+2001	EM QUAD
U+2002	EN SPACE
U+2003	EM SPACE
U+2004	THREE-PER-EM SPACE
U+2005	FOUR-PER-EM SPACE
U+2006	SIX-PER-EM SPACE
U+2007	FIGURE SPACE
U+2008	PUNCTUATION SPACE
U+2009	THIN SPACE
U+200A	HAIR SPACE
U+202F	NARROW NO-BREAK SPACE
U+205F	MEDIUM MATHEMATICAL SPACE
U+3000	IDEOGRAPHIC SPACE

很多公司的编码规范要求 JavaScript 源代码控制在 ASCII 范围内，那么，就只有<TAB> <VT> <FF> <SP> <NBSP>五种空白可用了。

### 换行符 LineTerminator

接下来我们来看看换行符，JavaScript 中只提供了 4 种字符作为换行符。

<LF>

<CR>

<LS>

<PS>

其中，<LF>是 U+000A，就是最正常换行符，在字符串中的\n。

<CR>是 U+000D，这个字符真正意义上的“回车”，在字符串中是\r，在一部分 Windows 风格文本编辑器中，换行是两个字符\r\n。

<LS>是 U+2028，是 Unicode 中的行分隔符。<PS>是 U+2029，是 Unicode 中的段落分隔符。

大部分 LineTerminator 在被词法分析器扫描出之后，会被语法分析器丢弃，但是换行符会影响 JavaScript 的两个重要语法特性：自动插入分号和“no line terminator”规则。

## 注释 Comment

JavaScript 的注释分为单行注释和多行注释两种：

```
1 /* MultiLineCommentChars */
2 // SingleLineCommentChars
```

 复制代码

多行注释中允许自由地出现MultiLineNotAsteriskChar，也就是除了\*之外的所有字符。而每一个\*之后，不能出现正斜杠符/。

除了四种 LineTerminator 之外，所有字符都可以作为单行注释。

我们需要注意，多行注释中是否包含换行符号，会对 JavaScript 语法产生影响，对于“no line terminator”规则来说，带换行的多行注释与换行符是等效的。

## 标识符名称 IdentifierName

IdentifierName可以以美元符“\$”、下划线“\_”或者 Unicode 字母开始，除了开始字符以外，IdentifierName中还可以使用 Unicode 中的连接标记、数字、以及连接符号。

IdentifierName的任意字符可以使用 JavaScript 的 Unicode 转义写法，使用 Unicode 转义写法时，没有任何字符限制。

IdentifierName可以是Identifier、NullLiteral、BooleanLiteral或者keyword，在ObjectLiteral中，IdentifierName还可以被直接当做属性名称使用。

仅当不是保留字的时候，IdentifierName会被解析为Identifier。

注意<ZWNJ>和<ZWJ>是 ES5 新加入的两个格式控制字符，它们都是 0 宽的。

我在前面提到了，关键字也属于这个部分，在 JavaScript 中，关键字有：

复制代码

```
1  await break case catch class const continue debugger default delete do else ex|
```

除了上述的内容之外，还有 1 个为了未来使用而保留的关键字：

复制代码

```
1  enum
```

在严格模式下，有一些额外的为未来使用而保留的关键字：

复制代码

```
1  implements package protected interface private public
```

除了这些之外，NullLiteral (null) 和BooleanLiteral (true false) 也是保留字，不能用于Identifier。

### 符号 Punctuator

因为前面提到的除法和正则问题，/ 和 /= 两个运算符被拆分为 DivPunctuator，因为前面提到的字符串模板问题，}也被独立拆分。加在一起，所有符号为：

复制代码


```
1  { ( ) [ ] . ... ; , < > <= >= == != === !== + - * % ** ++ -- << >> >>> & | ^ !
```

### 数字直接量 NumericLiteral

我们来看看今天标题提出的问题，JavaScript 规范中规定的数字直接量可以支持四种写法：十进制数、二进制整数、八进制整数和十六进制整数。


十进制的 Number 可以带小数，小数点前后部分都可以省略，但是不能同时省略，我们看几个例子：

```
1  .01
2  12.
3  12.01
```

 复制代码


这都是合法的数字直接量。这里就有一个问题，也是我们标题提出的问题，我们看一段代码：

```
1  12.toString()
```

 复制代码


这时候12. 会被当做省略了小数点后面部分的数字而看成一个整体，所以我们要想让点单独成为一个 token，就要加入空格，这样写：

```
1  12 .toString()
```

