26-JavaScript词法：为什么12.toString会报错？

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：

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

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，这个我们将会在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   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：

```
/* MultiLineCommentChars */
// SingleLineCommentChars
```

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:

```
await break case catch class const continue debugger default delete do else export extends finally for func
```

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:

```
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:

```
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：

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

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：

```
.01
12.
12.01
```

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：

```
12.toString()
```

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：

```
12 .toString()
```

数字直接量还支持科学计数法，例如：

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

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：

```
0xFA
0o73
0b10000
```

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。

```
    " DoubleStringCharacters "
    ' SingleStringCharacters '
```

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：

```
/RegularExpressionBody/g
```

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：

```
/[/]/.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：

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

它在JavaScript中被认为是：

```
`a${
b
}c${
d
}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：

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

var a = "world"
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

精选留言：

- Snow同學 2019-03-24 05:24:33
  只有我看完，还是不知道12.toString()为什么会报错嘛？ [28赞]

- 🐺🔫🐱 2019-03-24 21:44:47
  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 [4赞]

- 田野的嘴好冰 2019-03-26 21:07:18
  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 [3赞]

- 华洛 2019-03-25 19:13:01
  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。 [3赞]

- 商志远 2019-03-19 09:25:55
  【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 [3赞]

- 水儿涵涵 2019-03-20 15:22:59
  老师好，前端工作一年多，需要学一门后端语言吗？我是想把精力放到前端上，但是现在很多公司都要求熟练一门后端语言，但是工作又用不上后端语言，现在有点纠结。希望给点建议，感谢！ [2赞]

- 曾侃 2019-04-10 15:55:01
  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。 [1赞]

- 一步 2019-03-29 08:13:22
  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。

  对于这句活我有个疑问，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？？？？？ [1赞]

- 0xAC7 2019-04-14 18:13:46
  @Snow同學

  只有我看完，还是不知道12.toString()为什么会报错嘛？

  我的理解是，12.toString() 前面的“12.”（注意有点）对于引擎来讲是有歧义的，这到底是代表浮点数，还是代表要转为数字的包装对象，然后调用 toString 方法呢？引擎默认代表是浮点数，这个时候如果要调用 toString 方法，需要再加一个点，像这样：12..toString()。
  顺便说一句，中文也有很多歧义呢。

- Sherry 2019-04-12 09:42:59
  一篇不落看到现在，真心想说一句，这些知识，对于我们工作中做开发，或者进阶，或者成长为能独当一面的大神，真的有用吗……自己一直很重视基础，只是，老师讲的这些东西，不止感官上觉得比较偏，而且感觉逻辑性真的有点弱啊……知识哪怕再偏再难，系统性讲出来，讲清楚，也会看得懂吧。但是到现在，我的感觉是，一头雾水…3年多的开发经验，我怕是个假的前端😊

- 桂马 2019-04-02 00:42:19
  经典的USD.replace(/\B(?=(\d{3})+$)/g,',')

- 一步 2019-03-29 08:29:34
  @商志远😀
  你可以尝试一下在控制台输入：`test } ${}` 看看会发生什么？

  Uncaught SyntaxError: Unexpected token }

- Skyling 2019-03-21 09:30:57
  重学前端是夯实前端基础，那前端进阶方向在哪里?还是一定要修一门后端语言扩展服务端，希望老师可以指点迷津😊

- leslee 2019-03-19 20:25:45

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：…… 有点蒙