 复制代码


数字直接量还支持科学计数法，例如：

```
1  10.24E+2
2  10.24e-2
3  10.24e2
```

 复制代码

这里 e 后面的部分，只允许使用整数。当以0x 0b 或者0o 开头时，表示特定进制的整数：

```
1  0xFA
2  0o73
3  0b10000
```


 复制代码

上面这几种进制都不支持小数，也不支持科学计数法。

## 字符串直接量 StringLiteral

JavaScript 中的 StringLiteral 支持单引号和双引号两种写法。

```
1      " DoubleStringCharacters "  
2      ' SingleStringCharacters '
```

 复制代码

单双引号的区别仅仅在于写法，在双引号字符串直接量中，双引号必须转义，在单引号字符串直接量中，单引号必须转义。字符串中其他必须转义的字符是\和所有换行符。

JavaScript 中支持四种转义形式，还有一种虽然标准没有定义，但是大部分实现都支持的八进制转义。

第一种是单字符转义。即一个反斜杠\后面跟一个字符这种形式。

有特别意义的字符包括有SingleEscapeCharacter所定义的 9 种，见下表：



转义字符	转义Unicode	产生字符
'	U+0022	"
"	U+0027	'
\	U+005C	\
b	U+0008	<BS>
f	U+000C	<FF>
n	U+000A	<LF>
r	U+000D	<CR>
t	U+0009	<HT>
v	U+000B	<VT>

除了这 9 种字符、数字、x 和 u 以及所有的换行符之外，其它字符经过\转义后都是自身。

## 正则表达式直接量 RegularExpressionLiteral

正则表达式由 Body 和 Flags 两部分组成，例如：

1

/RegularExpressionBody/g

复制代码

其中 Body 部分至少有一个字符，第一个字符不能是 \*（因为 /\* 跟多行注释有词法冲突）。

正则表达式有自己的语法规则，在词法阶段，仅会对它做简单解析。

正则表达式并非机械地见到/就停止，在正则表达式[ ]中的/就会被认为是普通字符。我们可以看一个例子：

```
1  /[ ]/.test("/");
```

除了 \、/ 和 [ 三个字符之外，JavaScript 正则表达式中的字符都是普通字符。

用 \ 和一个非换行符可以组成一个转义，[ ] 中也支持转义。正则表达式中的 flag 在词法阶段不会限制字符。

虽然只有 ig 几个是有效的，但是任何 IdentifierPart (Identifier 中合法的字符) 序列在词法阶段都会被认为是合法的。

## 字符串模板 Template

从语法结构上，Template 是个整体，其中的 \${ } 是并列关系。

但是实际上，在 JavaScript 词法中，包含 \${ } 的 Template，是被拆开分析的，如：

```
1  `a${b}c${d}e`
```

它在 JavaScript 中被认为是：

```
1  `a${  
2  b  
3  }c${  
4  d  
5  }e`
```

它被拆成了五个部分：

``a${` 这个被称为模板头

`}c${` 被称为模板中段

`}e`` 被称为模板尾

`b` 和 `d` 都是普通标识符

实际上，这里的词法分析过程已经跟语法分析深度耦合了。

不过我们学习的时候，大可不必按照标准和引擎工程师这样去理解，可以认为模板就是一个由反引号括起来的、可以在中间插入代码的字符串。

模板支持添加处理函数的写法，这时模板的各段会被拆开，传递给函数当参数：

复制代码

```
1 function f(){
2     console.log(arguments);
3 }
4
5 var a = "world"
6 f`Hello ${a}!`; // ["Hello", "!"], world
```

模板字符串不需要关心大多数字符的转义，但是至少 `${}` 和 ``` 还是需要处理的。

模板中的转义跟字符串几乎完全一样，都是使用 `\`。

## 总结

今天我们一起学习 JavaScript 的词法部分，这部分的内容包括了空白符号、换行符、注释、标识符名称、符号、数字直接量、字符串直接量、正则表达式直接量、字符串模板。掌握词法对我们平时调试代码至关重要。

最后，给你留一个问题：用零宽空格和零宽连接符、零宽非连接符，写一段好玩的代码。你可以给我留言，我们一起讨论。

## 猜你喜欢

# Vue 开发实战

## 从 0 开始搭建大型 Vue 项目

戳此试读



唐金州  
一点资讯前端技术专家  
Ant Design Vue 作者

## 精选留言(26)



为啥不支持直接回复呢？

这里讨论一下@Snow同学的问题 别忘了JS是允许直接写小数的，也就是说12.toString() 他无法分辨你是想要创建一个小数位为toString()的数 还是创建一个12 然后调用toString()这种情况。也就是说 JS里面的. 是拥有两种含义的 一种是小数点 一种是方法调用。 你可以试试12..toString() 这样就可以消除这种歧义

2019-03-24



31



华洛

我真的觉得这些东西已经超出普通前端对于基础的定义了。

2019-03-25



2

27



田野的嘴好冰

零宽空格

```
var a = '\uFEFF',b = 'b', c = 'c', d = (b+a+c);
console.log(d); //bc
console.log(d.length); //3
console.log(d.indexOf(a)); //1
```

2019-03-26



21



曾侃

之前没有接触过零宽字符，学完这节课后网上搜了下零宽字符的应用，看到了这篇文章《[翻译]小心你复制的内容：使用零宽字符将用户名不可见的插入文本中》，受益匪浅。自己用这个思路实现了一样的给字符串添加水印的功能。

代码地址：<https://github.com/zengkan0703/text-watermark>，有不对的地方请同学们指正。

2019-04-10



16



是零壹呀

12.toString() 会被解析成 12.（数字字面量） 和 toString()。  
所以正常的写法是12..toString()才是正常的



Nandy

十进制的number的小数点前后的内容可以省略，但是不能同时省略

.01 = 0.01 10. = 10

12.toString() 12.被当做了一个整体，所以会报错，  
加入空格 12 .toString() 这样.就成为了一个单独的token

嘻嘻~请winter老师表扬我学的认真(#^.^#)

作者回复: 对 表扬。

2019-07-25



1



5



CaveShao

js 中 . 有两种含义，一种是代表一个小数，一种是调用方法。12.toString() 中的 12. 会被浏览器解析为一个省略了小数后面部分的数字。一个数字后面直接写一个方法，就像 333toString 一样，肯定会报错。

Invalid or unexpected token

2019-05-20



3



商志远😁

【理论上，“\${ }”内部可以放任何 JavaScript 表达式代码，而这些代码是以“}”结尾的，也就是说，这部分词法不允许出现“}”运算符。】

这段话没理解

2019-03-19



1



3



Geek\_666

文中的<ZWNBS>(旧称<BOM>) 字符 BOM的全称应该是"byte-order mark"而不是 "bit order mark"吧

2020-03-15



1





Smallfly

`{ }` 的括号中完全可以出现 `}` 符号呀，老师你别骗人哦。

```
`${function(){console.log(1)}}
```

输出：

```
"function(){console.log(1)}"
```

2019-10-17



1



1



better man

转义字符' 产生字符为"是什么意思??? 没看懂，有没有理解的人举个例子解惑下

2019-08-02



3



1



lsy

```
'敏\u200d感词'.length === 4 // true
```

作者回复: 别干坏事啊

2019-07-10



1



一步

正则表达式冲突，这时候对词法分析来说，其实是没有办法处理的，所以 JavaScript 的解决方案是定义两组词法，然后靠语法分析传一个标志给词法分析器，让它来决定使用哪一套词法。

对于这句话我有个疑问，不是先进行词法分析，然后在进行语法分析吗？

难道这里是词法分析分析出来两种，然后在语法分析的选择其中的一种????

2019-03-29



1



Geek\_a55251


`12.toString()` 会把 `12.` 解析成一个整体 `12.toString()` 会把`12`解析成一个整体  
然后按道理可以`12..toString()` 就可以输出`"12"`了



wingsico

全文大概阐述了js中的词法分析中得到的不同类型的token，以及针对js语言特性的一些特殊token（需要根据语法分析来回传递标志来判断具体如何分词），也说了一些零宽空白符号等。但感觉实际使用时，这方面属于比较偏的方面了，但有助于我们去理解编译原理中的词法分析和一些特殊处理，以及对一些特殊场景的错误可以知其原因。

2020-04-13





王益

(12).toString()也可以

2020-03-31





KL宇

零宽空格和零宽连接符、零宽非连接符...感觉打开了新世界的大门。。。。

2020-01-31





小细胞

查到了一篇文章是用零宽空格，零宽非连接符和零宽连接符来抓信息泄露源头的

2019-12-02





小飒

打卡

2019-08-30





伍二娃

```
function f(){
  console.log(arguments);
}
```

```
var a = "world"
f`Hello ${a}!`; // ["Hello ", "!", "world"]
```

2019-06-28

